-

COMPUTER SCIENCE NEA

Shayaan Rashid - LDE Learner

CONTENTS

ANALYSIS

SYNOPSIS

The problem I have identified is the lack of gaming applications available for those with limited (temporary or permanent) access to hardware. The majority of games are dependent on an internet connection and those that aren't are commonly either GPU (Graphics Processing Unit) intensive (like God of War or Cyberpunk) or simply easily runnable; of those that are runnable, the majority are games intended to be played as full games with purpose/story (examples include Hollow Knight or Ori). My goal is to create a game that is simple and mainly played to pass time (whether it be until the players internet returns, or just to pass time on their laptop in places like an airport). The goal of the game is that there isn't a goal, and despite there being levels/different simulations, there is no end state in which the game is considered "complete" or "maxed out." In order to do this, I will avoid adding any such items in the game that increase performance, as a player may consider the game complete when all are bought. I will also have levels be configurable by the player, so that there is no "final level" which the player may assume is the epitome of the game, in which finishing would result in the games completion. Some existing solutions to this problem are Tetris, Skate, Plague Inc and Doodle Jump (Note that some of these are mobile only, whereas my game will target the PC market).

HOW IS THE PROBLEM SOLVABLE BY COMPUTATIONAL METHODS AND WHY?

ABSTRACTION

JUSTIFYING MY USE OF ABSTRACTION

My solution, as it needs to be run-on low-end devices, will be abstracted heavily. This not only helps it run smoother but also makes it easier for players to pick up and understand since there won't be misleading/confusing graphics.

DESCRIBING MY USE OF ABSTRACTION

Some of the ways in which my game will be abstracted include:

- Animations, these will be done using sprites. This helps the game run smoother as well as meets the demands of my stakeholders who have specified their interest in a pixelated game. This is an abstraction as quality is being decreased heavily (will not look as detailed as real life)
- Pause buttons, this abstraction allows players to stop the game at any time (which isn't possible in real life). The consequence of this is that players are able to put the game aside in order to attend to other matters such as housework or calls.
- Score, this will be a measure of an individual players success in the game, which increases as enemies are killed. This is an abstraction as in real life there is no such counter which measures success in scenarios.

THINKING AHEAD

JUSTIFYING MY USE OF THINKING AHEAD

Another computational method is thinking ahead; By thinking ahead, I can confront potential problems in my game before they appear, reducing the amount of maintenance and correction I need to do to my solution. Some foreseeable problems include:

DESCRIBING WAYS I HAVE THOUGHT AHEAD

- What happens if a user attempts to do an attack in a state where they shouldn't be able to (e.g., Dashing), this may disrupt the animation and sprites, as well as also cause unintentional collisions which can break the game.
- What happens when a user dashes/moves into a crevice, it may result in them glitching through/outside the map.
- Whether or not the score should increase if the player dies at the same time as the enemy. Should the score increase or should a completely alternative route be taken by adding mechanics such as parries.

THINKING PROCEDURALLY

JUSTIFYING MY USE OF PROCEDURAL THINKING

Lastly, thinking procedurally as well as decomposition can aid in the design and planning of my solution. Breaking down the problem into smaller pieces means that I can work on multiple things at once, and therefore my efficiency is increased (I can work on other modules if I'm stuck on one). I can focus on separate parts at a time making it easier for me to focus on the individual efficiency of modules and ensure they can be reused.

DESCRIBING MY PROCEDURAL THOUGHT PROCESS

I will use a modular design to break down my game into several problems/aspects:

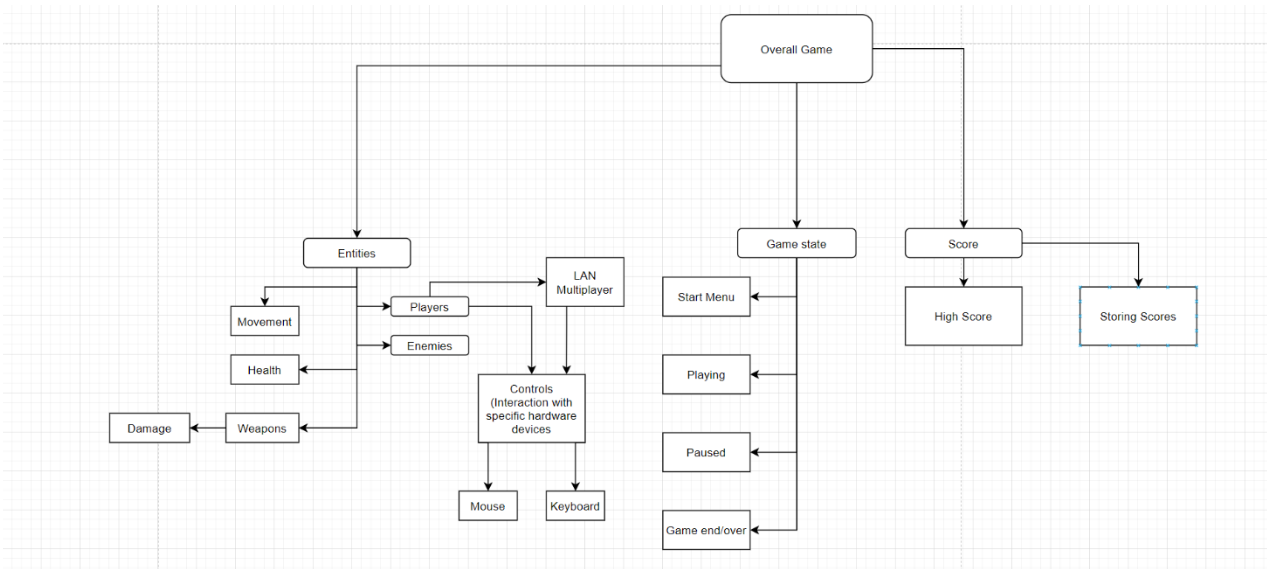


Figure 1: Diagram representing the game as sections (or modules)

I have split the game into 3 essential components:

- Entities
  - These are the beings (objects with behaviour) in my game. Aside from the player[s], there will be enemies with set movement and attacks. If the player and the enemies' sprites meet, there should be a collision where one of the beings take damage.
- Game state
  - There will be 4 game states:

        - Start menu, where the player begins the game or alters settings (which will also be stored in a database using SQL)
        - Playing, the main state of the game where the player is fighting the enemies
        - Paused, where the game is frozen, and the player can either quit or alter settings
        - Game end/over, where the game is over and results are displayed, the only place to go from here is the start menu
- Score
    - The main purpose of the score is to indicate the players success relative to a base starting point (there is no end number). This score will accumulate as enemies are killed based on the strength of the enemy. Scores will be stored in a database using SQL.

THINKING LOGICALLY

JUSTIFYING THE IMPORTANCE OF THINKING LOGICALLY

This allows me to consider logical parts of my problem. It helps me ensure that I'm tackling certain problems correctly and ensures that everything in the game "makes sense" (within its scope of a video game).

DESCRIBING WHERE I SHOULD THINK LOGICALLY.

There are a variety of parts that require me to think logically, one of the main parts being how collisions should be handled. Also, questions like how my character should react to being attacked, and how the character as well as the enemy should react if both attack at the same time. Other things to consider include how the menu buttons will work, which conditions should set the game over state, and how the character should behave if it collides with a second player (if this is implemented) etcetera.

THINKING CONCURRENTLY

JUSTIFYING THE IMPORTANCE OF THINKING CONCURRENTLY

Lastly, it is important to acknowledge ways that I can make my solution run faster or more efficiently (by consuming less resources). This ensures that my target market is able to play the game (given their specs) with the best experience, and also creates space for me to add new features without worrying too much about the impact on performance.

DESCRIBING PROBLEMS THAT CAN BE SOLVED BY THINKING CONCURRENTLY

Ideas involving thinking concurrently include:

- How to efficiently increment the score and animate the screen at the same time
- How to properly draw two players (if this feature is added) without redrawing the screen at twice the rate.

WHO ARE THE STAKEHOLDERS?

The stakeholders are people who are:

1. Unable to access internet or efficient hardware (this could be due to a variety of reasons, ranging from harsh living conditions to outdated hardware)
2. Temporarily unable to access internet OR strong hardware (The user may be at an airport, or they simply may not have a functioning router for the moment)

Although the solution is a game, the stakeholders are not necessarily gamers, the game should be simple enough for anyone of ages 12+ to play. The same applies for the target market, people with high end computers are still able to play the game, however the game won't be designed to revolve around their stronger hardware.

The stakeholders would make use of the solution in their spare time, the game is designed to be easy to pick up and play, as well as easy to put down/temporarily pause. The solution intends to be appropriate to their needs by being runnable on low end pcs, as well as being a game that doesn't require consistent time/effort and an internet connection.

QUESTIONNAIRE

To illustrate the demand for such product, I have used convenience sampling and sent a questionnaire to the public:

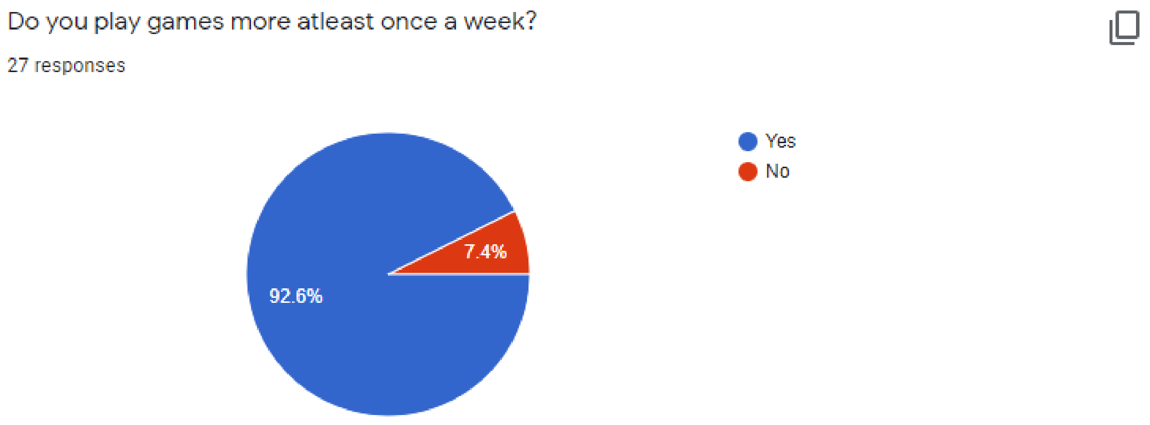


Figure 2: Initial question to ensure that people answering are people who play games and are likely to be interested in my solution

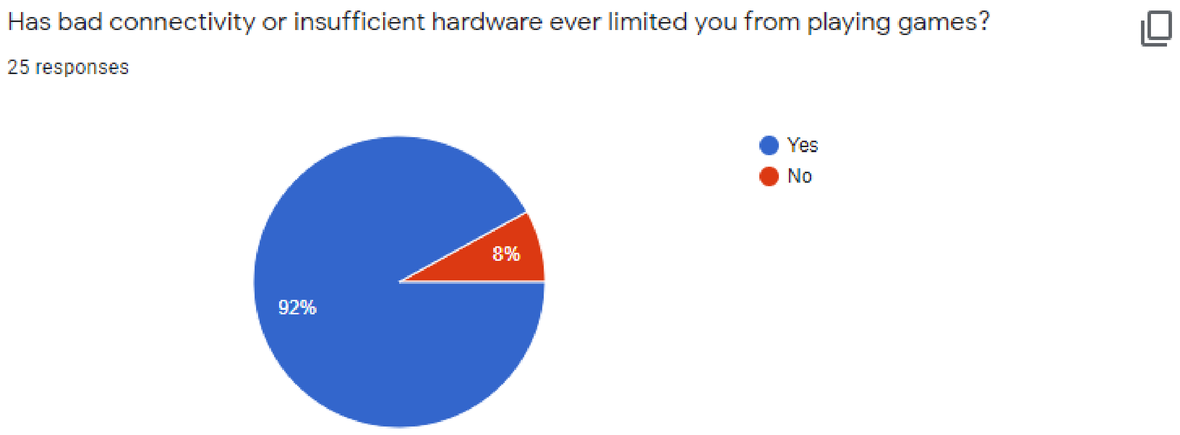
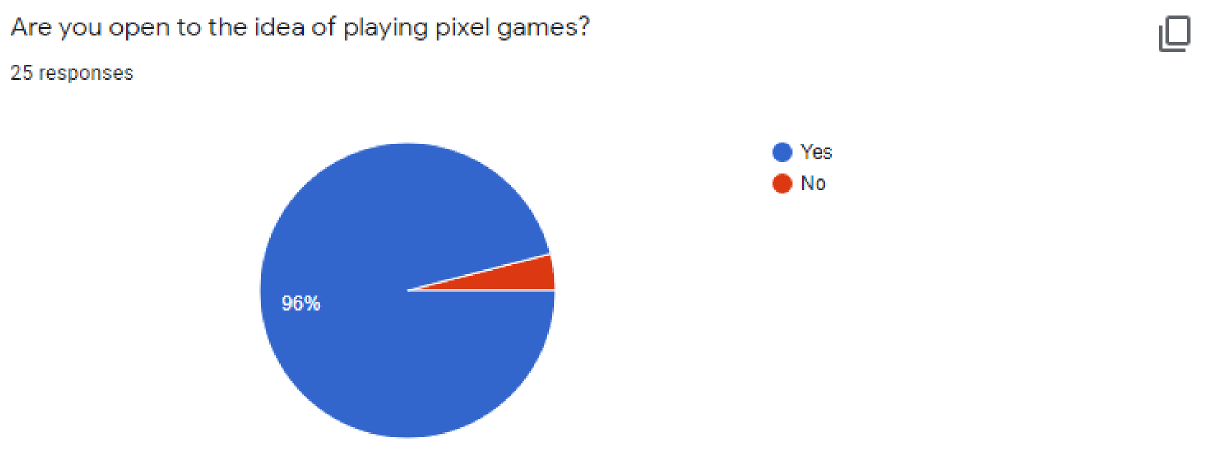
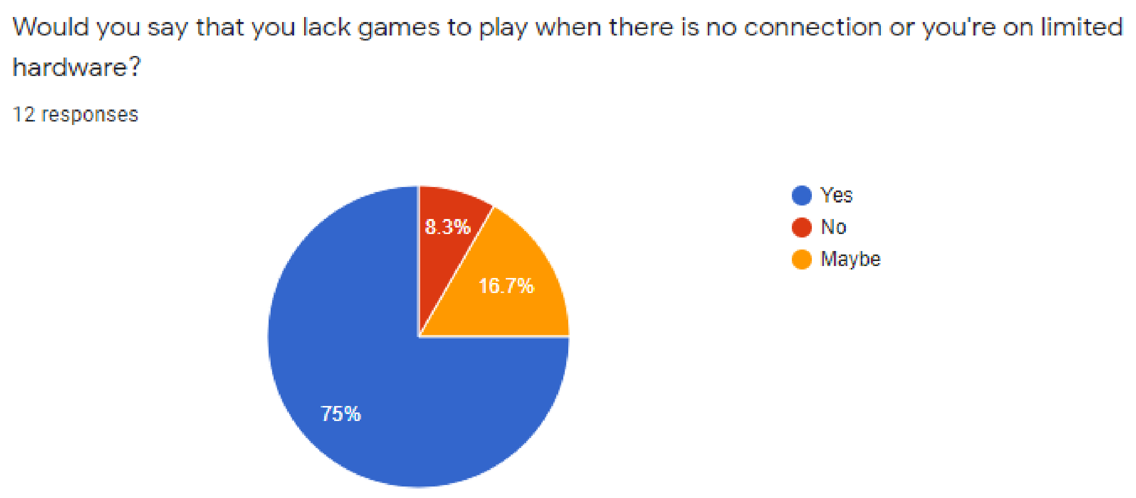


Figure 3: Illustration of the disruption caused by connectivity and hardware


Are you open to the idea of playing pixel games?

25 responses

Figure 4: Highlights the style of game I plan on making and checking whether survey recipients would be interested in the game


Would you say that you lack games to play when there is no connection or you're on limited hardware?

12 responses

Figure 5: Most important question, displaying the demand for my potential solution

These results show that there many people are affected by problems when playing games, and so there is an evident need for the solution I'm proposing. My potential solution is appropriate to my stakeholders need as 75% of people have said that they lack games to play given the problem that I am trying to solve. Furthermore, people can also make use of my proposed solution outside its intended use (e.g., While a game is downloading, or internet drivers are updating etc.) which further outlines the need for my proposal.

ARE THERE SIMILAR SOLUTIONS?

Similar solutions include Dinosaur Game, Solitaire, Tetris, Skate, Plague Inc, Doodle Jump, Stardew Valley (to an extent), Slope (although it's a web game therefore demanding internet) and Snake; Dinosaur Game by Google is one of the closest solutions to mine, functioning as a browser only game that is able to be run when no internet connection is found.

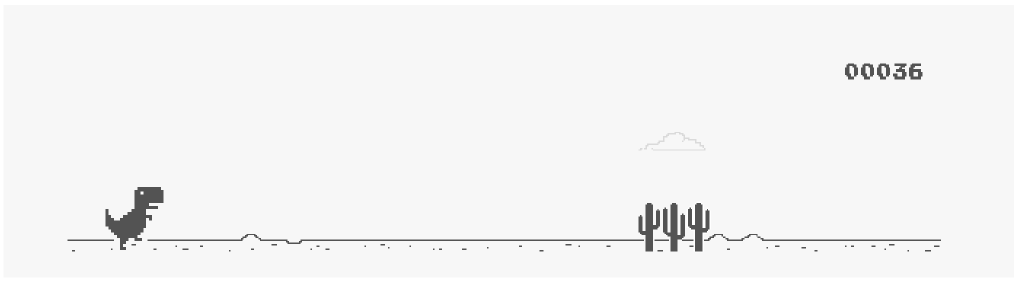


Figure 6: Google's dinosaur game which utilises only a single key to play the game

Based on these solutions, my game will be an installable game available on a platform like Steam or Itch.io. This is because Google's dinosaur game is exceedingly popular and meets the demands of my stakeholders if they were to be on a browser. Further, I will try and keep my game as simple as possible with basic sprites (following the approach of the above examples) as the above solutions has proved this to be the best way to create the most efficient solution for the problem of hardware.

It is important to note that some of these games, despite fulfilling the problems (or a subset of them), can be complete in the sense of gaining the highest score. This is one of the features I would like my game to inherit as it promotes competitiveness and helps the game spread among peers.

As for the style of my game, I would like to follow a 2d platformer style approach, with combat mechanics similar to that of Hollow Knight (A famously known Metroid Vania). This style of combat is successful as it heavily depends on the player's skill and the challenging fights (that mainly depend on the player itself instead of external factors) that result from this are heavily demanded by gamers.

Figure 7: Hollow Knight's mechanics revolve around a simple slash and dash style, with extra moves which require energy that can be built from dealing damage. Hollow Knight's massive popularity shows the success of this fighting style and justifies my similar approach to my problem.

As Hollow Knight's mechanics are an inspiration to my combat design, I have surveyed peers who have played Hollow Knight in order to understand, in more detail, the parts they loved the most about Hollow Knights combat mechanics. One of the notable features of hollow knight is the manoeuvrability as well as the range of attack/movement, this will therefore be a key component and main factor to consider in the approach of creating my game.

SUCCESS CRITERIA

WHAT ARE THE ESSENTIAL FEATURES AND LIMITATIONS OF THE SOLUTION?

The essential features of the game are as such that it must be:

- Runnable on practically any device
  - This means that the user should be able to run the game regardless of how weak their computer is, and that the game should run smoothly without any frame drops or bad quality gameplay
    - This is so that it meets the needs of the target market, which is that it can be run-on low-end devices that may not have the best hardware.
- Runnable without internet
  - This means that the user can run the game regardless of location, and won't need any sort of connection to the internet. (The user may require internet for the initial download of the game; however, this is a one-time requirement. There are ways this requirement can be circumvented e.g., Selling the game on a USB locally such that a user just needs to plug in the USB and run the game)
    - Similar to above, this is so that the game is available to all who have a computer, therefore meeting the needs of the target market.

The game has limitations such that it will not be able to be:

- Run with online multiplayer compatibility.
  - This means that players will not be able to play/see each other in their games. And that there will be no way for players to connect with each other through my game.
    - This is because my main target market will likely not have access to internet, however I will consider making the game function online as a bonus option after the main criteria has been met.
- Graphically intense (realistic)
  - This means that the game will aim to avoid straining the GPU. The game will not look flashy or detailed as a result of this (although a level of basic design would still be necessary in order to make the game enjoyable)
    - This is because my target market won't have the strongest computing power.
- Run without a keyboard/mouse
  - This means the game will depend on inputs from a keyboard or/and a mouse
    - This is because the fundamental movement requires input from keyboard and menu selection may require input from mouse (I may consider making the game entirely keyboard as this feature can reach even more of my target market)

WHAT ARE THE REQUIREMENTS?

The requirements for the game include:

- Having a functioning computer with any functioning OS
  - This means the hardware parts of the computer must be intact, and the operating system must work as intended; the computer would need at least 2 gigabytes of ram (in order for the OS to run), a monitor bigger than 300x200 (game resolution), compatible CPU/GPU devices, a functioning mouse and keyboard. In regards to operating system, it must be one that can run EXE files, examples including Windows or Linux.
    - The computer must function as there would be no way to open the game or have it interact with components otherwise. A functioning OS is also necessary as the game requires it as an intermediary so memory can be used, hardware inputs can be recognised, monitor outputs can be displayed etc.

- Having a working keyboard, mouse[1] and monitor
    - So that the player can interact/see the game
- Having enough space for the game to run.
    - To avoid crashes/lagging (this space is not expected to be much), the user can run the game on an external storage device (e.g., a USB) however they will need to have enough main memory (RAM).

As intended, the hardware and software requirements are extremely small, in order to be considerate for the hardware possessed by the target user.

In terms of game expectations, as my game will develop/stem from the needs of users during testing and other types of feedback, there are not many exact requirements I can set. The main measurable/checkable criteria set for the game is that is develops to be:

- A 2 dimensional fighting game
    - Can be checked by simply playing the game
- Carries on endlessly (unless the player dies)
    - Can be checked by trying to last as long as possible (or editing code to make player invincible) and seeing if the game keeps going
- Has a score counter
    - Can be checked in the end phase, score should be visible if a score counter was in place.
- Has a player that can fight enemies
    - Can be checked by playing the main phase and attempting to interact near enemies (if they exist)

The requirements for the user include:

- Being over 12
    - In order to properly understand how the game's concepts work, as well as ensure that violent moves in the game are not reciprocated by children who may not fully understand the game.
- Able to correctly use a computer, as well as a keyboard and mouse
    - This is so that the user can actually play the game as it is intended to be played

HOW WILL SUCCESS BE MEASURED?

Success will be measured mainly via black box testing, as well as reviews and feedback from spectators who are watching gameplay. The satisfaction of my peers and others testing the game will prove that the problem has been solved. Furthermore, checking my solution against my initial criteria (requirements and essential features) is proof that my solution is viable and properly applicable to my given problem.
These seem to be the best ways to measure success as black box testing depends on user satisfaction, and user satisfaction is my main goal in creating this solution. Testing against criteria similarly ensures the solution does everything the user would like for it to do.

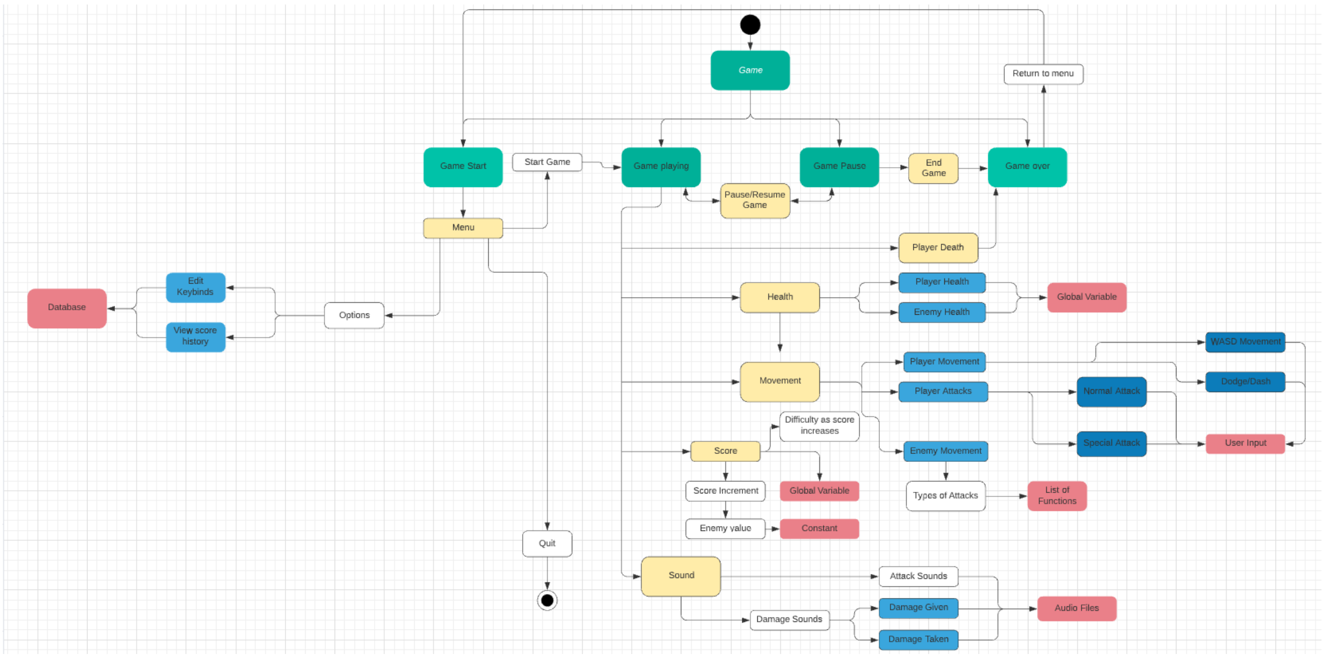
[1] Mousepad can also be used, although experience may be worse.

DESIGN

All designed components will be hyperlinked for download via GitHub.

PROBLEM BREAKDOWN

I will begin by breaking down my problem into distinct parts/modules, as I have done when thinking procedurally.



I have taken a top-down module approach as it allows me to split the game into manageable components, which makes the game a lot easier to code as I can develop algorithms independently and put them together in the end. I can also look at specific components in detail and use "thinking ahead" to understand the parameters and returns of functions and the results I want from my algorithms. "Thinking logically" also helps me with this approach as I can understand how components are linked and what components will be used depending on choices made by the user (e.g., inputs for directions)

Furthermore, because the game is split into components, I can now work on all game states at once, using agile development strategies. This also means that if I get stuck with any particular problem I can move onto a different problem while brainstorming and therefore be more efficient with my time.

I can now refer to my design through multiple stages of development to look at what I should work on. I can also compare the initial design to my final result in order to see what is missing or not fully developed.

COMPONENT ANALYSIS AND ALGORITHM DESIGN

GAME START

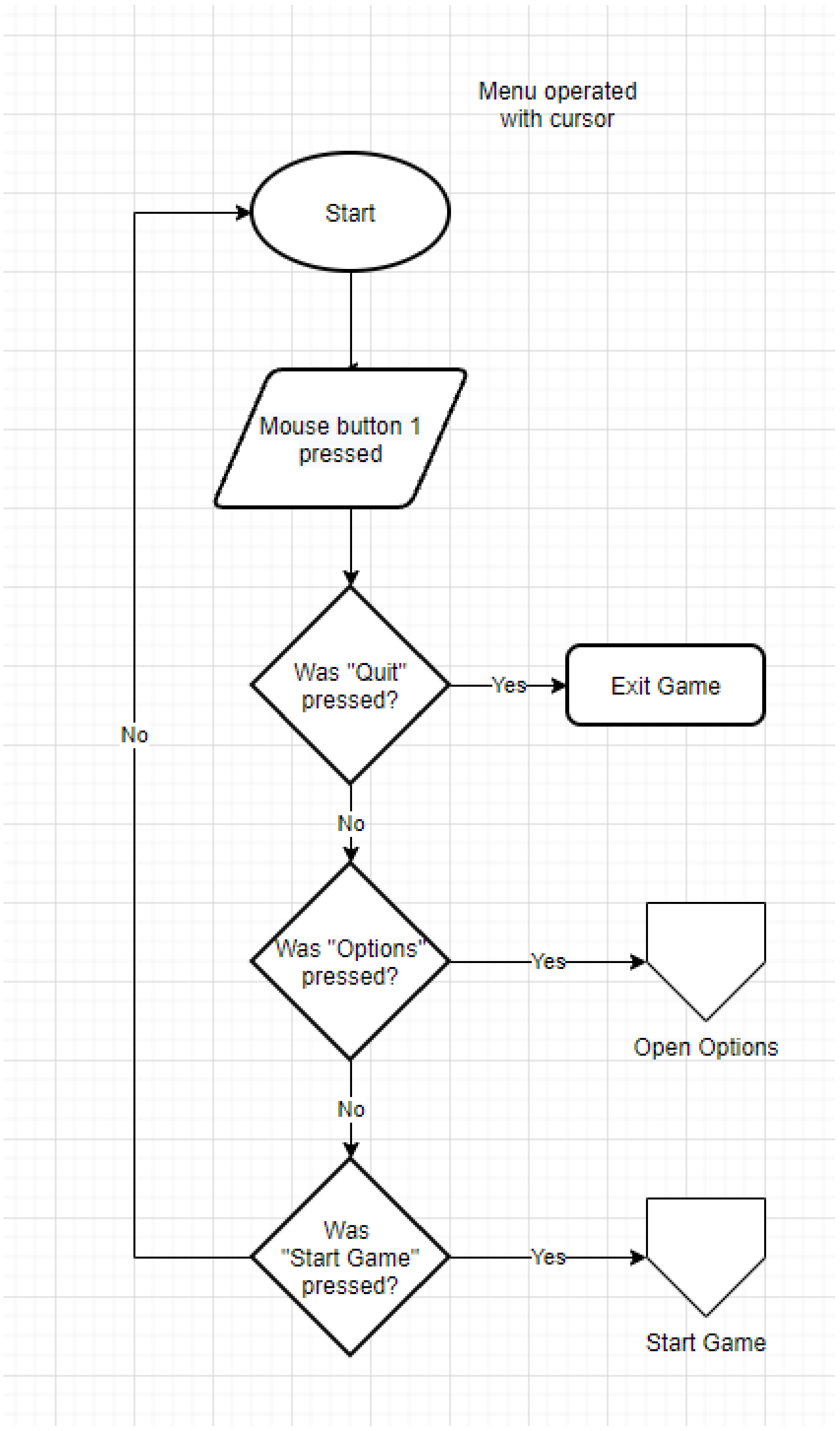
I need a game start menu in order for players to understand the idea of the game and fully prepare themselves before they start, it also sets the atmosphere via the background and music.

In my game start screen, there will be few components. This screen will mainly consist of an animated welcome screen and my menu.

If the application crashes/is force closed, there will be no impact on the game, and it will open from the beginning again upon start-up.

MENU

From the menu the user can either begin the game, go to options, or quit.



The menu screen will have the past scores of the user on the top right corner so they can see their progress.

Upon selecting “options “, the user can select values for key binds and update them. I will store and update the key binds via a database using SQL.

When “quit “is selected the application will close.

If the user selects “Start Game” the game will begin, this part of my solution will be addressed in the Game Playing section.

The navigation will be done with the cursor; however, I plan to create a custom cursor in order to make the game more aesthetically pleasing.

My justification of designing a menu system in this way is that it is easy to use and doesn't introduce any complex features at the beginning of the game. So that newcomers are able to quickly understand where they can find what they need.

GAME PLAYING

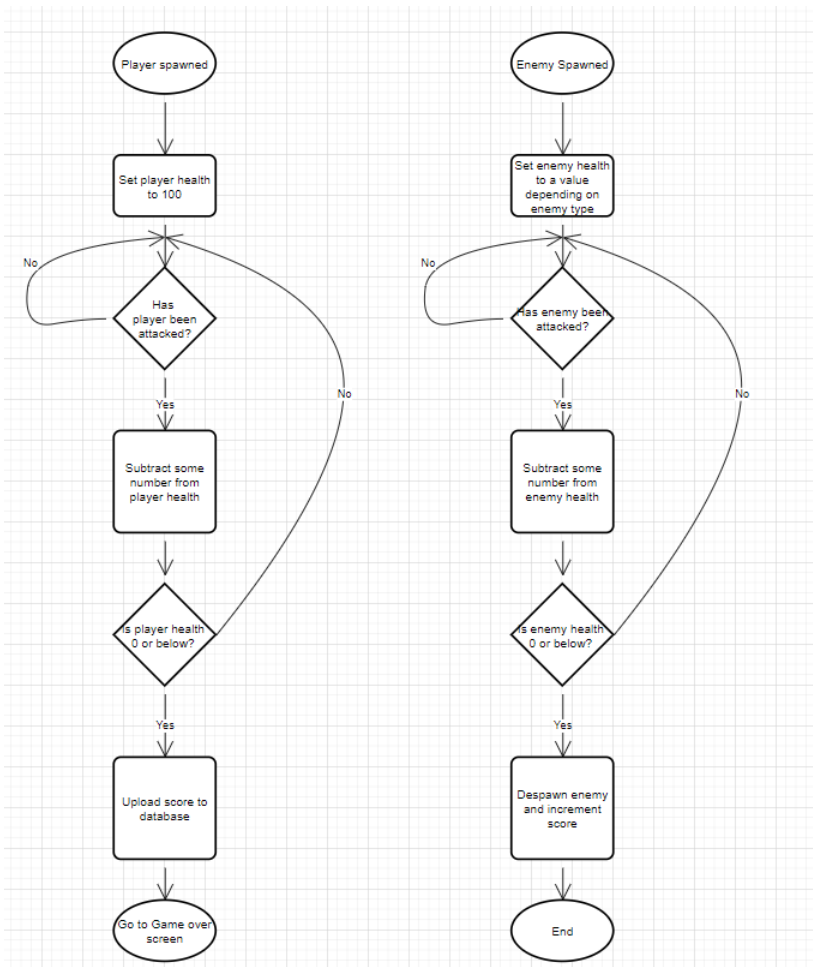
This will be where the majority of the users time is spent, therefore this is the most essential part of my solution.

If the application crashes/force closes here, the players progress will be lost and when the application is run it will start at the Game Start menu, for this reason the user should pause the game and end it manually to save their score.

HEALTH

Health is a critical part in any fighting game, adding health makes the game more serious for the player as they know the game will end/they will die if their performance isn't good enough. This helps create competition between peers on who a better survivalist is.

The players health will always start at a constant value of 100HP, this health will decrease if the player takes damage due to an attack. If the players health reaches 0, the game will end, and the user will be brought to the game over screen.
Similarly, different enemies' health values will be defined by a variety of constants depending on the enemy, and if this enemy takes damage due to a player attack, it will decrease. If the enemy's health reaches 0, some amount of score will be awarded to the player and the enemy itself will die (disappear from the screen).


Player spawned
Set player health to 100
No
Has player been attacked?
Yes
No
Subtract some number from player health
Is player health 0 or below?
Yes
Upload score to database
Go to Game over screen
Enemy Spawned
Set enemy health to a value depending on enemy type
No
Has enemy been attacked?
Yes
No
Subtract some number from enemy health
Is enemy health 0 or below?
Yes
Despawn enemy and increment score
End

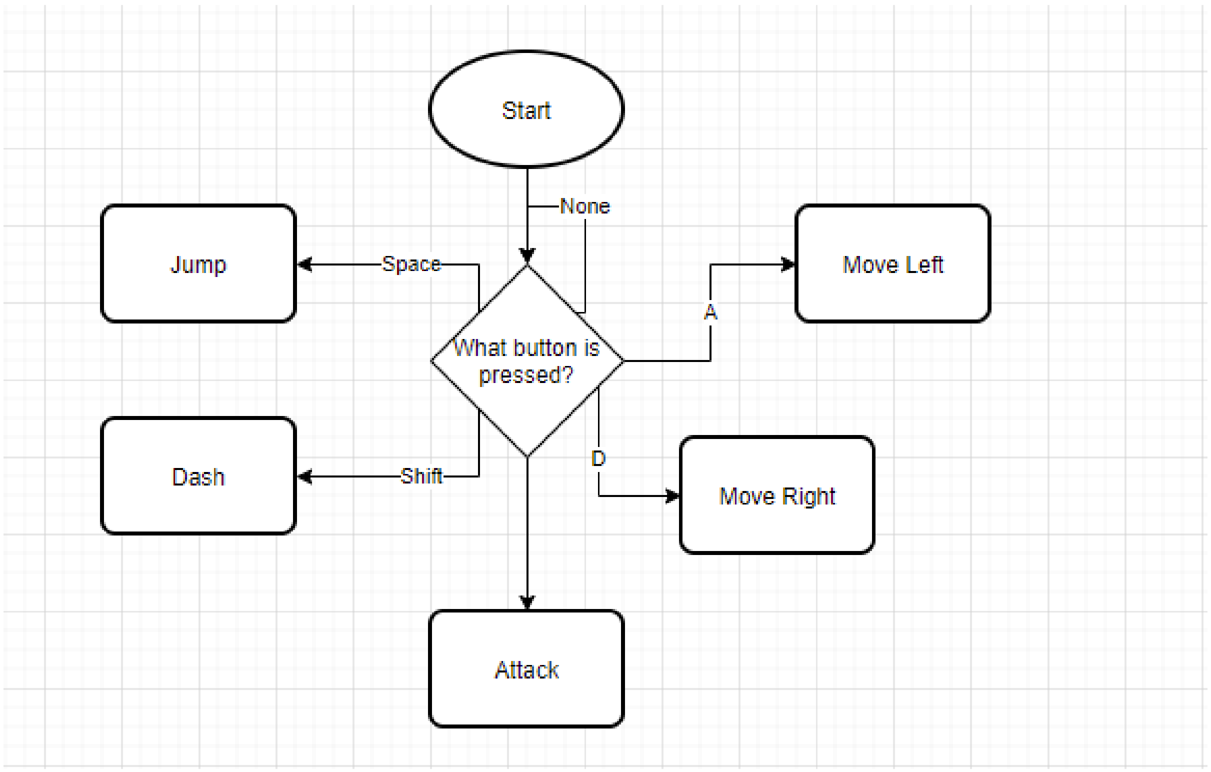
MOVEMENT

Movement is arguably one of the most important parts of the game, I need enjoyable movement that is free from any errors/glitches in order for my target market to have a good experience in the game.

Movement for the player will be rather rigid and will not fully obey laws physics, this means that they will be able to change directions in the air in order for increased versatility for evasion. Furthermore, I also intend on adding special movement like dashes and wall jumps. The player will have a basic direction slash that will depend on which keys are being pressed (or which direction is being faced)

The enemies will all have their own styles of movement, some will fly and will be grounded. This movement will also affect how they'll attack.

Since attacks from the enemy will be mainly randomised, there is no use to visualising it in a flowchart. Therefore, I will only represent player movements.

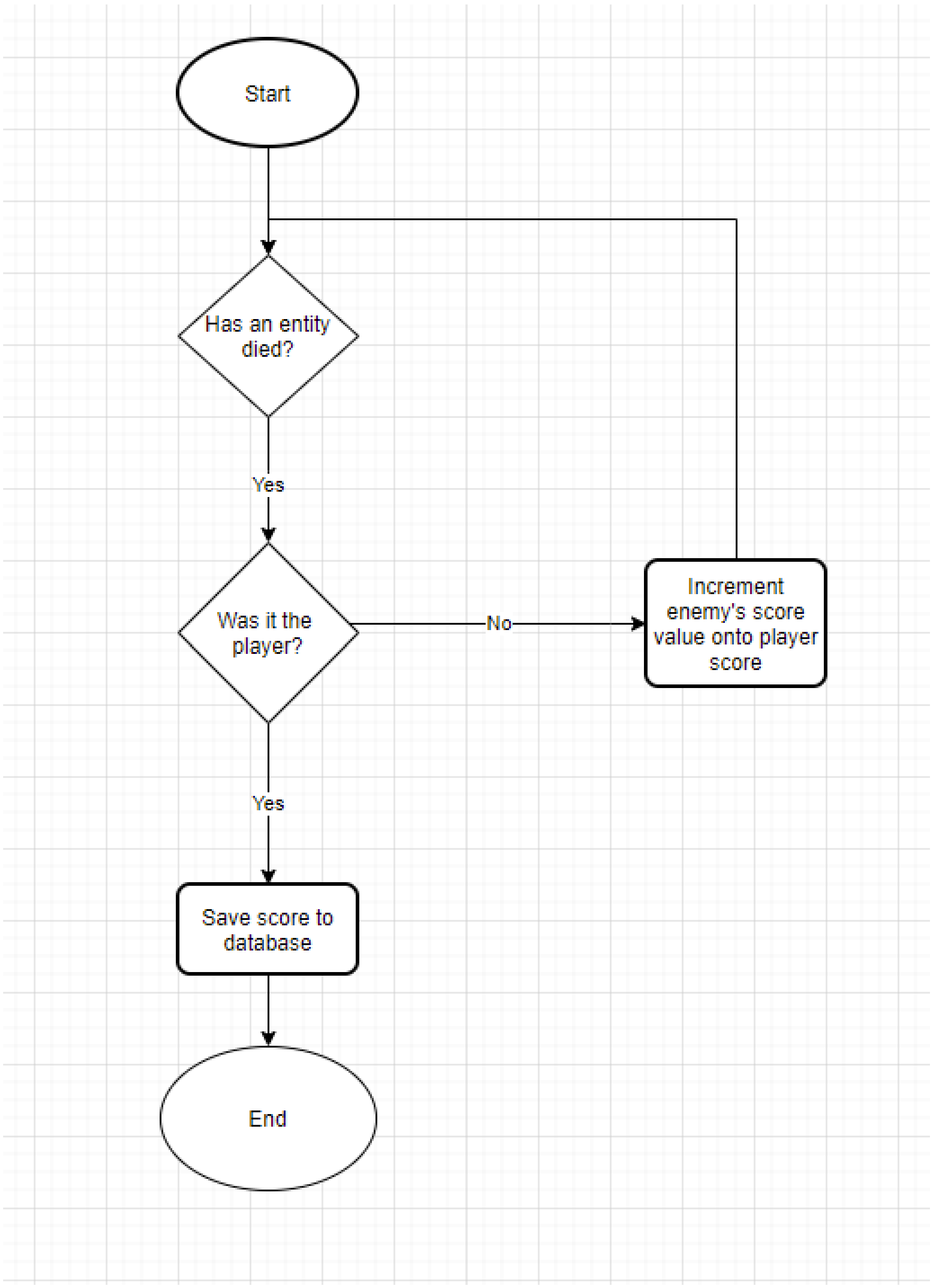


I have attached all possible buttons to one decision as I feel that having one button per decision incorrectly implies that the player can only do one type of movement at a time. I may add more attack types after the basic types of movement are added.

SCORE

The score is one of my measurable criteria, it is important for users to have a score system so that they can compare with friends and therefore the score system adds a sense of community to my game. Furthermore, having a score system makes the game competitive and encourages users to play the game more, in an attempt to get a higher score.

The score will be visualised in the top left/right corner of the screen and will be incremented based on a kill.



I have made this extremely simple as it shouldn't be hard for a player to understand where they're getting their score from. And this helps identify the object of the game for the player.

SOUND

Having sound is important to creating ambience in a game, as without it the player won't be pulled into the game as effectively.

The sound also won't require visualisation, as a sound will simply be played according to an event that happens e.g., If the player attacks than the slash sound will play. In this sense the flowchart would hypothetically resemble the movement diagram except the process would be to play a sound instead of moving.

There may also be background music during the game, which will be sourced from some Copyright free sources due to the time constraint (If I have time, I may implement my own music however I must ensure the main product is complete before adding extra features).

PAUSE

Having a pause system means that players can leave my game whenever they need to, for whatever reason. It gives players reason to start my game as they know they can stop it if something pops up and they need to divert their attention.

Upon pressing the escape key, the Game Pause screen will be shown.

GAME PAUSE

In this screen there will only be 2 options:

- Resume Game, which sends the player back to the Game Playing screen with their progressed resumed
- End Game, which, after prompting further confirmation to avoid accidents, will send the player to the Game Over screen and save their current score.

If the application crashes/is force closed here, the player will lose their progress and the application will start at the Game Start screen when ran again. This means that it is in the players best interest to use the End Game button (unless for some reason they would not like their score saved).

GAME OVER

On the game over screen, graphics will be displayed as well as the score. If this score is the highest the user has, it will also display a congratulations for reaching the score. There will be one button which sends the user back to the Game Start screen,

COMPONENT DESIGN

I need to design the player, 3 types of enemies, a boss, the map, the game

GAME START

CURSOR DESIGN

The cursor needs a style that fits the theme as well as the colour scheme, pygame currently only supports black and white cursors therefore my cursor will also be in black and white.

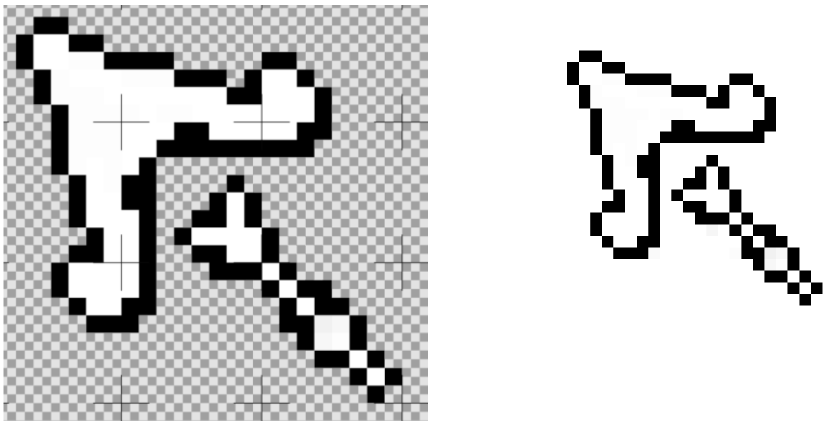


Figure 8: Cursor, created using Aseprite
This will be what the user sees (replacing the default cursor) when interacting with the UI.

Right now, these are .cur files, however I'll need to convert them to .xbm files so that pygame can use them.

I've used this design as I feel it stands out from the background so that the user can easily see the cursor, and still somewhat resembles an original cursor so the user will know what it is.

MAIN SCREEN

This menu will have the game name text, the past scores, and the options to play the game or edit key binds.

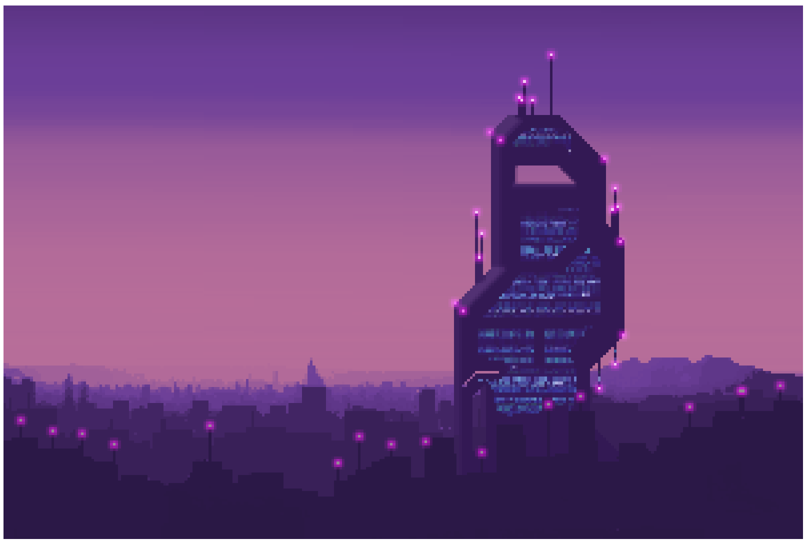


Figure 9: Background for start screen, created using Photopea
When a user clicks a button on this screen, it'll jump to the options/start game depending on what is pressed.

This retro building design fits the blocky style of my game. This is contrasted by the use of neon lights which make the game seem futuristic.

The Game name as well as the options will be available on the left, the past scores will either be visible on the right of the building or in its own option.

MENU INTERFACE

This will be a small interface with all the movement and ways to change the key binds.

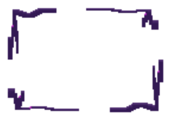


Figure 10: Transparent borders for the options to be held inside, as there is a gradient background this is the easiest and nicest way to maintain a good look without sudden changes in colour. This was created using Aseprite

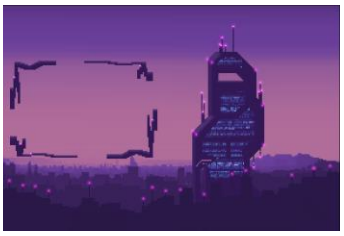


Figure 11: Interface on top of the background

This block should be big enough to hold options regarding the movement controls (Left, Right, Jump, Dash, Attack etc.) as well as change audio levels.

This interface may also host the past scores.

AUDIO TRACK

GAME PLAYING

Most the design will be for this part, as I will need to visualise fighting and score incrementing. However, a lot of the design in this game will not be creating using design tools, but rather through code which uses a variety of objects.

AUDIO

This will contain the audio that plays during the main phase of my game.

<u>Evaluation Note:</u> This section of my game was not completed due to time constraints, I talk about this more in future improvements.

BACKGROUND TRACK

The background track will be the song that goes on in the back of the game permanently

SOUNDS

The sounds will be tracks that are played based on events

SPRITES

These will be the designs for the player and the enemies.

SKETCH DESIGN

These designs will be simple unanimated low effort sprites which will simply be used to test how well the shape looks when encountering collisions and other details.

After using these sprites and understanding how well the design performs, I will return and add proper detail to the sprites.

PLAYER

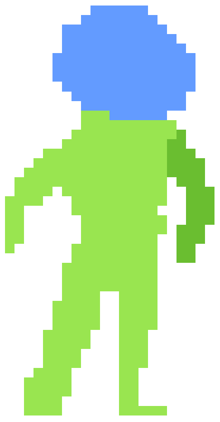


Figure 12: The player will be a girl with rather ordinary features, however, will have some extraordinary capabilities.
The player will be able to use when the user interacts using the keyboard keys that they have chosen (defaulted to WASD).

ENEMIES

The enemies are likely to be the only objects that are in a class

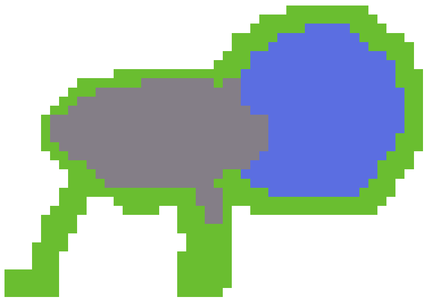


Figure 13: This test enemy will resemble a dog, although it is nothing of human nature. It has only 2 legs and it will attack with its head
The user can interact with this entity through collisions, where either the player or this entity will take damage

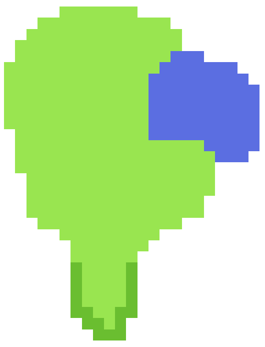


Figure 14: The second enemy will be an unfamiliar animal with one leg, almost acting like a pogo stick. This enemy will also attack using its blue area
The user can interact with this entity through collisions, where either the player or this entity will take damage

FINAL SPRITE

Each sprite will also need animations for running, jumping, and attacking. The player will also need an idle animation.

PLAYER

ENEMIES

BACKGROUND / MAP

This will be the field where most action takes place. I will use a tile set and generate a world using python rather than the design software used in game start

SCORE COUNTER

The score counter will need to be displayed in a font that matches the game theme.

I will be using the below font called Thaleah Fat

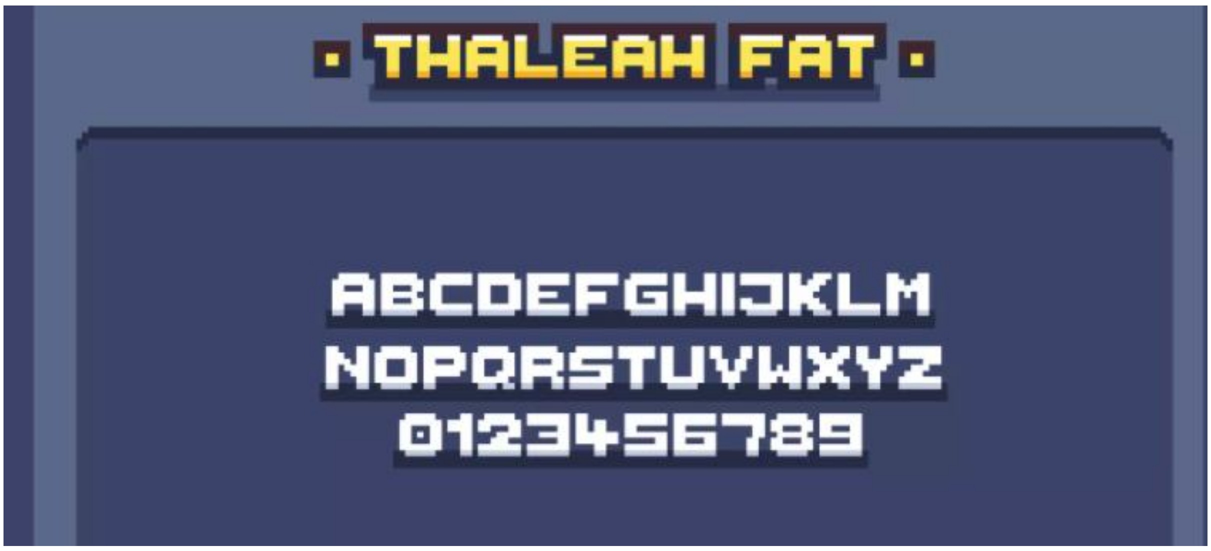


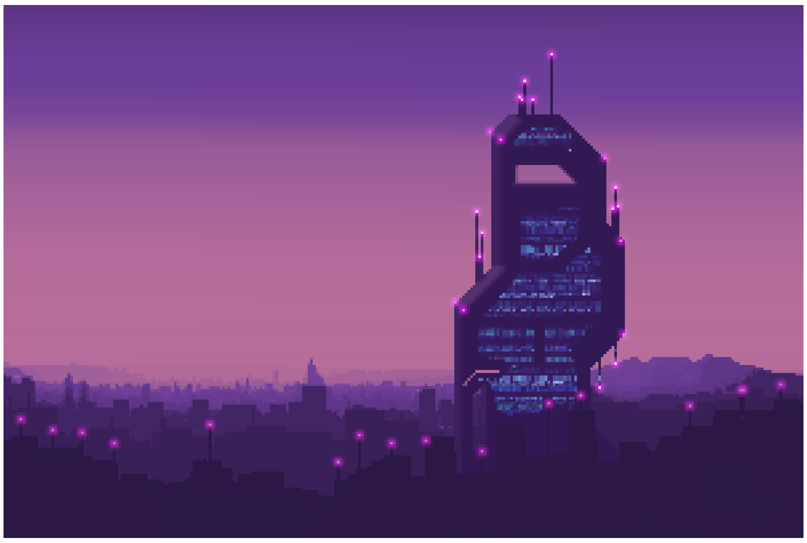
Figure 15: A pixelated font suits the style for my 2D game
This will be the style of text the user sees the score in, when the user interacts with enemies and gains points it will update in this font accordingly

GAME PAUSE

The game pause screen will mainly have one background with 2 options, either resume or end game.
As it is simply a pause screen this background can be much simpler and could even be a gradient/solid. The options will be blitted onto the screen using code.

Aside from the 2 buttons, there will also be two scrollers or settings that will allow the user to alter audio settings. These will also be blitted onto the screen with code.

MAIN SCREEN



I will use the same background as for the game start; however, the building will be alight and there will be effects going on in the screen (flashes and lights etc.), I may also dim the image and zoom in/crop the image so that some parts aren't visible.

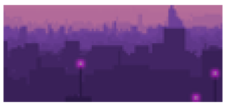


Figure 16: For example, I may use a portion like this as the background.

GAME OVER

This will be the end screen where the user can choose to save their score under a username or discard it. After this they will be sent back to the game start screen.

MAIN SCREEN

The screen for this will most likely be plain black, or it'll have the gradient of the game start background. I will blit a sun going through the screen in the background as well to give the screen a more worldly feeling (as this is what is being maintained in my other screens).

USABILITY FEATURES OF SOLUTION

The solution has been designed so that it is effective and efficient as it doesn't overwhelm the player's pc with graphics that may slow down the game. It is engaging due to the pixelated visuals which my stakeholders have shown an interest in. My solution will also be error tolerant through using me test data which will ensure that everything reacts the way it is intended to. Lastly, my solution should be easy to learn as there are simple mechanics (buttons can also be altered to the user's preferences by changing key binds).

The usability of each feature as described is dependent on user feedback and other black box testing results.

KEY DATA

The key variables/information to be tested will be:

- Key binds for controls

- Functionality for all buttons
- Collisions

TABLE

KEY VARIABLE	DATA TYPE	EXPLANATION	JUSTIFICATION	VALIDATION
CURSOR	Image/. Cur	A cursor allows the user to select options	This is providing a more pleasing aesthetic for the user	Cursor will be validated through user experience, custom cursor image should be visible on the screen.
BUTTONS	N/A	Buttons allow the user to navigate across my game easily	This means that my user can switch between game states or change settings.	Button are validated manually by clicking on them and comparing expected result to real result
SPRITES	Image	Sprites represent the entities visually	This makes helps complete my user experience as they have things they can interact with	Sprites are validated through user experience, should be visible during main phase.
SCORE	Integer	Scores a way of measuring success	This creates a competitive aspect, makes the player feel like they are working towards something.	Score is validated by watching the counter in the main phase, as well as seeing a valid score in the end phase.
BACKGROUND	Image/collection of objects	First layer of the screen	Provides scenery to give the game a theme and make the user feel as if they're in a world	Should be validated through user experience, should be seen when in each phase.

TEST DATA IDENTIFICATION AND JUSTIFICATION

SCORES

Test Data	*Output/Validity*
Non-Player entity is killed	Score increases
Player is killed	Score stays the same
Score in game end phase is higher than high score	Score becomes new high score

Score changes will be validated by monitoring the score in the main playing game phase.

HIGH SCORE NAME

Test Data	Output/Validity
Alphanumerical (letters and numbers)	Valid output, name is added to score list
Non alphanumerical characters (e.g., special characters)	Invalid output, error message shown

High score name can be validated by adding the high score and name, and then going back and checking the score list to see it exists as typed

INPUTS

Test Data	Output/Validity
MOVEMENT keys (WASD by default)	Moves the character
Attack keys (LMB + RMB)	Makes the characters attack

Input will be validated by pressing the keys/buttons and checking more movement/attack

BUTTONS

Test Data	Output/Validity
Buttons pressed	Script related to button is run

Buttons can be pressed and their responses checked to see if they match the script's purpose

SPRITE ANIMATIONS

Test Data	Output/Validity
MOVEMENT keys (WASD by default)	Moving animations are played
Attack keys (LMB + RMB)	Attack animations are played
Death of an entity	Death animations are played

Animations will be tested by playing the game and watching player during the main phase, animations should be playing when character is moving, attacking or has died

SOUND

Test Data	Output/Validity
Sound related event occurs (e.g., movement, attack, or death)	Sound plays
Game is running	Background music plays

Sound is tested via speakers, output should constantly be playing and event related audio cues should also result in audio effects

GAME STATES

Test Data	*Output/Validity*
Start game button pressed	Game moves from start to playing phase.
Game paused during playing phase (using escape)	Game moves from start to pause phase
Player dies	Game moves from playing to over phase
Player ends game	Game moves from paused to over phase
Player selects go back to menu	Game moves from over to start phase

Game states can be validated by starting the game, and then pausing the game, letting the player die, and then going back to menu. This should result in a full cycle validating the game phases.

IDENTIFICATION AND JUSTIFICATION OF FURTHER DATA

Where data isn't included in the above tables, they can be validated/checked via black box testing. User experiences can dictate whether variables are acting the way they are supposed to and success can be measured through surveys.

Data that is not identified in the above (that is visible in my end solution) will most likely be data added as improvisation, and will be identified and justified during my development stage

DEVELOPING A CODED SOLUTION

The development of my solution will follow an agile development style, I will have sessions focussing on various functions and parts of the game based off my systems diagram. These functions will then be tested using my test data as well as extra data that I may not have originally thought of.

The development of my solution will involve my test sprites, then once I am confident with the physics of my game, I will finish creating my proper sprites, then iterating through my code for minor physics changes that may occur due to size/shape changes.

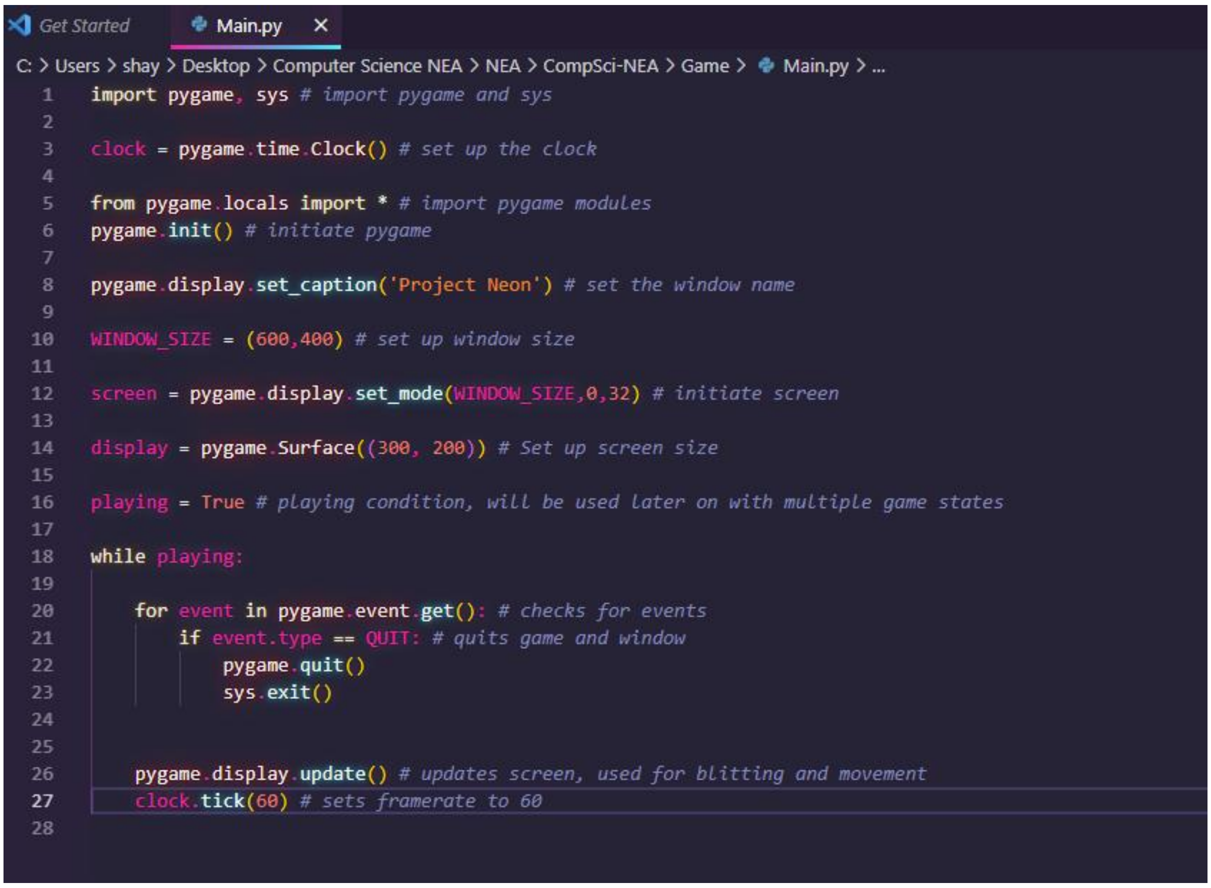
For the most part I will be coding using Visual Studio Code as well as Atom depending on where I'm working from. When I feel I have made significant progress in my game I will also send the game to peers through via Docker's containers, from which I will use feedback to make change to the code, repeating at every considerable progress interval.

I will use various resources such as the Python and PyGame docs to better understand the tools I have available to me. I will also look at tutorials/coding sessions on YouTube and Twitch in order to understand the application of tools and in which contexts they are most efficient in.

ALPHA

MODULE 0 – INITIATING ENVIRONMENT

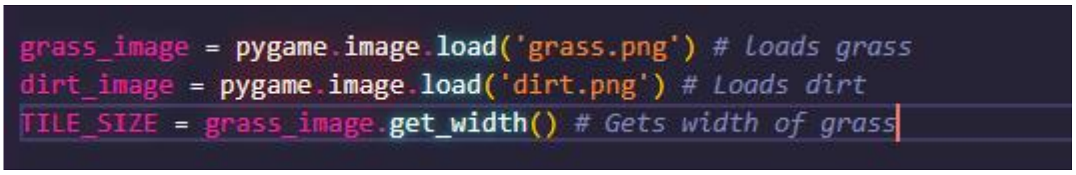
I will begin by adding the essential components into my game and creating the initial window.


Get Started
Main.py
C: > Users > shay > Desktop > Computer Science NEA > NEA > CompSci-NEA > Game > Main.py > ...

```python
import pygame, sys # import pygame and sys

clock = pygame.time.Clock() # set up the clock

from pygame.locals import * # import pygame modules
pygame.init() # initiate pygame

pygame.display.set_caption('Project Neon') # set the window name

WINDOW_SIZE = (600,400) # set up window size

screen = pygame.display.set_mode(WINDOW_SIZE,0,32) # initiate screen

display = pygame.Surface((300, 200)) # Set up screen size

playing = True # playing condition, will be used later on with multiple game states

while playing:

    for event in pygame.event.get(): # checks for events
        if event.type == QUIT: # quits game and window
            pygame.quit()
            sys.exit()


    pygame.display.update() # updates screen, used for blitting and movement
    clock.tick(60) # sets framerate to 60

```

I will then consider basic functionalities for images, input, and collisions.

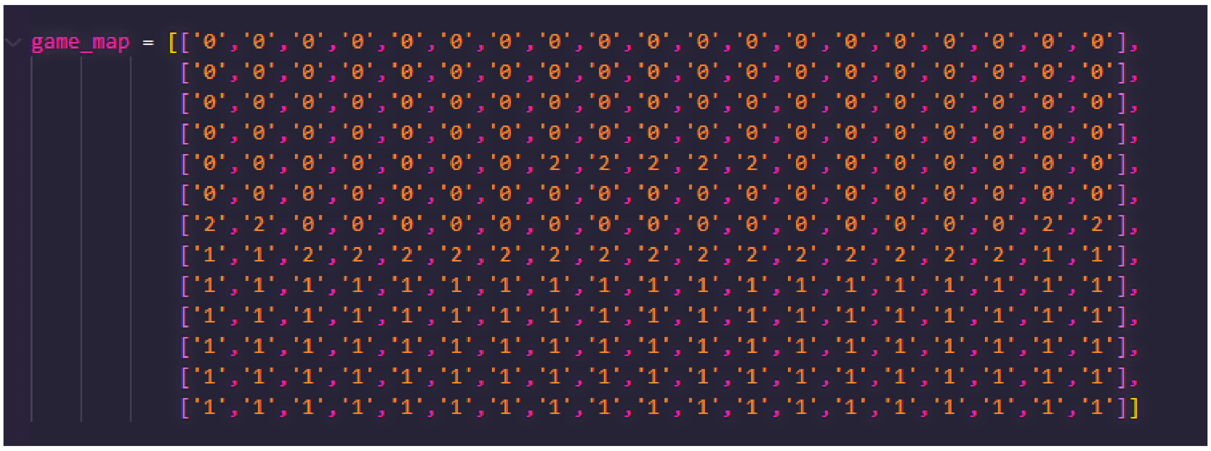
Images can be added simply by calling pygame.image.load() function which just loads the image into the script.



```python
grass_image = pygame.image.load('grass.png') # Loads grass
dirt_image = pygame.image.load('dirt.png') # Loads dirt
TILE_SIZE = grass_image.get_width() # Gets width of grass
```

Figure 17: I have used sampled grass and dirt textures; this is because I am focussing on the functionality of the game at the moment, and I will use proper textures once my basic functionality works.
The reason I have width of the grass is so that I can perfectly blit it onto the screen with no gaps between the image textures.

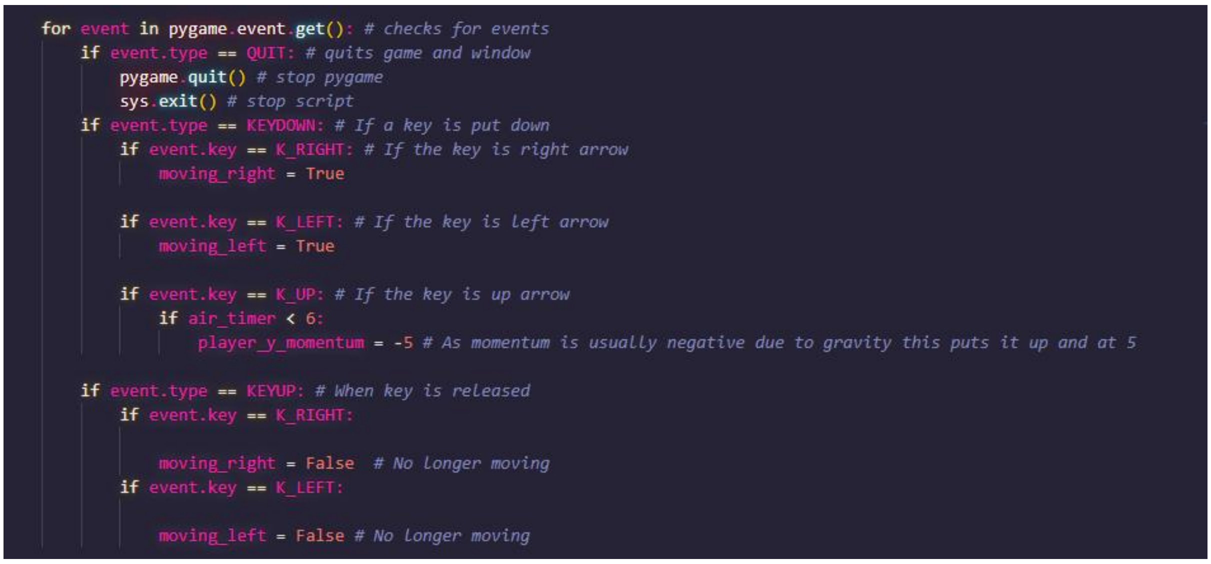
Before I ensure input and movement, I need to create a space for my character to move around on.



```
game_map = [['0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0'],
            ['0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0'],
            ['0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0'],
            ['0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0'],
            ['0','0','0','0','0','0','0','2','2','2','2','2','0','0','0','0','0','0','0'],
            ['0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0'],
            ['2','2','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','2','2'],
            ['1','1','2','2','2','2','2','2','2','2','2','2','2','2','2','2','2','1','1'],
            ['1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1'],
            ['1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1'],
            ['1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1'],
            ['1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1'],
            ['1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1']]
```

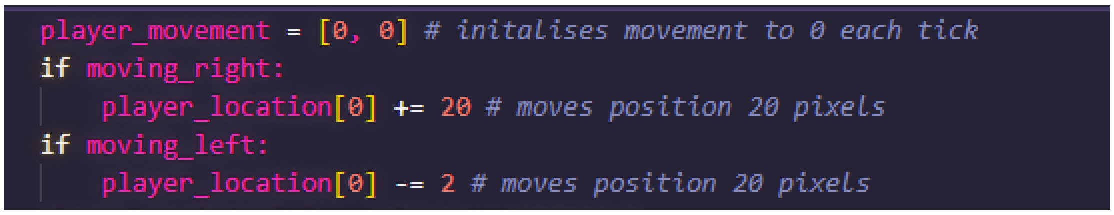
Figure 18: I'm using a 2d game map so I can test the physics and movement of my character, each list inside the main list represents a row of tiles, and each item in each list represents the object to fill that space with, whether its air, grass, dirt.

For movement I am going to be detecting inputs by checking through all the pygame events that are recorded every tick. Then looking for the input I want and making my player react accordingly.

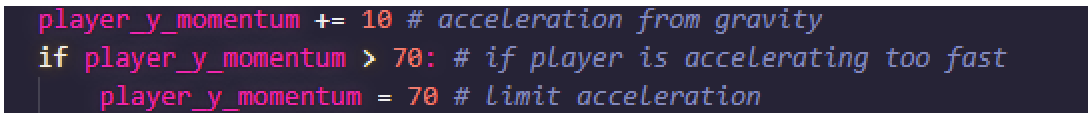


```
for event in pygame.event.get(): # checks for events
    if event.type == QUIT: # quits game and window
        pygame.quit() # stop pygame
        sys.exit() # stop script
    if event.type == KEYDOWN: # If a key is put down
        if event.key == K_RIGHT: # If the key is right arrow
            moving_right = True

        if event.key == K_LEFT: # If the key is left arrow
            moving_left = True

        if event.key == K_UP: # If the key is up arrow
            if air_timer < 6:
                player_y_momentum = -5 # As momentum is usually negative due to gravity this puts it up and at 5

    if event.type == KEYUP: # When key is released
        if event.key == K_RIGHT:

            moving_right = False  # No longer moving
        if event.key == K_LEFT:

            moving_left = False # No longer moving
```

Figure 19: My character will likely move left and right however falls through the screen quickly after being spawned. I added my air timer and momentum earlier than I should've however it's not a big problem as I would've added it shortly after regardless.



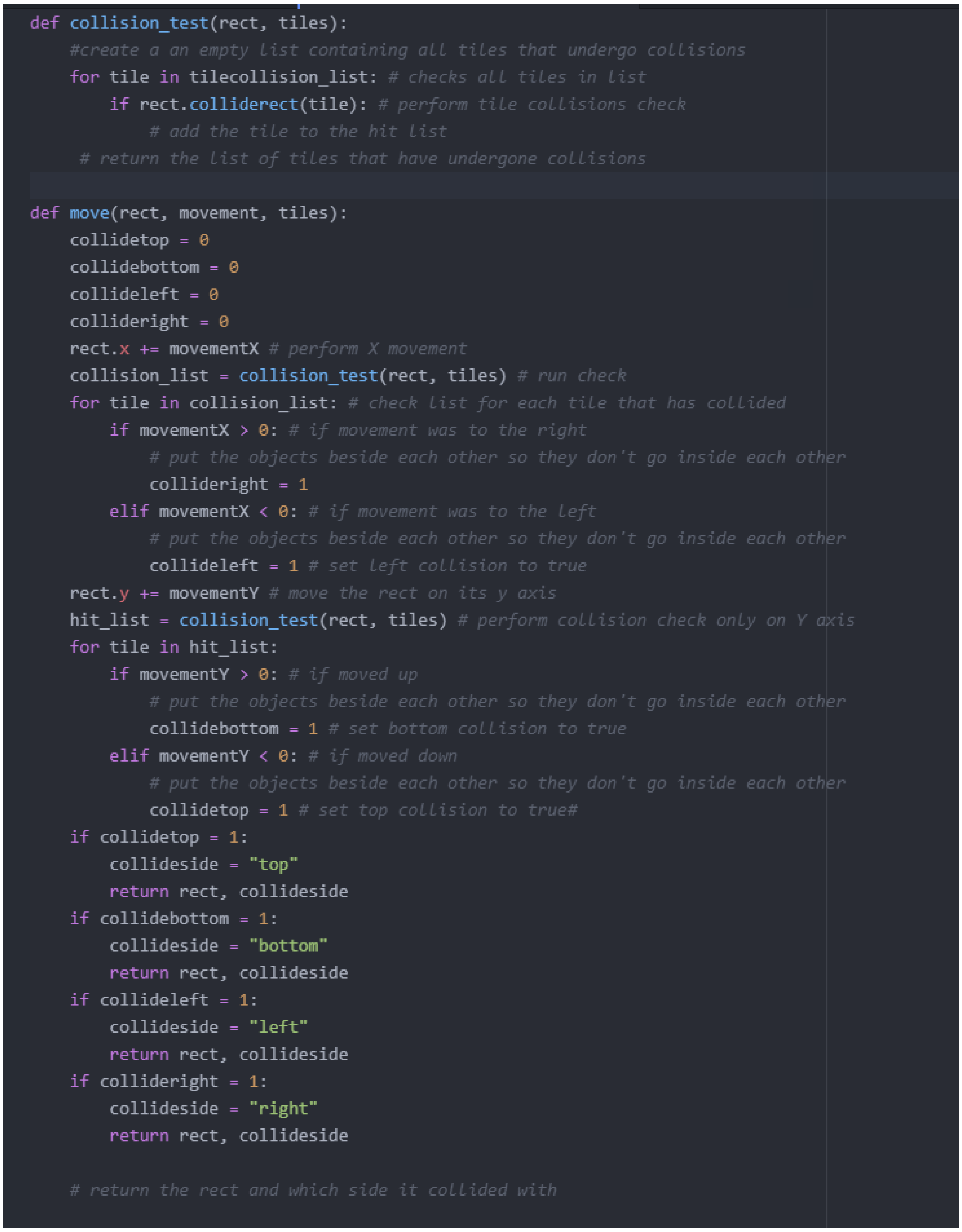
```
player_movement = [0, 0] # initalises movement to 0 each tick
if moving_right:
    player_location[0] += 20 # moves position 20 pixels
if moving_left:
    player_location[0] -= 2 # moves position 20 pixels
```



```
player_y_momentum += 10 # acceleration from gravity
if player_y_momentum > 70: # if player is accelerating too fast
    player_y_momentum = 70 # limit acceleration
```

I am not too sure how to use the pygame library in order to make collisions, however I understand that the basic idea is to check whether the co-ordinates of the surfaces of two objects are inside the height/width of another. For now, I will look for how people have approached collisions online and try and understand how they have implemented functions that revitalise their concepts

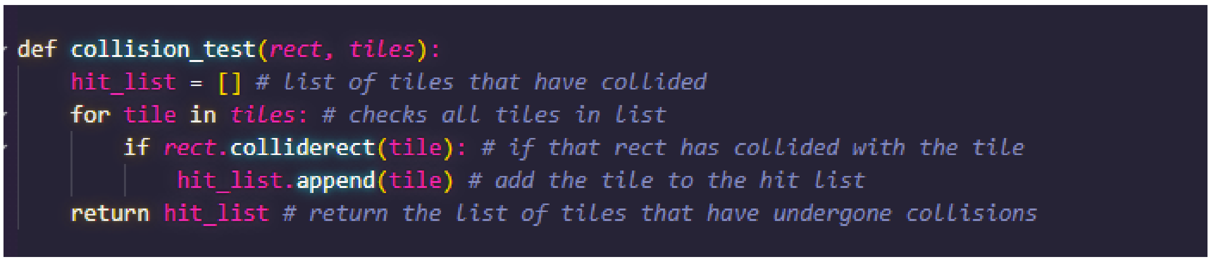
Following how people have handled collisions online, I will begin by creating a collision test function that checks whether the rect hitbox of my player collides with any tiles.



```
def collision_test(rect, tiles):
    #create a an empty list containing all tiles that undergo collisions
    for tile in tilecollision_list: # checks all tiles in list
        if rect.colliderect(tile): # perform tile collisions check
            # add the tile to the hit list
     # return the list of tiles that have undergone collisions

def move(rect, movement, tiles):
    collidetop = 0
    collidebottom = 0
    collideleft = 0
    collideright = 0
    rect.x += movementX # perform X movement
    collision_list = collision_test(rect, tiles) # run check
    for tile in collision_list: # check list for each tile that has collided
        if movementX > 0: # if movement was to the right
            # put the objects beside each other so they don't go inside each other
            collideright = 1
        elif movementX < 0: # if movement was to the left
            # put the objects beside each other so they don't go inside each other
            collideleft = 1 # set left collision to true
    rect.y += movementY # move the rect on its y axis
    hit_list = collision_test(rect, tiles) # perform collision check only on Y axis
    for tile in hit_list:
        if movementY > 0: # if moved up
            # put the objects beside each other so they don't go inside each other
            collidebottom = 1 # set bottom collision to true
        elif movementY < 0: # if moved down
            # put the objects beside each other so they don't go inside each other
            collidetop = 1 # set top collision to true#
    if collidetop = 1:
        collideside = "top"
        return rect, collideside
    if collidebottom = 1:
        collideside = "bottom"
        return rect, collideside
    if collideleft = 1:
        collideside = "left"
        return rect, collideside
    if collideright = 1:
        collideside = "right"
        return rect, collideside

    # return the rect and which side it collided with
```

Figure 20: My basic idea based on my theory and how people have approached it using 2 functions

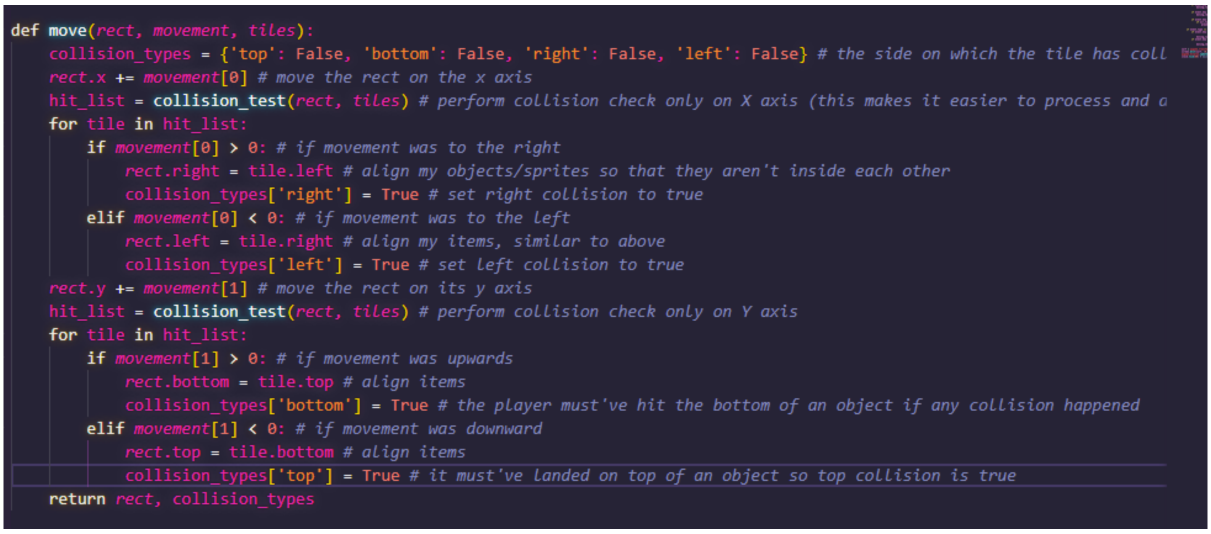
I could make this a lot cleaner by using dictionaries and lists to avoid multiple variables, however I have also found some better approaches on the internet that I would prefer to understand and take inspiration from so that I can better handle and understand collisions with enemies in the future. As proof of understanding I will comment each line of code.



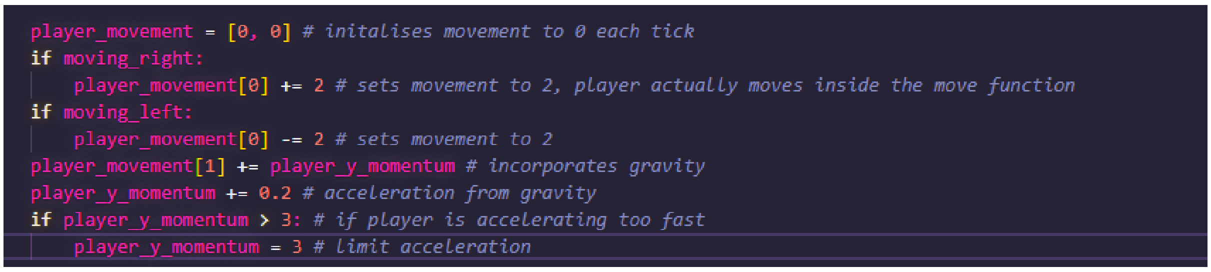
```python
def collision_test(rect, tiles):
    hit_list = [] # list of tiles that have collided
    for tile in tiles: # checks all tiles in list
        if rect.colliderect(tile): # if that rect has collided with the tile
            hit_list.append(tile) # add the tile to the hit list
    return hit_list # return the list of tiles that have undergone collisions
```

Figure 21: I prefer this method as it uses built in pygame functions, which deepens my understanding of the library therefore enabling me to tackle future problems (e.g. regarding collisions between two enemies) more easily.

Then using my collision test a move function is created which runs every time a player moves



```python
def move(rect, movement, tiles):
    collision_types = {'top': False, 'bottom': False, 'right': False, 'left': False} # the side on which the tile has coll
    rect.x += movement[0] # move the rect on the x axis
    hit_list = collision_test(rect, tiles) # perform collision check only on X axis (this makes it easier to process and a
    for tile in hit_list:
        if movement[0] > 0: # if movement was to the right
            rect.right = tile.left # align my objects/sprites so that they aren't inside each other
            collision_types['right'] = True # set right collision to true
        elif movement[0] < 0: # if movement was to the left
            rect.left = tile.right # align my items, similar to above
            collision_types['left'] = True # set left collision to true
    rect.y += movement[1] # move the rect on its y axis
    hit_list = collision_test(rect, tiles) # perform collision check only on Y axis
    for tile in hit_list:
        if movement[1] > 0: # if movement was upwards
            rect.bottom = tile.top # align items
            collision_types['bottom'] = True # the player must've hit the bottom of an object if any collision happened
        elif movement[1] < 0: # if movement was downward
            rect.top = tile.bottom # align items
            collision_types['top'] = True # it must've landed on top of an object so top collision is true
    return rect, collision_types
```



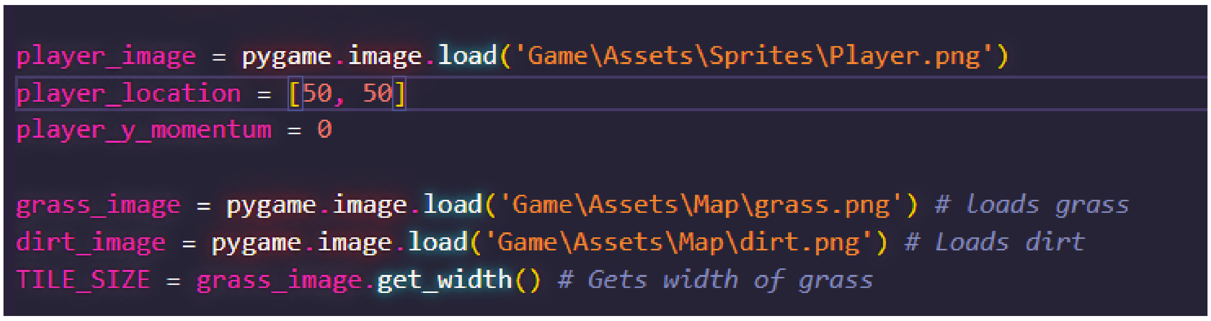
```python
player_movement = [0, 0] # initalises movement to 0 each tick
if moving_right:
    player_movement[0] += 2 # sets movement to 2, player actually moves inside the move function
if moving_left:
    player_movement[0] -= 2 # sets movement to 2
player_movement[1] += player_y_momentum # incorporates gravity
player_y_momentum += 0.2 # acceleration from gravity
if player_y_momentum > 3: # if player is accelerating too fast
    player_y_momentum = 3 # limit acceleration
```

Figure 22: My movement has changed to more suitable numbers, and it uses momentum to help display the effects of collision

MODULE 1 - CAMERA MOVEMENT

This will be my first branch and attempt at slowly adapting my current basic script into my game.

I have performed small directory changes to keep my work organised and ensure the game still works

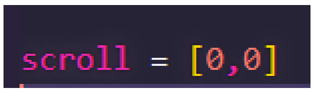


```
player_image = pygame.image.load('Game\Assets\Sprites\Player.png')
player_location = [50, 50]
player_y_momentum = 0

grass_image = pygame.image.load('Game\Assets\Map\grass.png') # loads grass
dirt_image = pygame.image.load('Game\Assets\Map\dirt.png') # Loads dirt
TILE_SIZE = grass_image.get_width() # Gets width of grass
```

Figure 23: More specifically, the paths to each item in this code block has changed.

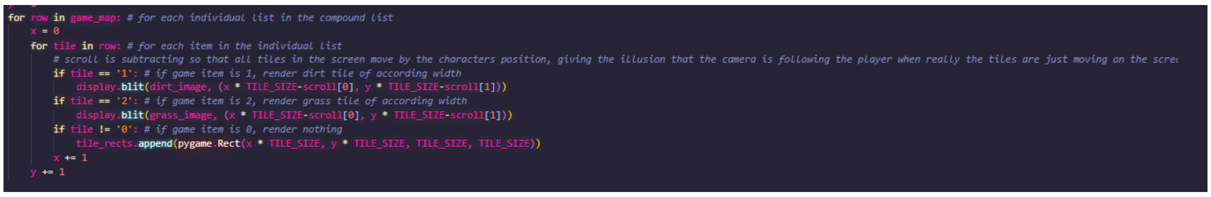
Firstly, I will add a camera to my game for extended movement in the screen. As I'm not working with a game engine, I cannot literally implement a camera that follows the character; looking at similar projects online I can tell that the camera movement is simulated by moving the whole map left or right across the screen.

In order to keep track of how much my map has moved/scrolled by, I will introduce 2 new variables, ScrollX and ScrollY which will both be contained in a list scroll.

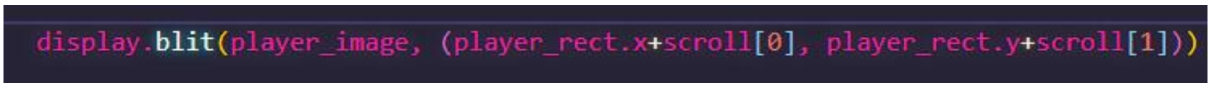


```
scroll = [0,0]
```

And then I can add on these scroll variables to my map loading function in my main loop.

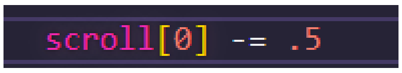


```
for row in game_map: # for each individual list in the compound list
    x = 0
    for tile in row: # for each item in the individual list
        # scroll is subtracting so that all tiles in the screen move by the characters position, giving the illusion that the camera is following the player when really the tiles are just moving on the scre
        if tile == '1': # if game item is 1, render dirt tile of according width
            display.blit(dirt_image, (x * TILE_SIZE-scroll[0], y * TILE_SIZE-scroll[1]))
        if tile == '2': # if game item is 2, render grass tile of according width
            display.blit(grass_image, (x * TILE_SIZE-scroll[0], y * TILE_SIZE-scroll[1]))
        if tile != '0': # if game item is 0, render nothing
            tile_rects.append(pygame.Rect(x * TILE_SIZE, y * TILE_SIZE, TILE_SIZE, TILE_SIZE))
        x += 1
    y += 1
```



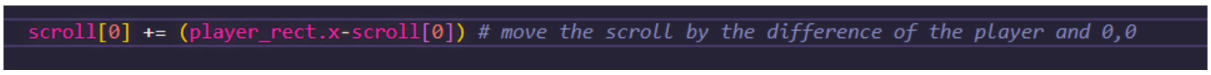
```
display.blit(player_image, (player_rect.x+scroll[0], player_rect.y+scroll[1]))
```

To test this, I will set my scroll value to decrement by 0.5 and the map should move/the camera should pan.



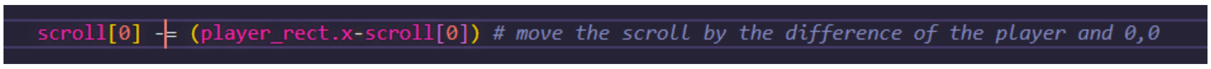
```
scroll[0] -= .5
```

As this has worked successfully, I will now match the scroll to the players position and centre it to the middle of the screen.



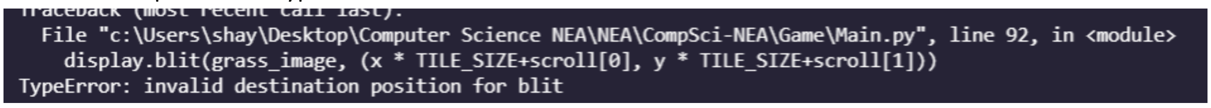
```
scroll[0] += (player_rect.x-scroll[0]) # move the scroll by the difference of the player and 0,0
```

Doing this I notice that my camera actually moves in the opposite direction than I want it to, so I will change the scroll to decrement instead of increment.



```
scroll[0] -= (player_rect.x-scroll[0]) # move the scroll by the difference of the player and 0,0
```

This however provides a TypeError:



```
Traceback (most recent call last):
  File "c:\Users\shay\Desktop\Computer Science NEA\NEA\CompSci-NEA\Game\Main.py", line 92, in <module>
    display.blit(grass_image, (x * TILE_SIZE+scroll[0], y * TILE_SIZE+scroll[1]))
TypeError: invalid destination position for blit
```

As my error describes, the destination position for the blit is invalid. I assume that this is because the original x position for the scroll and leftmost tiles is 0, and therefore when I move, the tiles would be in a negative x position. This error impacts my gameplay heavily as the camera function won't work at all and so my player won't see anything. One possible solution to this is to invert all the addition/subtraction signs, this means that my scroll moves the tile to a valid place on the screen and therefore doesn't cause an error.

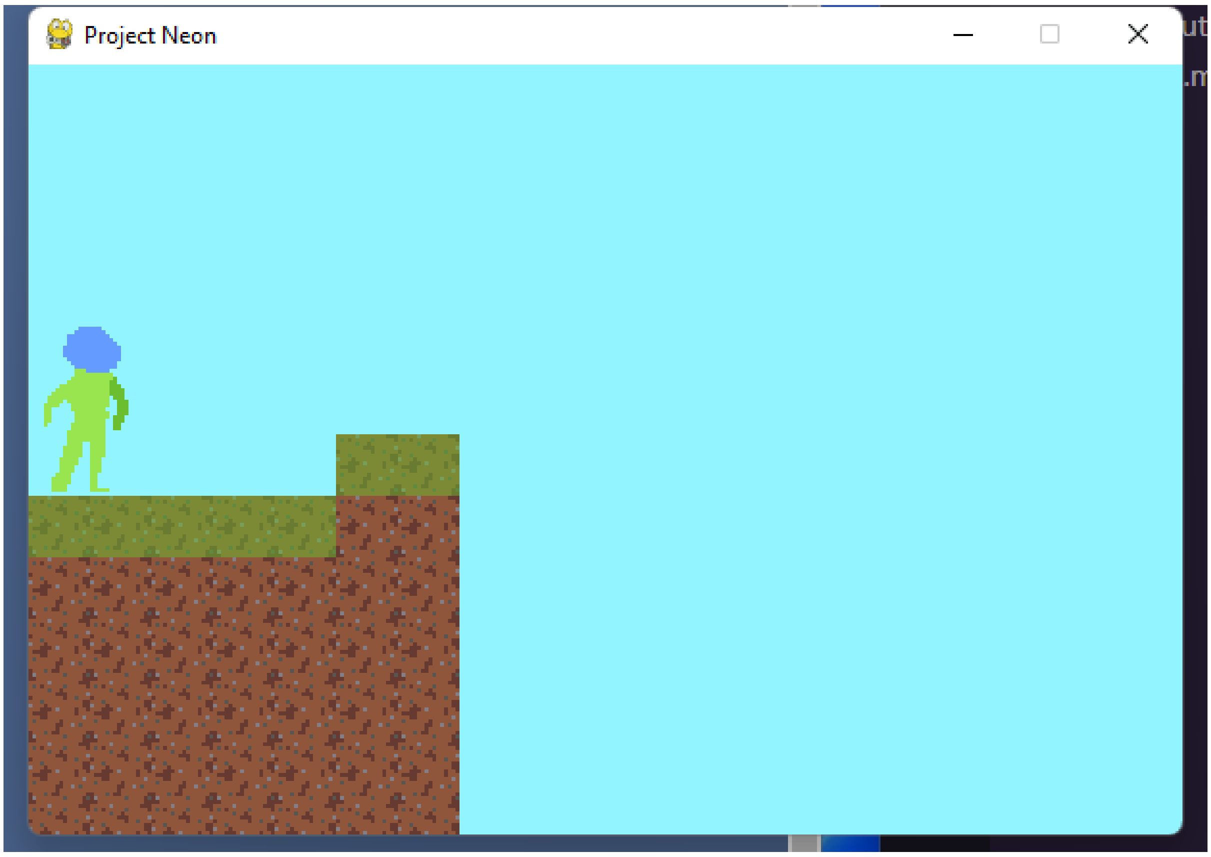
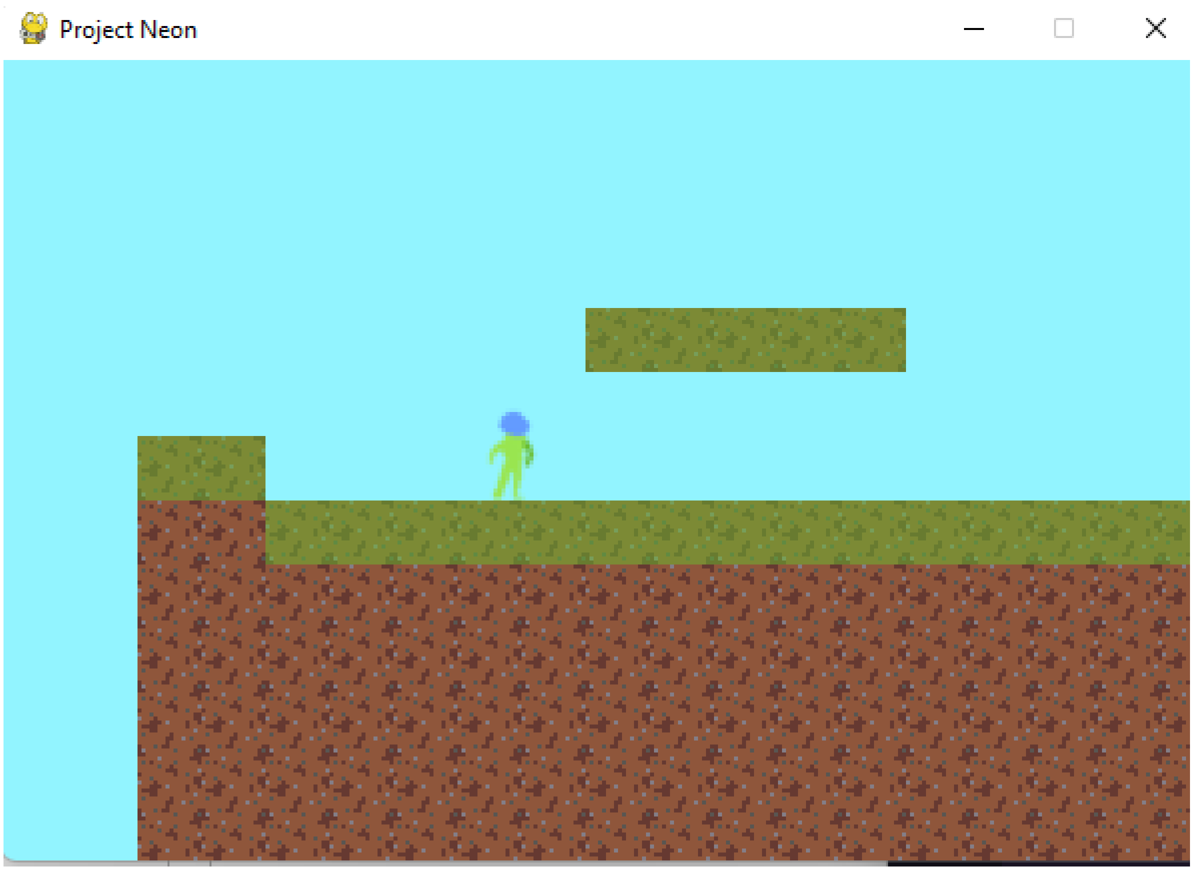


Figure 24: Character moving around a small land face

I can simply move my player to the middle by subtracting half of my screen width as well as half of my character width, however as I want my player to be more on the left-middle of the screen I will use a value of 120 which is slightly less than my 166. I will also take a similar approach for vertical scroll.

```
scroll[0] += (player_rect.x-scroll[0]-120) # move the scroll by the difference of the player and 0,0 on the X
scroll[1] += (player_rect.y-scroll[1]-116) # move the scroll by the difference of the player and 0,0 on the Y
```

I also feel like my character is a little too big so I will scale him down by 2x and see if it suits the screen better



I prefer this size more so I will keep using this until I do proper sprite creation.

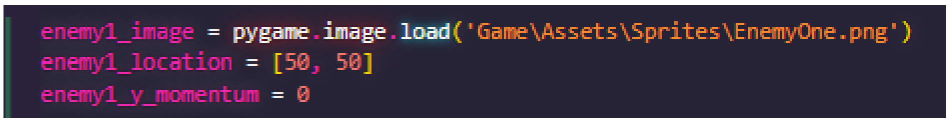
My camera movement feels a little stiff and sudden at the moment however I don't really know how to fix this so I will come back to it later.

MODULE 2 – ENEMY GENERATION AND INTERACTION

MODULE 2.1 ENEMY GENERATION

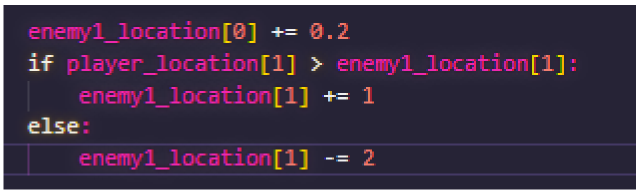
This module will be around creating enemies as well as being able to interact (damage or take damage) from them. When creating enemies, I should refer to my enemy design in the analysis and design stages. The enemy must be able to attack and deal damage to the player, and ideally it should take damage and die after it loses enough health.

The concept of creating an enemy at first strongly resembles how I would create my first player, using a sprite and a hitbox.



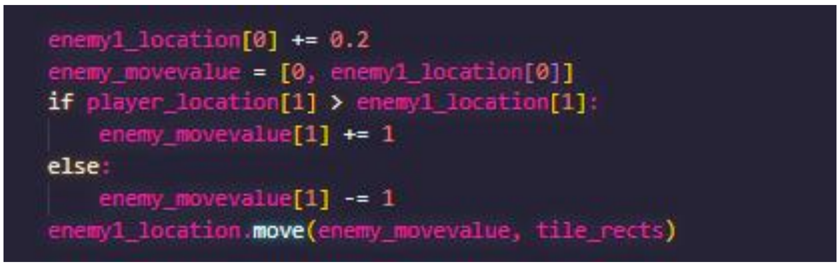
```
enemy1_image = pygame.image.load('Game\Assets\Sprites\EnemyOne.png')
enemy1_location = [50, 50]
enemy1_y_momentum = 0
```

Initially I had begun coding the enemy to follow the movement through something like this:



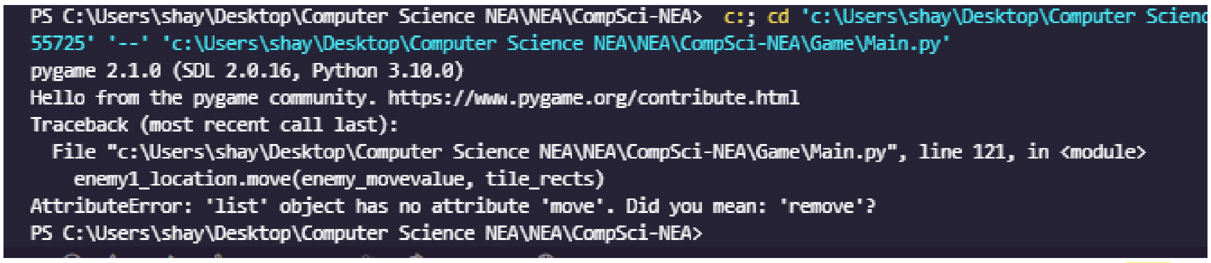
```
enemy1_location[0] += 0.2
if player_location[1] > enemy1_location[1]:
    enemy1_location[1] += 1
else:
    enemy1_location[1] -= 2
```

However, I realised I have the pygame move function for the player which I can simply apply to the enemy as well, that way I don't need to consider collisions in a more complex way for the enemy later on.



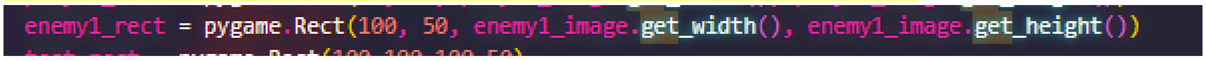
```
enemy1_location[0] += 0.2
enemy_movevalue = [0, enemy1_location[0]]
if player_location[1] > enemy1_location[1]:
    enemy_movevalue[1] += 1
else:
    enemy_movevalue[1] -= 1
enemy1_location.move(enemy_movevalue, tile_rects)
```

This however provides me an AttributeError:

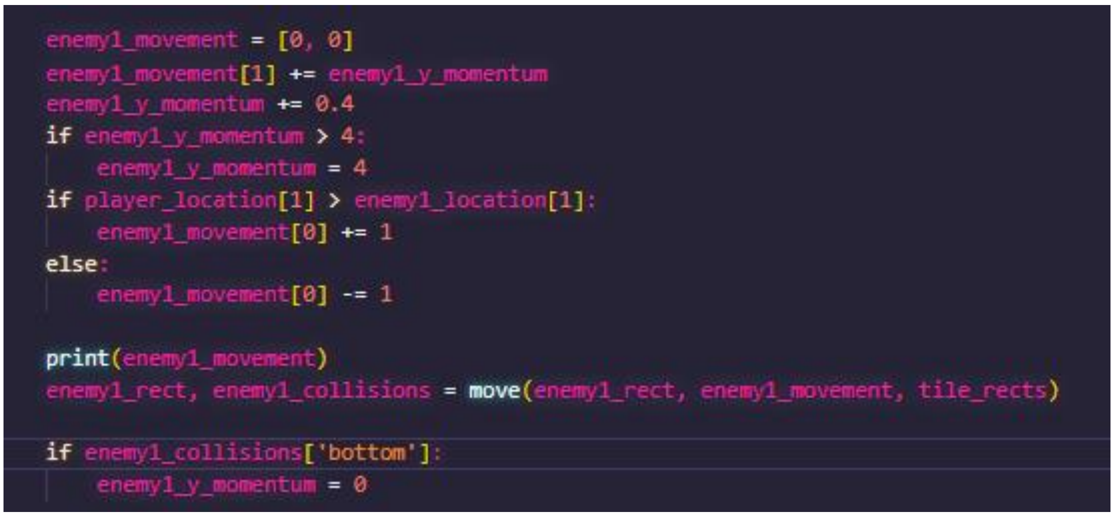


```
PS C:\Users\shay\Desktop\Computer Science NEA\NEA\CompSci-NEA>  c:; cd 'c:\Users\shay\Desktop\Computer Scienc
55725' '--' 'c:\Users\shay\Desktop\Computer Science NEA\NEA\CompSci-NEA\Game\Main.py'
pygame 2.1.0 (SDL 2.0.16, Python 3.10.0)
Hello from the pygame community. https://www.pygame.org/contribute.html
Traceback (most recent call last):
  File "c:\Users\shay\Desktop\Computer Science NEA\NEA\CompSci-NEA\Game\Main.py", line 121, in <module>
    enemy1_location.move(enemy_movevalue, tile_rects)
AttributeError: 'list' object has no attribute 'move'. Did you mean: 'remove'?
PS C:\Users\shay\Desktop\Computer Science NEA\NEA\CompSci-NEA>
```

This is because the move method applies to rects, and I had forgotten to create the rect for the enemy. This error effects my game as a functioning, moving enemy is critical in a combat system.



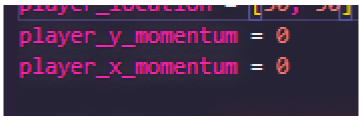
```
enemy1_rect = pygame.Rect(100, 50, enemy1_image.get_width(), enemy1_image.get_height())
```



```
enemy1_movement = [0, 0]
enemy1_movement[1] += enemy1_y_momentum
enemy1_y_momentum += 0.4
if enemy1_y_momentum > 4:
    enemy1_y_momentum = 4
if player_location[1] > enemy1_location[1]:
    enemy1_movement[0] += 1
else:
    enemy1_movement[0] -= 1

print(enemy1_movement)
enemy1_rect, enemy1_collisions = move(enemy1_rect, enemy1_movement, tile_rects)

if enemy1_collisions['bottom']:
    enemy1_y_momentum = 0
```

Figure 25: After changing the code to work based off of the move method, as well as taking into account things like momentum.

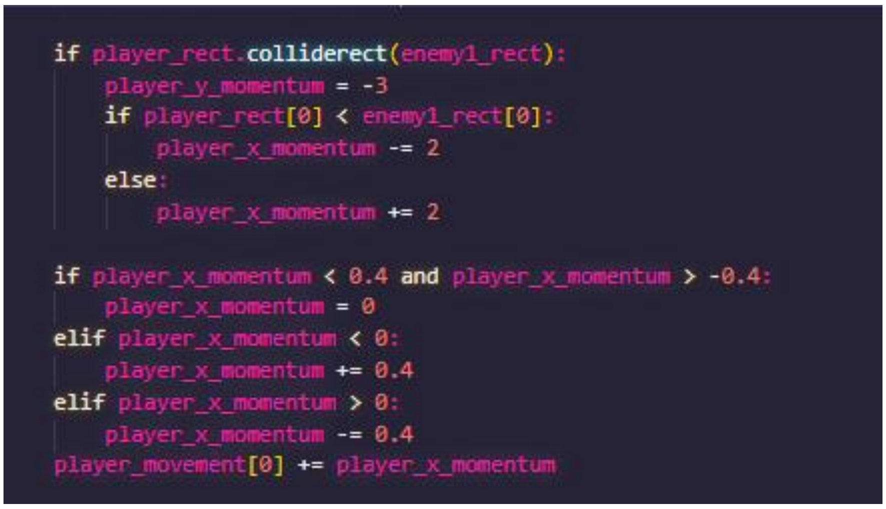
This lets my enemy successfully follow the player. Next, I will add in collision detection which will make the player bump if her rect collides with that of the enemy.



```
player_y_momentum = 0
player_x_momentum = 0
```

Figure 26: Beginning by adding an x momentum variable. I could optimise this by turning momentum into a list with x and y variables however I will do that once I know my logic in handling knockback is correct.

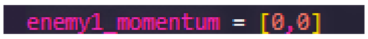
I will also make my map a bit longer by making the list longer and adding more “1” s for the sake of better understanding knockback effects and intensity without falling off the map.



```
if player_rect.colliderect(enemy1_rect):
    player_y_momentum = -3
    if player_rect[0] < enemy1_rect[0]:
        player_x_momentum -= 2
    else:
        player_x_momentum += 2

if player_x_momentum < 0.4 and player_x_momentum > -0.4:
    player_x_momentum = 0
elif player_x_momentum < 0:
    player_x_momentum += 0.4
elif player_x_momentum > 0:
    player_x_momentum -= 0.4
player_movement[0] += player_x_momentum
```

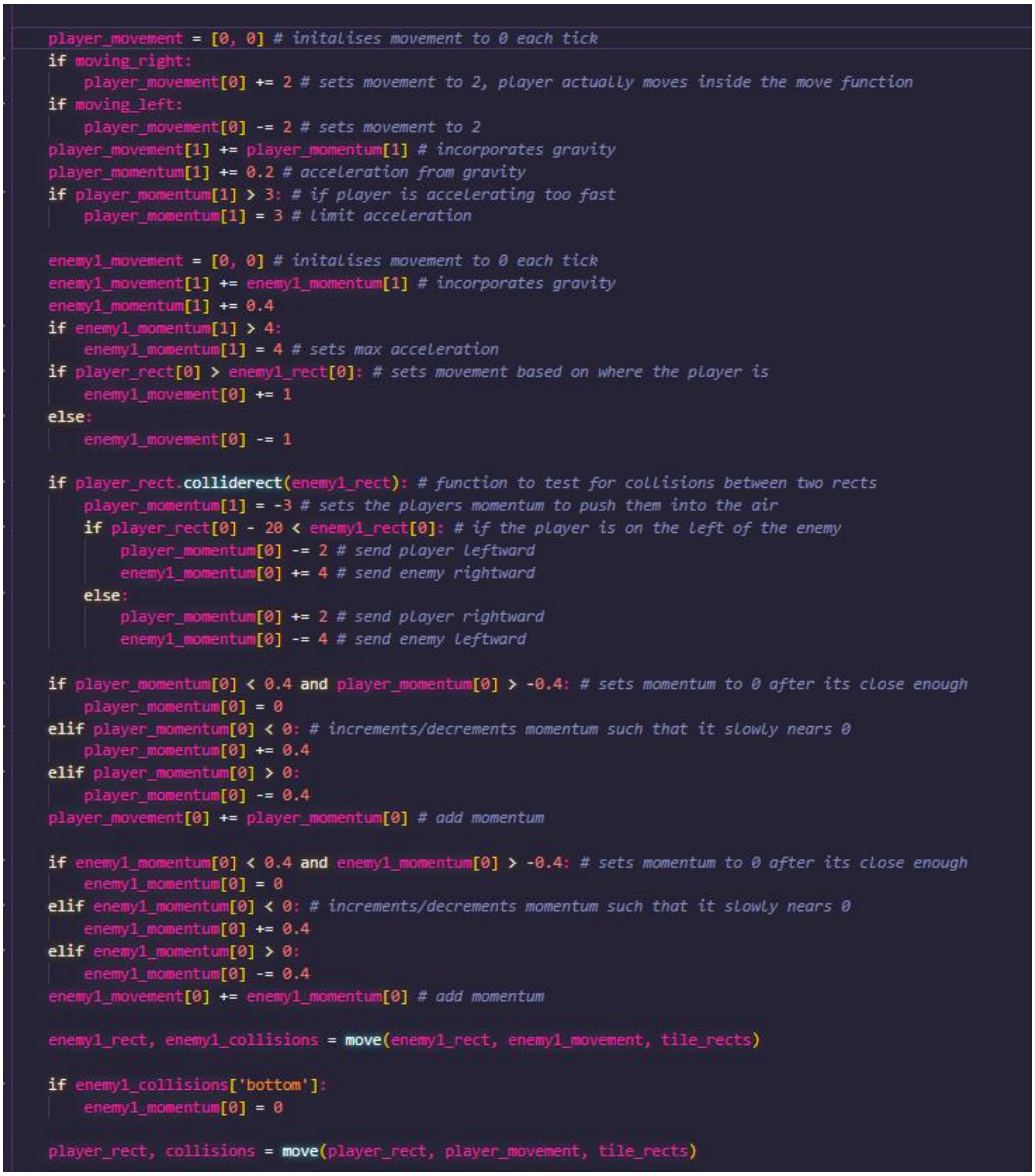
Figure 27: This successfully shows player knockback.

I would also like to add a little knockback on the enemy’s side to avoid multiple collisions at once as well as turn the momentum on the players side into a list.



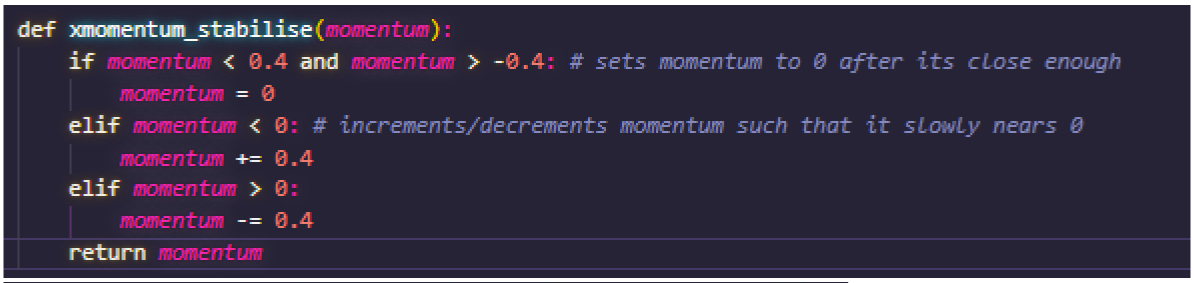
```
enemy1_momentum = [0,0]
```

Adding in the enemies' components for momentum as well as a limit to 0 and fixing small issues with hitboxes:

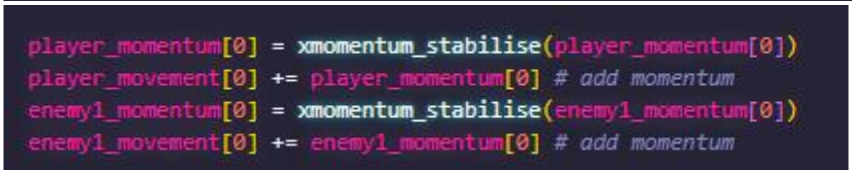


```
player_movement = [0, 0] # initalises movement to 0 each tick
if moving_right:
    player_movement[0] += 2 # sets movement to 2, player actually moves inside the move function
if moving_left:
    player_movement[0] -= 2 # sets movement to 2
player_movement[1] += player_momentum[1] # incorporates gravity
player_momentum[1] += 0.2 # acceleration from gravity
if player_momentum[1] > 3: # if player is accelerating too fast
    player_momentum[1] = 3 # limit acceleration

enemy1_movement = [0, 0] # initalises movement to 0 each tick
enemy1_movement[1] += enemy1_momentum[1] # incorporates gravity
enemy1_momentum[1] += 0.4
if enemy1_momentum[1] > 4:
    enemy1_momentum[1] = 4 # sets max acceleration
if player_rect[0] > enemy1_rect[0]: # sets movement based on where the player is
    enemy1_movement[0] += 1
else:
    enemy1_movement[0] -= 1

if player_rect.colliderect(enemy1_rect): # function to test for collisions between two rects
    player_momentum[1] = -3 # sets the players momentum to push them into the air
    if player_rect[0] - 20 < enemy1_rect[0]: # if the player is on the left of the enemy
        player_momentum[0] -= 2 # send player leftward
        enemy1_momentum[0] += 4 # send enemy rightward
    else:
        player_momentum[0] += 2 # send player rightward
        enemy1_momentum[0] -= 4 # send enemy leftward

if player_momentum[0] < 0.4 and player_momentum[0] > -0.4: # sets momentum to 0 after its close enough
    player_momentum[0] = 0
elif player_momentum[0] < 0: # increments/decrements momentum such that it slowly nears 0
    player_momentum[0] += 0.4
elif player_momentum[0] > 0:
    player_momentum[0] -= 0.4
player_movement[0] += player_momentum[0] # add momentum

if enemy1_momentum[0] < 0.4 and enemy1_momentum[0] > -0.4: # sets momentum to 0 after its close enough
    enemy1_momentum[0] = 0
elif enemy1_momentum[0] < 0: # increments/decrements momentum such that it slowly nears 0
    enemy1_momentum[0] += 0.4
elif enemy1_momentum[0] > 0:
    enemy1_momentum[0] -= 0.4
enemy1_movement[0] += enemy1_momentum[0] # add momentum

enemy1_rect, enemy1_collisions = move(enemy1_rect, enemy1_movement, tile_rects)

if enemy1_collisions['bottom']:
    enemy1_momentum[0] = 0

player_rect, collisions = move(player_rect, player_movement, tile_rects)
```

I will turn the sequences I created today into functions for selection as I plan on having more enemies and it would be easier for me to implement their physics by just calling the same function

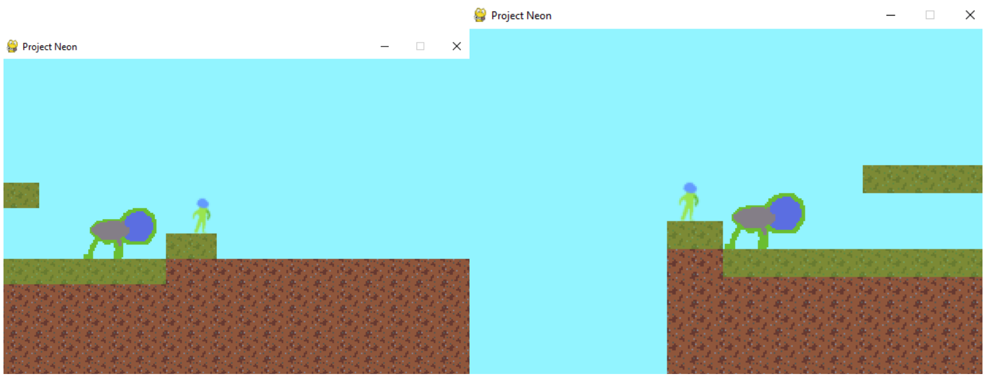


```
def xmomentum_stabilise(momentum):
    if momentum < 0.4 and momentum > -0.4: # sets momentum to 0 after its close enough
        momentum = 0
    elif momentum < 0: # increments/decrements momentum such that it slowly nears 0
        momentum += 0.4
    elif momentum > 0:
        momentum -= 0.4
    return momentum
```



```
player_momentum[0] = xmomentum_stabilise(player_momentum[0])
player_movement[0] += player_momentum[0] # add momentum
enemy1_momentum[0] = xmomentum_stabilise(enemy1_momentum[0])
enemy1_movement[0] += enemy1_momentum[0] # add momentum
```

ENEMY BEHAVIOUR



The enemy properly always runs towards the player, and attacks the player when in contact, applying knockback. One thing I am unhappy with is the behaviour of the sprite, which I would like to have flip horizontally so that its always facing the direction its going.

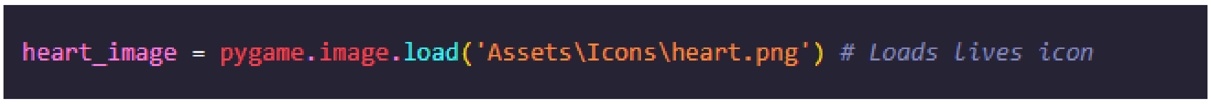
Unfortunately, due to complexity & time constraints, I will not be randomising enemy behaviour. This would be something for me to focus on during future development.

MODULE 2.2 ENEMY INTERACTION

In order for me to perform enemy interaction, I will need to add attacking states and a visual representation of the player and enemy taking damage as well as the player swinging.

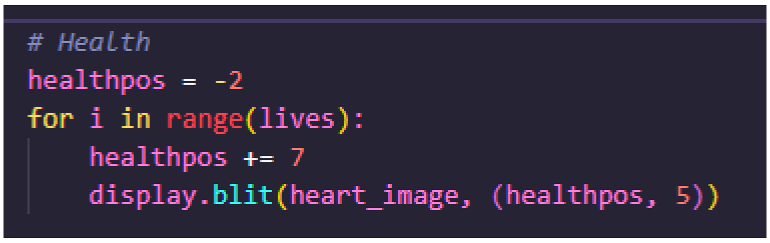
I will begin by creating some sort of counter on the screen and then having it decrement per collision.

LIVES



```
heart_image = pygame.image.load('Assets\Icons\heart.png') # Loads lives icon
```

Figure 28: Using an icon I designed using pixel design online



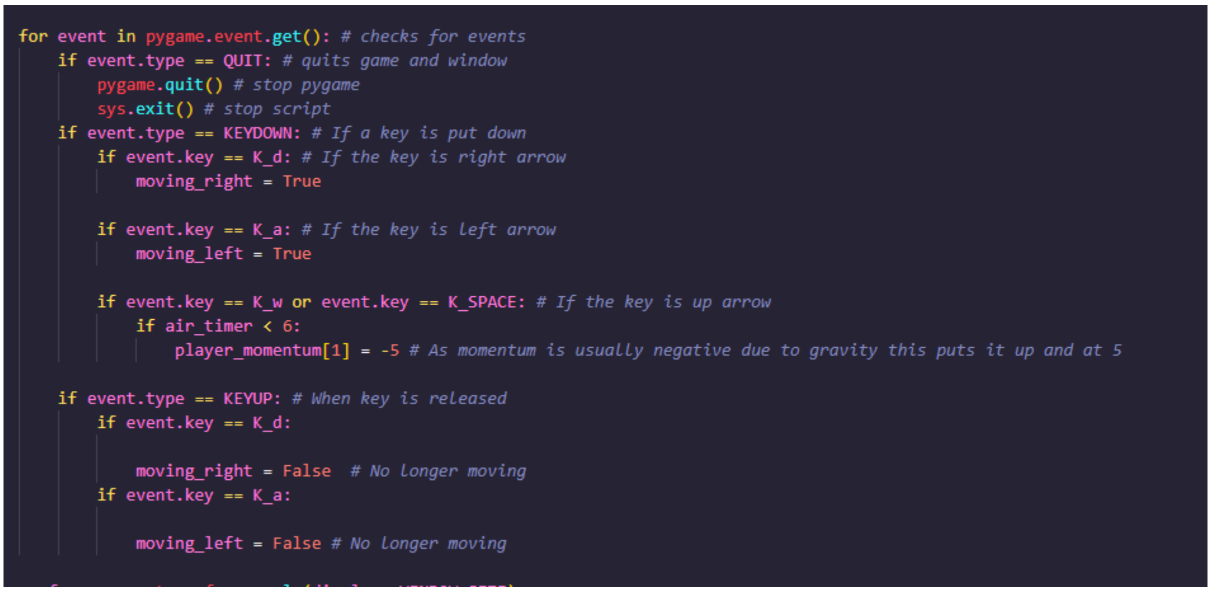
```
# Health
healthpos = -2
for i in range(lives):
    healthpos += 7
    display.blit(heart_image, (healthpos, 5))
```

Figure 29: This defines the heart surface, and then places several of them on the top right of the screen.

ATTACKING STATE

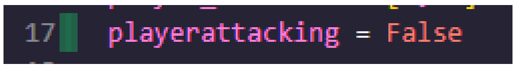
The enemy should always be in an attacking state, the player however will only enter an attacking state once the attack key bind is pressed, after which for a short duration it will deal damage instead of taking some (given she is facing the correct direction).

At this moment in time, I will also move the directional keys to WASD instead of arrow keys as they are more appropriate when keys are used for more than just movement.



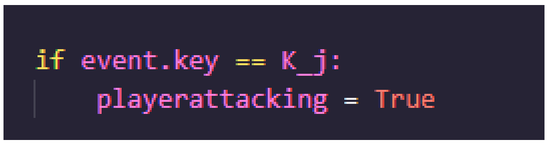
```
for event in pygame.event.get(): # checks for events
    if event.type == QUIT: # quits game and window
        pygame.quit() # stop pygame
        sys.exit() # stop script
    if event.type == KEYDOWN: # If a key is put down
        if event.key == K_d: # If the key is right arrow
            moving_right = True

        if event.key == K_a: # If the key is left arrow
            moving_left = True

        if event.key == K_w or event.key == K_SPACE: # If the key is up arrow
            if air_timer < 6:
                player_momentum[1] = -5 # As momentum is usually negative due to gravity this puts it up and at 5

    if event.type == KEYUP: # When key is released
        if event.key == K_d:

            moving_right = False  # No longer moving
        if event.key == K_a:

            moving_left = False # No longer moving
```

In order to create an attacking state, I'll have to define a variable that shows if the player is attacking. This would be a Boolean as its either attacking or isn't.



```
playerattacking = False
```

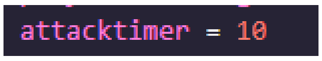
Now I'll add a keydown event checker that, when triggered, changes the players state to attack.



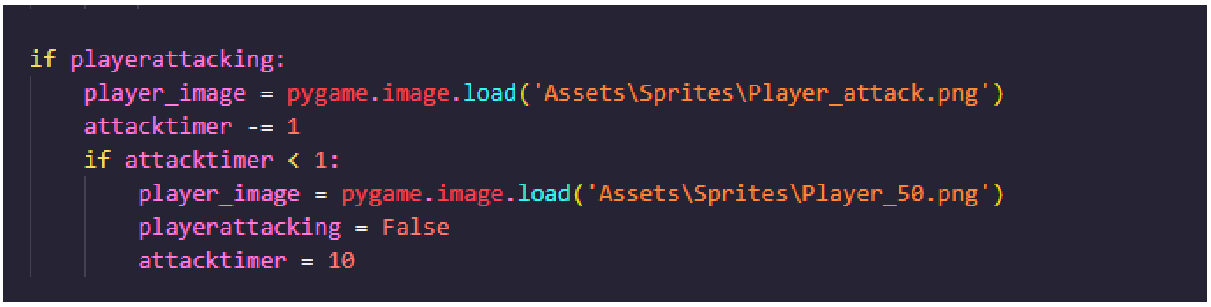
```
if event.key == K_j:
    playerattacking = True
```

I have shaded the existing player sprite into red for reference for when it is attacking.

The player will adopt this colour in his attacking state.

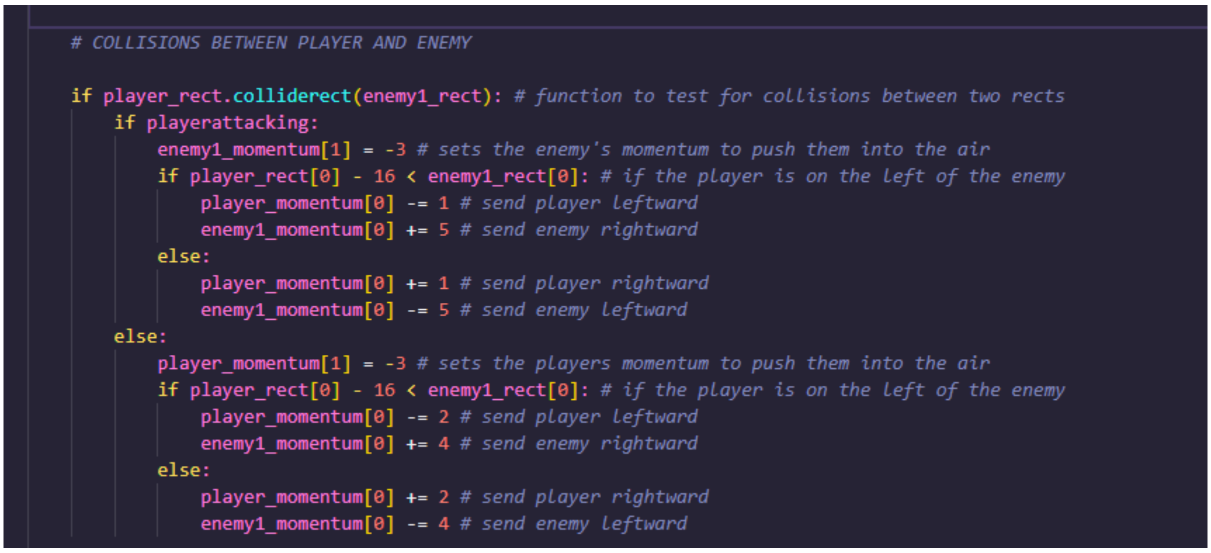


```
attacktimer = 10
```



```
if playerattacking:
    player_image = pygame.image.load('Assets\Sprites\Player_attack.png')
    attacktimer -= 1
    if attacktimer < 1:
        player_image = pygame.image.load('Assets\Sprites\Player_50.png')
        playerattacking = False
        attacktimer = 10
```

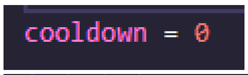
Figure 30: This code initiates a timer of 10 frames, and whenever the player enters the attacking state it slowly counts down to 0 from which it returns to the normal state



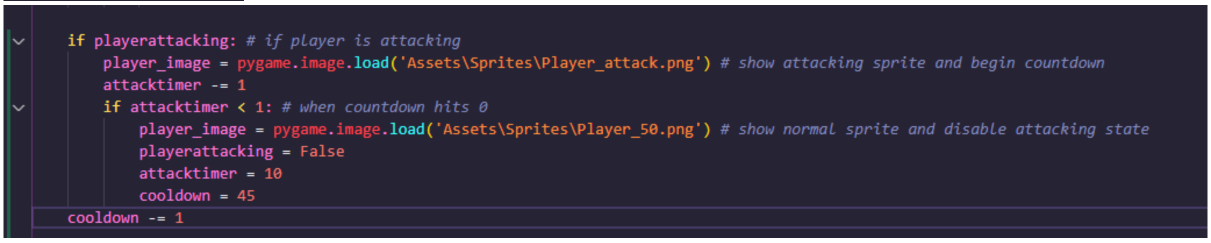
```
# COLLISIONS BETWEEN PLAYER AND ENEMY

if player_rect.colliderect(enemy1_rect): # function to test for collisions between two rects
    if playerattacking:
        enemy1_momentum[1] = -3 # sets the enemy's momentum to push them into the air
        if player_rect[0] - 16 < enemy1_rect[0]: # if the player is on the left of the enemy
            player_momentum[0] -= 1 # send player leftward
            enemy1_momentum[0] += 5 # send enemy rightward
        else:
            player_momentum[0] += 1 # send player rightward
            enemy1_momentum[0] -= 5 # send enemy leftward
    else:
        player_momentum[1] = -3 # sets the players momentum to push them into the air
        if player_rect[0] - 16 < enemy1_rect[0]: # if the player is on the left of the enemy
            player_momentum[0] -= 2 # send player leftward
            enemy1_momentum[0] += 4 # send enemy rightward
        else:
            player_momentum[0] += 2 # send player rightward
            enemy1_momentum[0] -= 4 # send enemy leftward
```

Figure 31: Changing the knockback dynamic so that the enemy is set back instead of the player

A problem I counter using this code is that while the knockback works, attacking the enemy too many times causes their momentum to jump and send them out of the screen. I will fix this by adding cooldowns into the players attack as this also creates a sense of difficulty in the game (rather than allowing the player to smash to enemy against a wall with no cooldown).



```
cooldown = 0
```

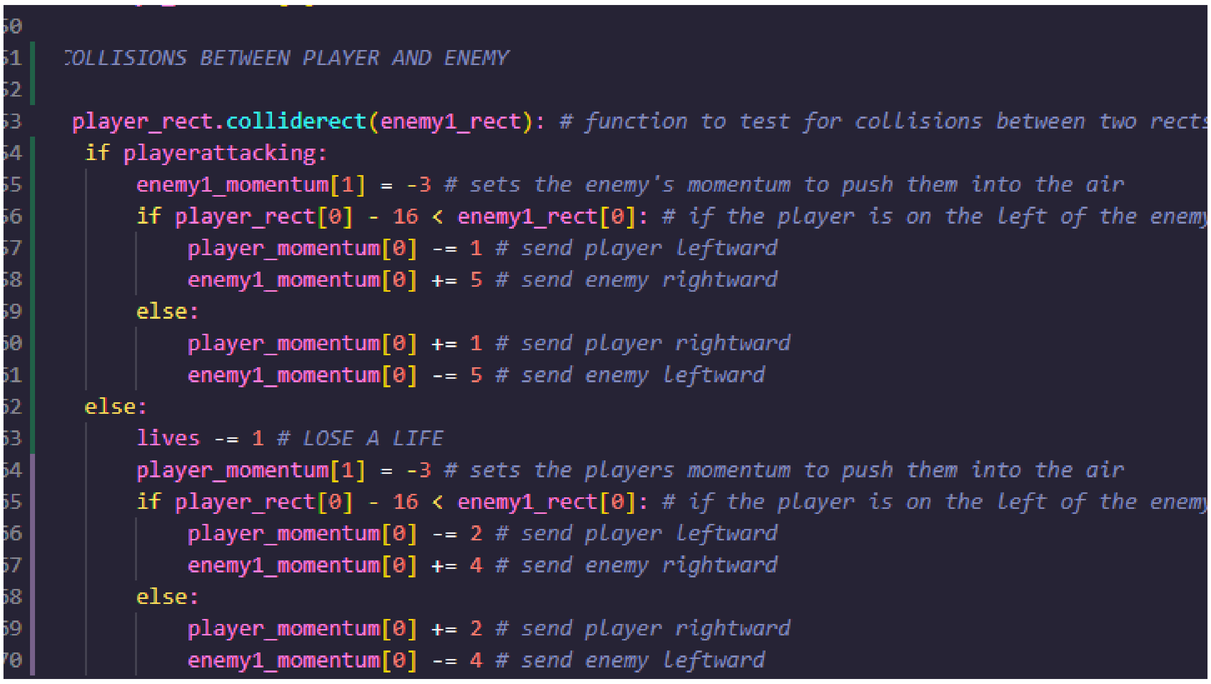


```
    if playerattacking: # if player is attacking
        player_image = pygame.image.load('Assets\Sprites\Player_attack.png') # show attacking sprite and begin countdown
        attacktimer -= 1
        if attacktimer < 1: # when countdown hits 0
            player_image = pygame.image.load('Assets\Sprites\Player_50.png') # show normal sprite and disable attacking state
            playerattacking = False
            attacktimer = 10
            cooldown = 45
    cooldown -= 1
```

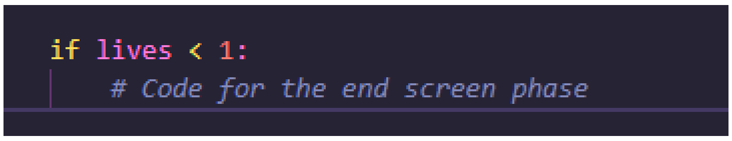
Adding a cooldown variable and setting it to decrement fixes the problem of sending the enemy through tiles while also keeping the momentum varying. This allows for critical hits where if the player attacks the enemy mid-air the enemy is sent far.

COMBINING LIVES AND ATTACKING STATE

I can now combine the two parts of the game so that the health of the player goes down when he takes damage from the enemy. And when the player health hits 0 it goes to an end screen phase. Which I will cover next.



```
COLLISIONS BETWEEN PLAYER AND ENEMY

 player_rect.colliderect(enemy1_rect): # function to test for collisions between two rects
  if playerattacking:
      enemy1_momentum[1] = -3 # sets the enemy's momentum to push them into the air
      if player_rect[0] - 16 < enemy1_rect[0]: # if the player is on the left of the enemy
          player_momentum[0] -= 1 # send player leftward
          enemy1_momentum[0] += 5 # send enemy rightward
      else:
          player_momentum[0] += 1 # send player rightward
          enemy1_momentum[0] -= 5 # send enemy leftward
  else:
      lives -= 1 # LOSE A LIFE
      player_momentum[1] = -3 # sets the players momentum to push them into the air
      if player_rect[0] - 16 < enemy1_rect[0]: # if the player is on the left of the enemy
          player_momentum[0] -= 2 # send player leftward
          enemy1_momentum[0] += 4 # send enemy rightward
      else:
          player_momentum[0] += 2 # send player rightward
          enemy1_momentum[0] -= 4 # send enemy leftward
```



```
if lives < 1:
    # Code for the end screen phase
```

THINGS TO WORK ON

One of the biggest problems in this module is that sometimes the enemy will perform multiple collisions in one interaction, causing the player to lose multiple hearts and die early, as well as create some weird momentum.

Regarding quality of gameplay, there are a lot of things I'd like to work on in this phase as it is the most important, some of which being cooldowns, animations, extended map generation, enemy rework etc. However, in order to complete the minimum viable product, I will move onto the end screen phase and REVISIT the main phase in beta.

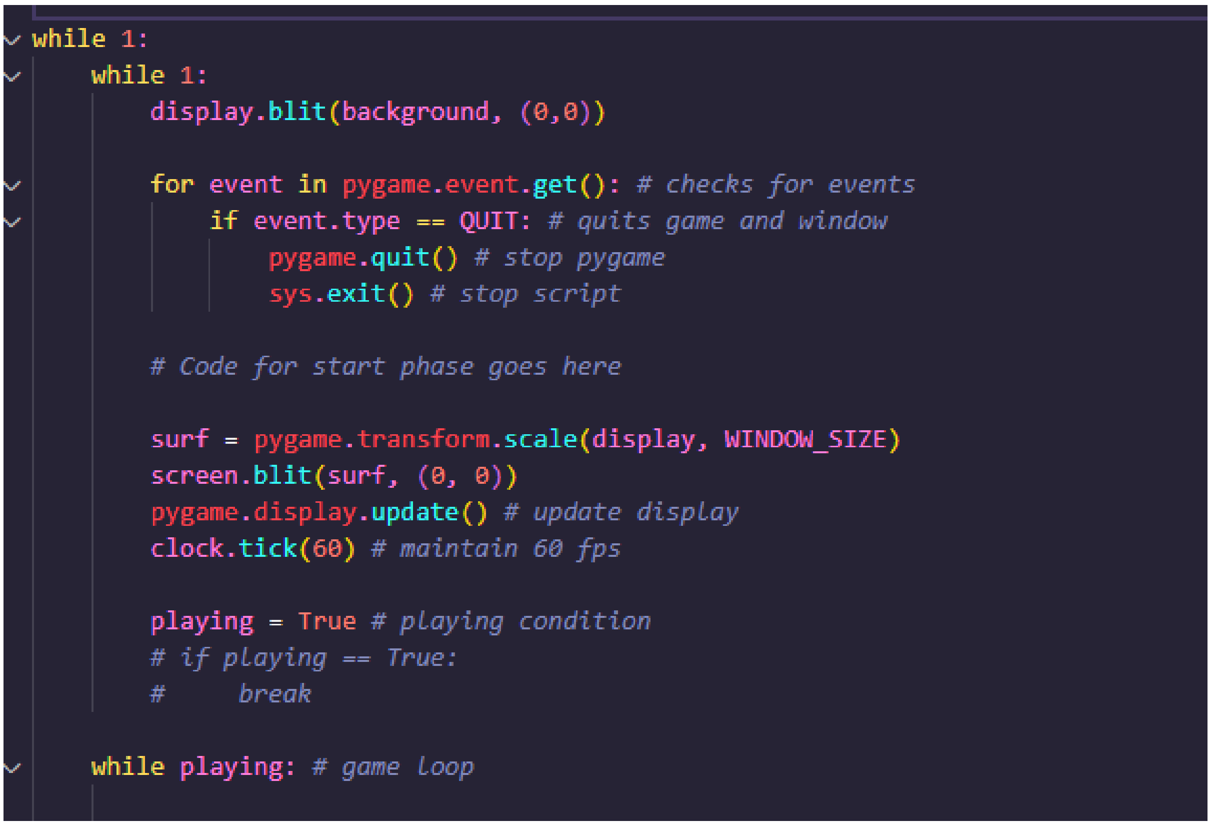
MODULE 3 - START PHASE

This module will serve as an opening screen for the game as well as allow the player to change key binds hopefully.

CREATING SCREEN AND BUTTONS

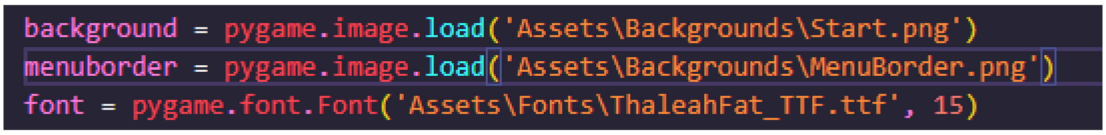
BUTTON DESIGN

I'll begin by editing the main phase and placing it inside while loop that the beginning phase is also in. This means that after the beginning screen closes the main phase should begin itself. Then initialising the screen to run properly like in the main phase (I can simply use the code word for word).



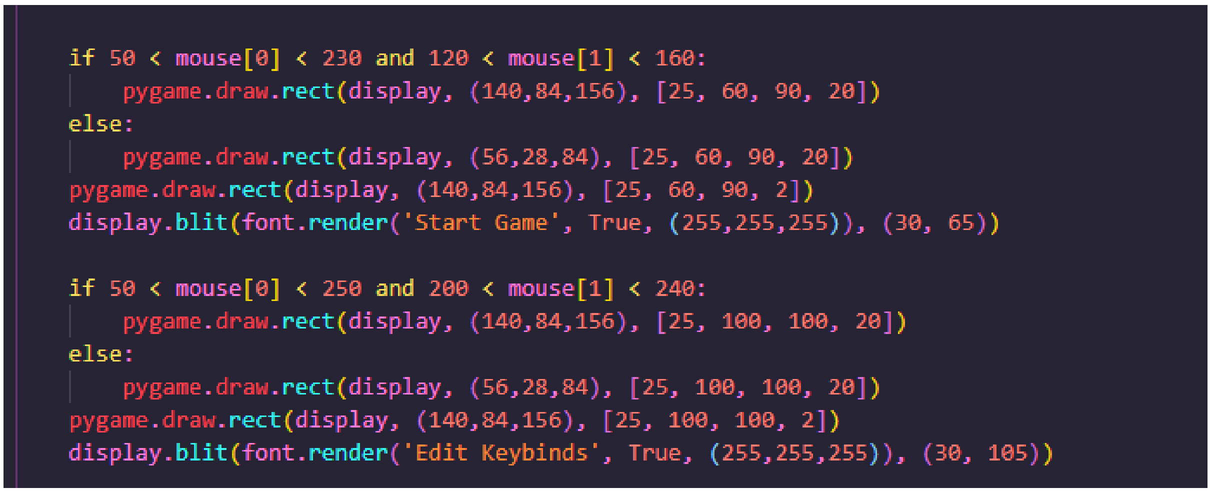
```
while 1:
    while 1:
        display.blit(background, (0,0))

        for event in pygame.event.get(): # checks for events
            if event.type == QUIT: # quits game and window
                pygame.quit() # stop pygame
                sys.exit() # stop script

        # Code for start phase goes here

        surf = pygame.transform.scale(display, WINDOW_SIZE)
        screen.blit(surf, (0, 0))
        pygame.display.update() # update display
        clock.tick(60) # maintain 60 fps

        playing = True # playing condition
        # if playing == True:
        #     break

    while playing: # game loop
```

Figure 32: This allows the screen to run and be exited, I've also stored new variables of background and mousePos using pygame.image.load() and pygame.mouse.get_pos() respectively



```
background = pygame.image.load('Assets\Backgrounds\Start.png')
menuborder = pygame.image.load('Assets\Backgrounds\MenuBorder.png')
font = pygame.font.Font('Assets\Fonts\ThaleahFat_TTF.ttf', 15)
```

After playing around with co-ordinates, I ended up placing the start button and edit key binds button as below:



```
if 50 < mouse[0] < 230 and 120 < mouse[1] < 160:
    pygame.draw.rect(display, (140,84,156), [25, 60, 90, 20])
else:
    pygame.draw.rect(display, (56,28,84), [25, 60, 90, 20])
pygame.draw.rect(display, (140,84,156), [25, 60, 90, 2])
display.blit(font.render('Start Game', True, (255,255,255)), (30, 65))

if 50 < mouse[0] < 250 and 200 < mouse[1] < 240:
    pygame.draw.rect(display, (140,84,156), [25, 100, 100, 20])
else:
    pygame.draw.rect(display, (56,28,84), [25, 100, 100, 20])
pygame.draw.rect(display, (140,84,156), [25, 100, 100, 2])
display.blit(font.render('Edit Keybinds', True, (255,255,255)), (30, 105))
```

Figure 33: The if statements allow for a change in colour when the mouse hovers over the buttons.

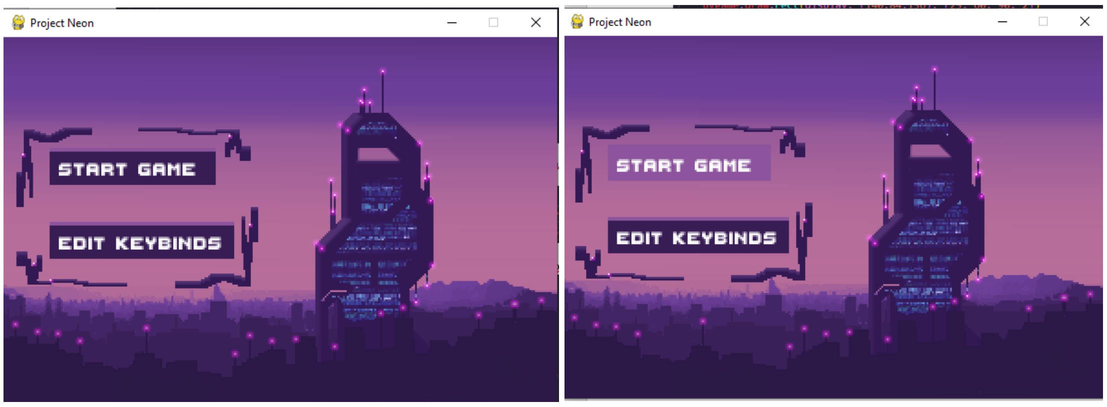
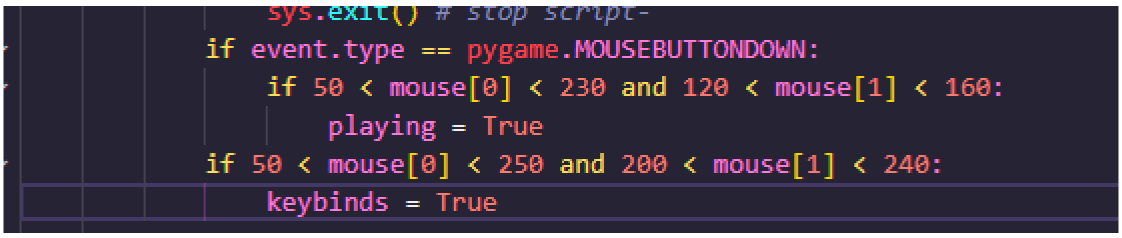


Figure 31 & 32: Without hovering and with hovering

BUTTON FUNCTIONALITY

As the hovering statements already define when the mouse is in bounds of the button, implementing the click functionality should be relatively easy.



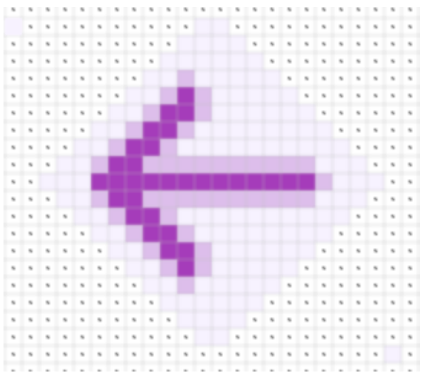
```
            sys.exit() # stop script
        if event.type == pygame.MOUSEBUTTONDOWN:
            if 50 < mouse[0] < 230 and 120 < mouse[1] < 160:
                playing = True
        if 50 < mouse[0] < 250 and 200 < mouse[1] < 240:
            keybinds = True
```

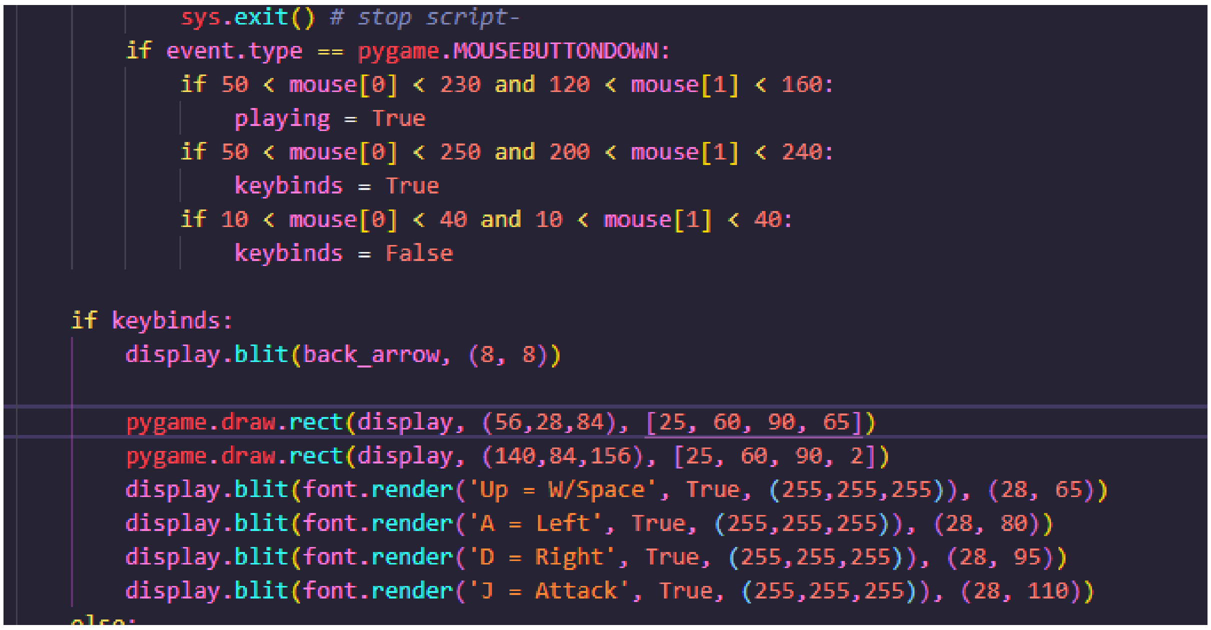
Figure 34: This allows for my player to now jump into the playing phase if Start Game is selected.

Assigning key binds to True means that I can make more buttons show up on the existing screen with if statements without having to enter a new loop and redo a lot of statements.

I would not yet like to tackle the problem of taking user inputs and converting them from letter to Unicode and then into a format that pygame recognises (which is some form of int). Because of time constraints I will come back to this issue later and simply display the existing key binds for now.

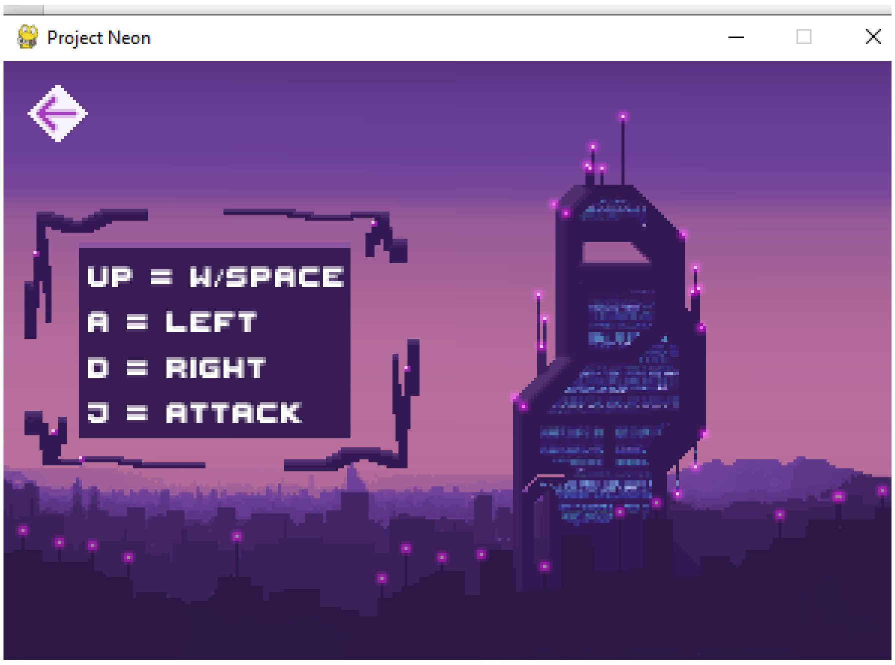
I have made a back icon using an online website pixel designer and will use this to go back from the key binds screen.





```
            sys.exit() # stop script-
        if event.type == pygame.MOUSEBUTTONDOWN:
            if 50 < mouse[0] < 230 and 120 < mouse[1] < 160:
                playing = True
            if 50 < mouse[0] < 250 and 200 < mouse[1] < 240:
                keybinds = True
            if 10 < mouse[0] < 40 and 10 < mouse[1] < 40:
                keybinds = False

    if keybinds:
        display.blit(back_arrow, (8, 8))

        pygame.draw.rect(display, (56,28,84), [25, 60, 90, 65])
        pygame.draw.rect(display, (140,84,156), [25, 60, 90, 2])
        display.blit(font.render('Up = W/Space', True, (255,255,255)), (28, 65))
        display.blit(font.render('A = Left', True, (255,255,255)), (28, 80))
        display.blit(font.render('D = Right', True, (255,255,255)), (28, 95))
        display.blit(font.render('J = Attack', True, (255,255,255)), (28, 110))
```

Using this code to display the movement and combat controls produces this:



```
UP = W/SPACE
A = LEFT
D = RIGHT
J = ATTACK
```

With this being done I can move onto the end phase screen, however before that I need to make sure my score counter works so that there is something to display at the end screen

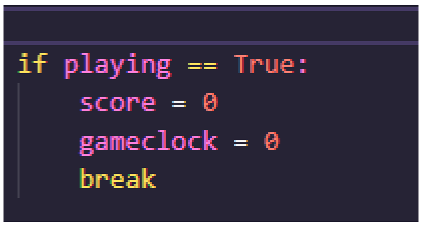
THINGS TO WORK ON

I'm relatively happy with the state of the start phase, with minimal improvements in mind except for maybe the redesign of the back button to fit the background and allowing users to enter their own key binds.

MODULE 4 – SCORE COUNTER

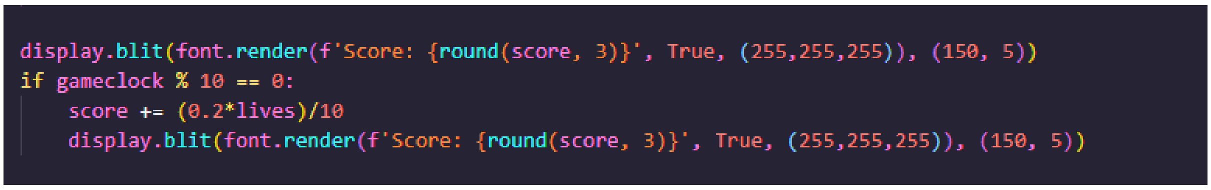
This will be a relatively small module which serves as a requirement for the end phase, which will be module 5.

My plan for the score counter is a timer that goes up every 30 seconds in a linear way, the score will be the multiple of the timer and the number of lives remaining.



```
if playing == True:
    score = 0
    gameclock = 0
    break
```

Figure 35: Initialising score to 0 and a clock that will count up in ticks, the game clock will increment in ones and every 10 ticks the score will update



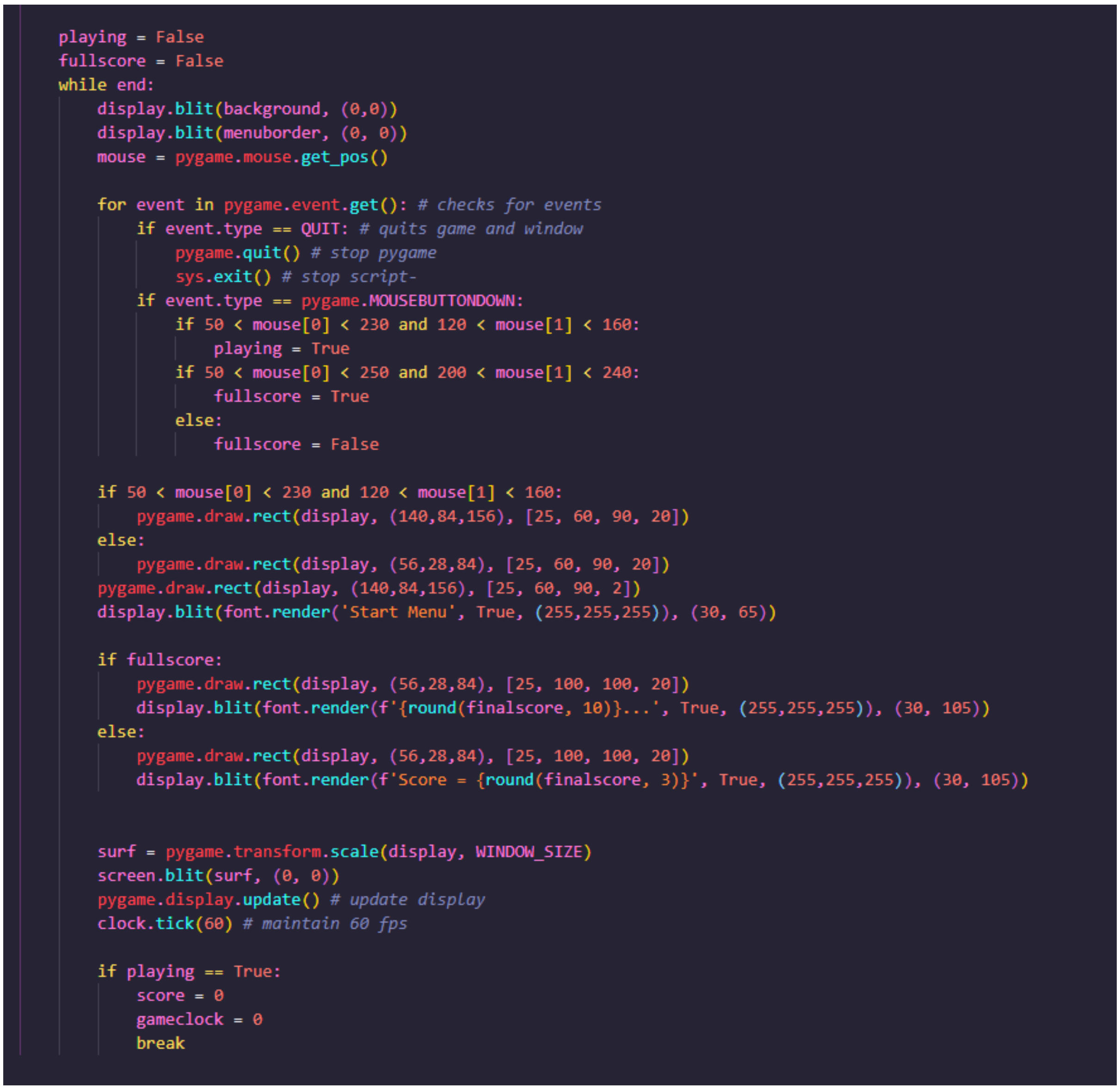
```
display.blit(font.render(f'Score: {round(score, 3)}', True, (255,255,255)), (150, 5))
if gameclock % 10 == 0:
    score += (0.2*lives)/10
    display.blit(font.render(f'Score: {round(score, 3)}', True, (255,255,255)), (150, 5))
```

Figure 36: Every 10 ticks the score updates, increment in lower values if the player has lost health

Now that a score is properly setup, I can move onto the end phase.

MODULE 5 – END PHASE

Similarly, to the start phase, the end phase is a small screen that will only serve as a divider between game loops. In this screen I will have the players score and the option to go back to the start screen.



```
    playing = False
    fullscore = False
    while end:
        display.blit(background, (0,0))
        display.blit(menuborder, (0, 0))
        mouse = pygame.mouse.get_pos()

        for event in pygame.event.get(): # checks for events
            if event.type == QUIT: # quits game and window
                pygame.quit() # stop pygame
                sys.exit() # stop script-
            if event.type == pygame.MOUSEBUTTONDOWN:
                if 50 < mouse[0] < 230 and 120 < mouse[1] < 160:
                    playing = True
                if 50 < mouse[0] < 250 and 200 < mouse[1] < 240:
                    fullscore = True
                else:
                    fullscore = False

        if 50 < mouse[0] < 230 and 120 < mouse[1] < 160:
            pygame.draw.rect(display, (140,84,156), [25, 60, 90, 20])
        else:
            pygame.draw.rect(display, (56,28,84), [25, 60, 90, 20])
        pygame.draw.rect(display, (140,84,156), [25, 60, 90, 2])
        display.blit(font.render('Start Menu', True, (255,255,255)), (30, 65))

        if fullscore:
            pygame.draw.rect(display, (56,28,84), [25, 100, 100, 20])
            display.blit(font.render(f'{round(finalscore, 10)}...', True, (255,255,255)), (30, 105))
        else:
            pygame.draw.rect(display, (56,28,84), [25, 100, 100, 20])
            display.blit(font.render(f'Score = {round(finalscore, 3)}', True, (255,255,255)), (30, 105))


        surf = pygame.transform.scale(display, WINDOW_SIZE)
        screen.blit(surf, (0, 0))
        pygame.display.update() # update display
        clock.tick(60) # maintain 60 fps

        if playing == True:
            score = 0
            gameclock = 0
            break
```

This code is extremely similar to that of the start screen. Just with different options and functions.

One problem I have noticed is that with the creation of a full loop, the main phase does not start in its default position. I will temporarily fix this by replacing the start again button with a "Quit" button, and then if it becomes a popular player concern, I will attempt to refactor code and allow it to loop seamlessly.

ALPHA PEER FEEDBACK

I have sent the game to a few select willing peers and they have given the below feedback which I will aim to include in my beta phase

- Include a "?" symbol to help players who might not understand the objective of the game, and when that symbol is clicked display text along the lines of "attack x opponents to reach y stage and win!"
- Make collisions occur one at a time and patch multiple lives lost in one collision, improve combat system so it's more clean and visually understandable.
- Give the playing phase a stronger theme/design

My personal goals for the beta are arguably bigger than the Alpha phase (meaning I hope that the game will have the biggest jump in quality during this next phase), I plan to keep refining code in cycles until it has a viable good product, and I should also look to include complexity.

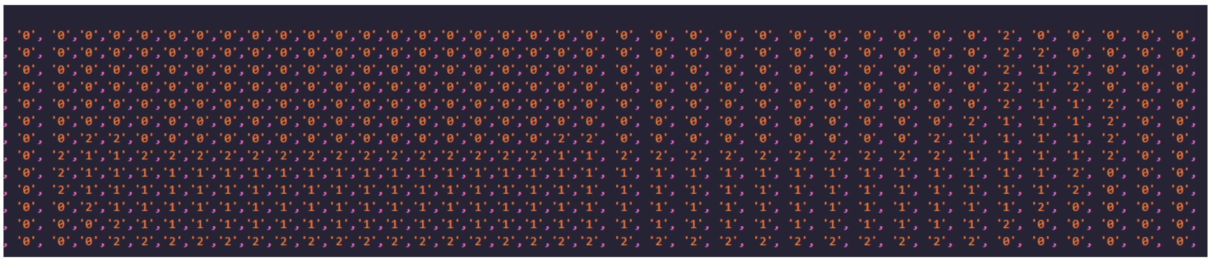
BETA

My basic game is complete, this means that all the major requirements outlined in the success criteria have been met in order to make the game suitable for my target audience, the beta phase will look into feedback from all my peers and making quality of life improvements as well as bug patches/meeting minor requirements.

CYCLE 1

MAIN PHASE RETEXTURING

As the game is now functional, I can replace the dirt textures with accurate textures that represent the theme of my game properly. I will also redo the map, making it more of an arena and making it much larger. This will help satiate the requests of one of the peers in alpha peer feedback



```
, '0', '0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '2', '0', '0', '0', '0', '0',
, '0', '0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '2', '2', '0', '0', '0', '0',
, '0', '0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '2', '1', '2', '0', '0', '0',
, '0', '0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '2', '1', '2', '0', '0', '0',
, '0', '0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '2', '1', '1', '2', '0', '0',
, '0', '0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '2', '1', '1', '1', '2', '0', '0',
, '0', '0','2','2','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','2','2', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '2', '1', '1', '1', '1', '2', '0', '0',
, '0', '2','1','1','2','2','2','2','2','2','2','2','2','2','2','2','2','2','2','1','1', '2', '2', '2', '2', '2', '2', '2', '2', '2', '2', '1', '1', '1', '1', '2', '0', '0',
, '0', '2','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '2', '0', '0', '0',
, '0', '2','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '2', '0', '0', '0',
, '0', '0','2','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '2', '0', '0', '0', '0',
, '0', '0','0','2','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '2', '0', '0', '0', '0', '0',
, '0', '0','0','2','2','2','2','2','2','2','2','2','2','2','2','2','2','2','2','2','2', '2', '2', '2', '2', '2', '2', '2', '2', '2', '2', '2', '0', '0', '0', '0', '0', '0',
```

Figure 37: Revised map, I will change it depending on player feedback from now on.

Now creating the new map tiles:

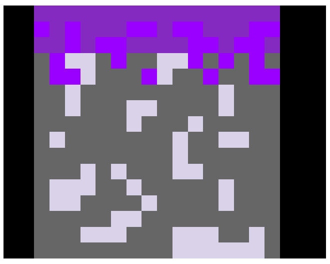


Figure 39: New Top Tile

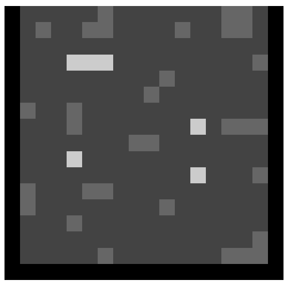
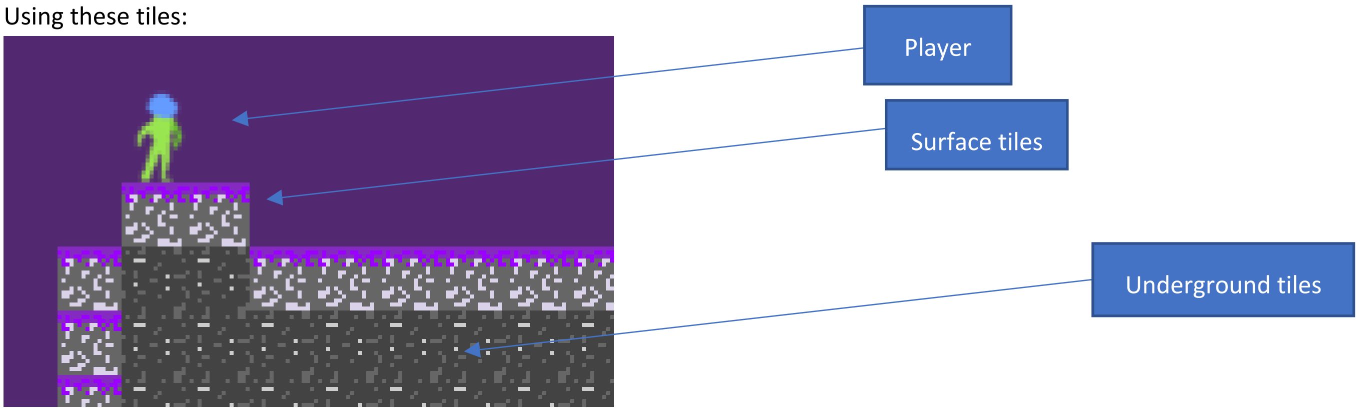
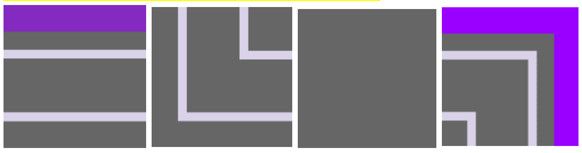


Figure 38: Stone Tile

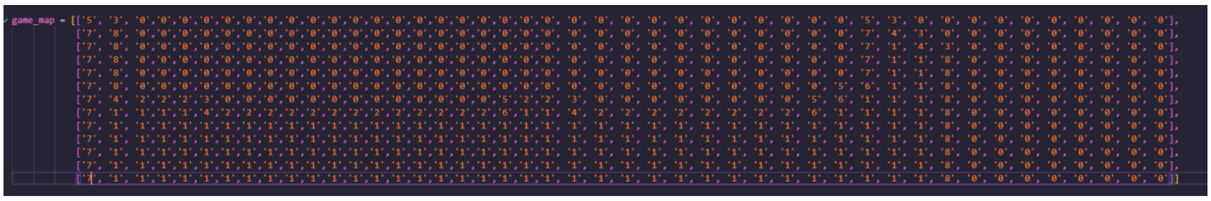
Using these tiles:



I feel like there is too many details in the tiles so I will try an alternative style of design. By having a simpler design the player will feel more comfortable playing the game as there is a bigger focus on the functional combat rather than the aesthetics.



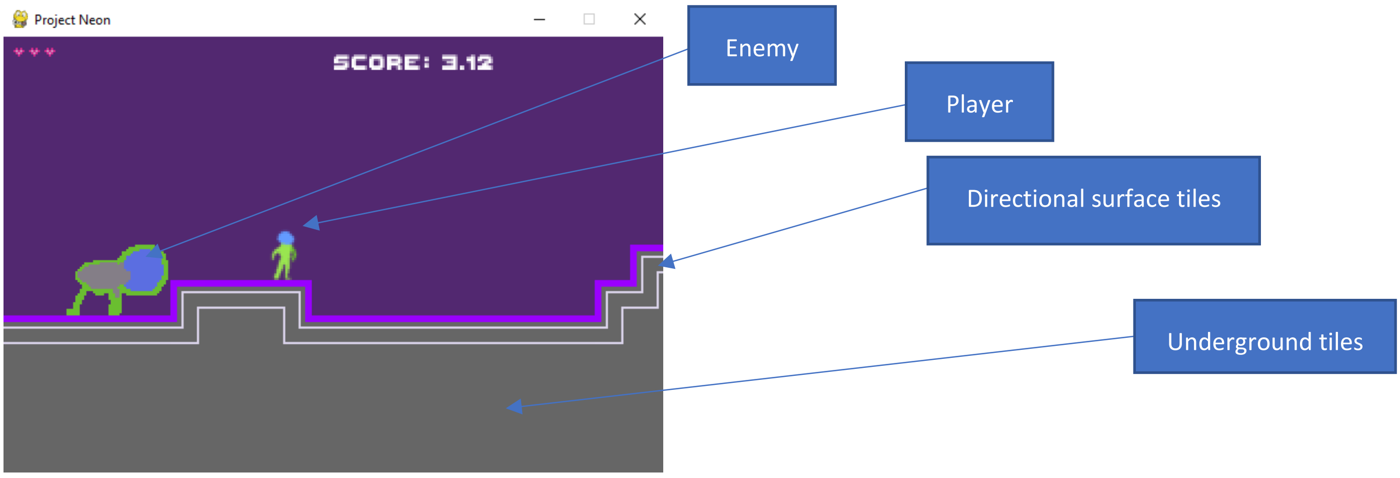
using this new set and redoing the map as below:



```
game_map = [['5', '3', '0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '5', '3', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0'],
            ['7', '8', '0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '7', '4', '3', '0', '0', '0', '0', '0', '0', '0', '0', '0'],
            ['7', '8', '0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '7', '1', '4', '3', '0', '0', '0', '0', '0', '0', '0', '0'],
            ['7', '8', '0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '7', '1', '1', '8', '0', '0', '0', '0', '0', '0', '0', '0'],
            ['7', '8', '0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '7', '1', '1', '8', '0', '0', '0', '0', '0', '0', '0', '0'],
            ['7', '8', '0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '5', '6', '1', '1', '8', '0', '0', '0', '0', '0', '0', '0', '0'],
            ['7', '4', '2','2','2','3','0','0','0','0','0','0','0','0','0','0','0','0','0','5','2','2', '3', '0', '0', '0', '0', '0', '0', '0', '0', '5', '6', '1', '1', '1', '8', '0', '0', '0', '0', '0', '0', '0', '0'],
            ['7', '1', '1','1','1','4','2','2','2','2','2','2','2','2','2','2','2','2','2','6','1','1', '4', '2', '2', '2', '2', '2', '2', '2', '2', '6', '1', '1', '1', '1', '8', '0', '0', '0', '0', '0', '0', '0', '0'],
            ['7', '1', '1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '8', '0', '0', '0', '0', '0', '0', '0', '0'],
            ['7', '1', '1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '8', '0', '0', '0', '0', '0', '0', '0', '0'],
            ['7', '1', '1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '8', '0', '0', '0', '0', '0', '0', '0', '0'],
            ['7', '1', '1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '8', '0', '0', '0', '0', '0', '0', '0', '0'],
            ['7', '1', '1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '8', '0', '0', '0', '0', '0', '0', '0', '0']]
```

Figure 40: Each number represents a different type of tile

I have fixed the map to now avoid players jumping out of it as well as making it fit the game theme.



REFINING COLLISIONS & LIFE DEDUCTION

Another mentioned problem in feedback was collisions.

As of now the enemy can perform multiple collisions in one interaction with the player, making them not only lose multiple lives but deal extended knockback.

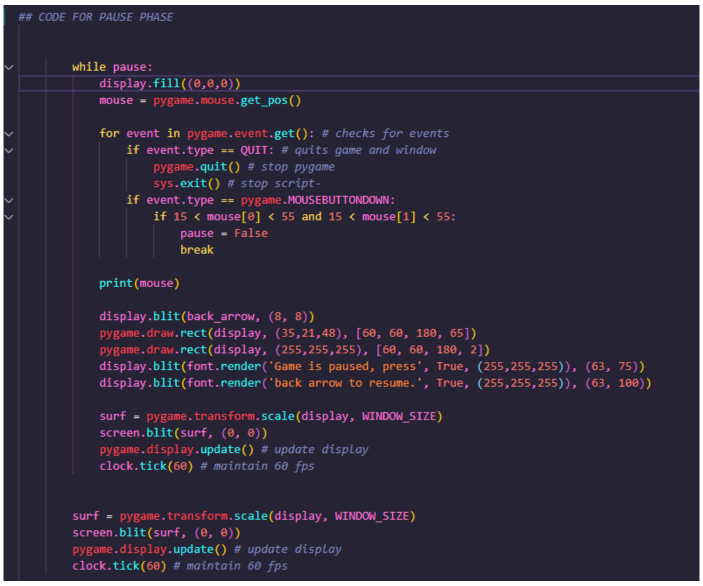
The root problem of this is how my collisions impact my entities, so I will play around with these variables so that the entities are not sent too far but not close enough to cause an instantaneous second collision.

CYCLE 2 – FINAL CYCLE

Due to time constraints, I will finish with one last cycle and attempt to add as many features as possible, briefly listing them as well as the code below.

PAUSE PHASE

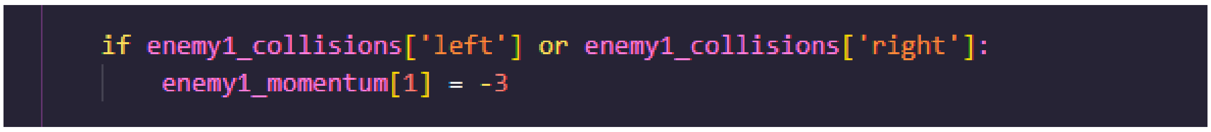
I created this pause phase mainly by copying the code from the key binds menu and re-ordering objects and changing text, this phase serves to allow players to stop mid game for whatever reason they may need to. This meets one of the smaller criteria in the analysis stage as well as completes one of the modules in my design phase.



```
## CODE FOR PAUSE PHASE


    while pause:
        display.fill((0,0,0))
        mouse = pygame.mouse.get_pos()

        for event in pygame.event.get(): # checks for events
            if event.type == QUIT: # quits game and window
                pygame.quit() # stop pygame
                sys.exit() # stop script-
            if event.type == pygame.MOUSEBUTTONDOWN:
                if 15 < mouse[0] < 55 and 15 < mouse[1] < 55:
                    pause = False
                    break

        print(mouse)

        display.blit(back_arrow, (8, 8))
        pygame.draw.rect(display, (35,21,48), [60, 60, 180, 65])
        pygame.draw.rect(display, (255,255,255), [60, 60, 180, 2])
        display.blit(font.render('Game is paused, press', True, (255,255,255)), (63, 75))
        display.blit(font.render('back arrow to resume.', True, (255,255,255)), (63, 100))

        surf = pygame.transform.scale(display, WINDOW_SIZE)
        screen.blit(surf, (0, 0))
        pygame.display.update() # update display
        clock.tick(60) # maintain 60 fps


    surf = pygame.transform.scale(display, WINDOW_SIZE)
    screen.blit(surf, (0, 0))
    pygame.display.update() # update display
    clock.tick(60) # maintain 60 fps
```

ENEMY COMBAT

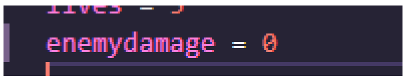
I would like to finish up enemy combat, this was one of the main concerns from peers who had tried the game. I will attempt to improve my system by giving the enemy a health meter as well as refined movement such that it can jump over obstacles.



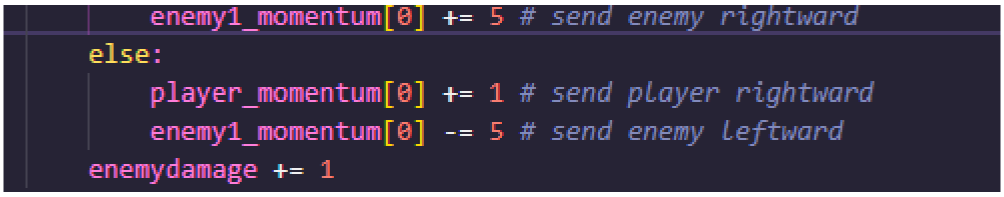
```
    if enemy1_collisions['left'] or enemy1_collisions['right']:
        enemy1_momentum[1] = -3
```

Figure 41: Simple added code so that when the enemy collides with anything on its left or right, it performs a jump

In my planning phase I discussed that I wanted my game to last with the user as the main constraint, meaning that it would go on until the user dies. I would like to follow through with this and therefore instead of the enemy having lives which decrements, it'll instead have a damage number which will increase depending on how many times the enemy was hit.

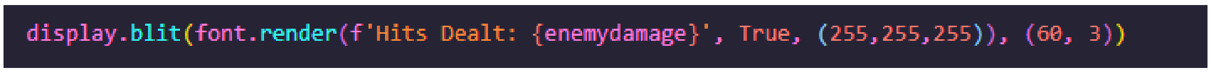


```
enemydamage = 0
```



```
            enemy1_momentum[0] += 5 # send enemy rightward
        else:
            player_momentum[0] += 1 # send player rightward
            enemy1_momentum[0] -= 5 # send enemy leftward
        enemydamage += 1
```

Now that the damage is setup, I'll add the GUI (Graphical User Interface) as a visual representation of how many hits have been dealt to the enemy.



```
display.blit(font.render(f'Hits Dealt: {enemydamage}', True, (255,255,255)), (60, 3))
```

Figure 42: Text display in the middle top of the screen that represents how many times the enemy has been hit.

Adding this meant that I had to move the score counter further to the right, as otherwise there would have been no space for hit counter on the top of the screen.

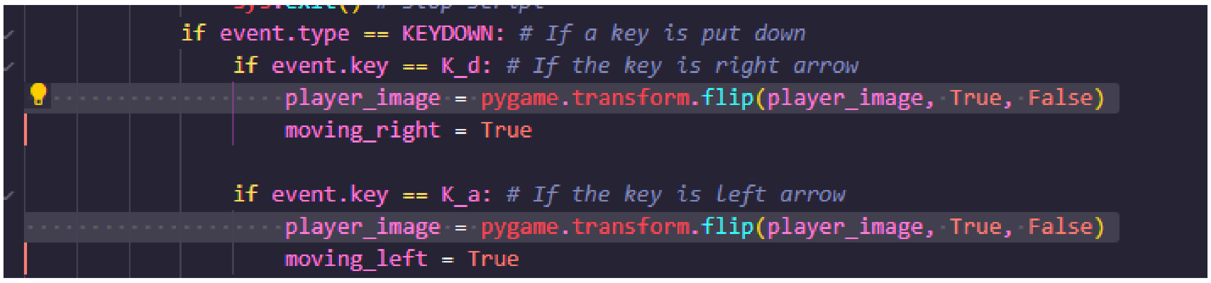
Adding the counter also had me realise that sometimes the player would inflict damage on the enemy multiple times during one physical collision, so I solved this similar to how I solved the player collision issue and added an enemy cooldown timer.

FIXING SCORE TO INCLUDE COMBAT COUNTER

Now that there is a hit counter, I can more accurately create a representation of score, rewarding players for how much damage they deal to the enemy.

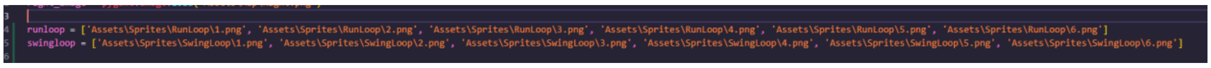
SPRITE CORRECTIONS + REDESIGN

Adding in the below lines, I'm able to have the player face the direction she is walking in.

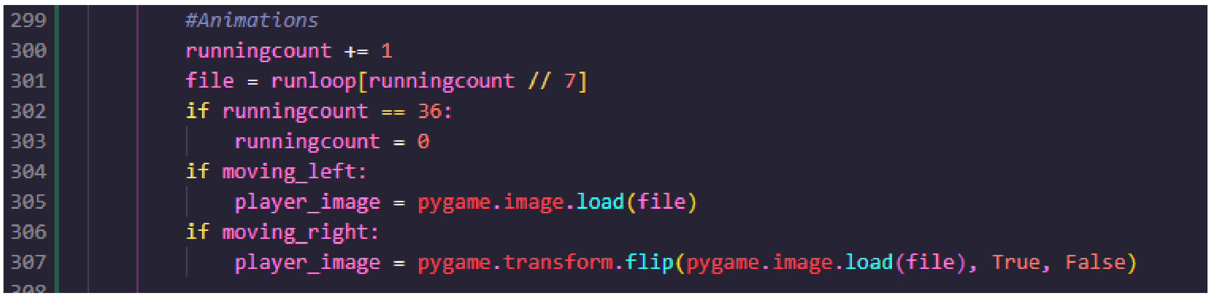


```
            if event.type == KEYDOWN: # If a key is put down
                if event.key == K_d: # If the key is right arrow
                    player_image = pygame.transform.flip(player_image, True, False)
                    moving_right = True

                if event.key == K_a: # If the key is left arrow
                    player_image = pygame.transform.flip(player_image, True, False)
                    moving_left = True
```

Now for a quick redesign of the sprite as well adding a swinging, jumping and walking animation.

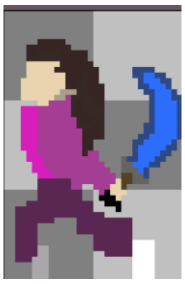


```
runloop = ['Assets\Sprites\RunLoop\1.png', 'Assets\Sprites\RunLoop\2.png', 'Assets\Sprites\RunLoop\3.png', 'Assets\Sprites\RunLoop\4.png', 'Assets\Sprites\RunLoop\5.png', 'Assets\Sprites\RunLoop\6.png']
swingloop = ['Assets\Sprites\SwingLoop\1.png', 'Assets\Sprites\SwingLoop\2.png', 'Assets\Sprites\SwingLoop\3.png', 'Assets\Sprites\SwingLoop\4.png', 'Assets\Sprites\SwingLoop\5.png', 'Assets\Sprites\SwingLoop\6.png']
```



```
        #Animations
        runningcount += 1
        file = runloop[runningcount // 7]
        if runningcount == 36:
            runningcount = 0
        if moving_left:
            player_image = pygame.image.load(file)
        if moving_right:
            player_image = pygame.transform.flip(pygame.image.load(file), True, False)
```

Now the player has movement based on the keys being pressed, as well as a prettier character design animation:



Lastly, fixing combat animations:

This part of documentation was done without access to my original python file, so I used GitHub snippets to display code.

Using sets of pngs (available to see in GitHub repository) and having them play after each other using the below code:

```
#Animations
runningcount += 1
file = runloop[runningcount // 7]
if runningcount == 36:
    runningcount = 0
if moving_left:
    player_image = pygame.image.load(file)
if moving_right:
    player_image = pygame.transform.flip(pygame.image.load(file), True, False)

if playerattacking:
    file = swingloop[swingcount]
    swingcount += 1
    if swingcount == 12:
        swingcount = 0
    if moving_left:
        player_image = pygame.image.load(file)
    else:
        player_image = pygame.transform.flip(pygame.image.load(file), True, False)
```

This creates a moving animation as well as an attacking animation for my player.

ICON DESIGN

I have finally designed an icon using the background to finish up with game production:

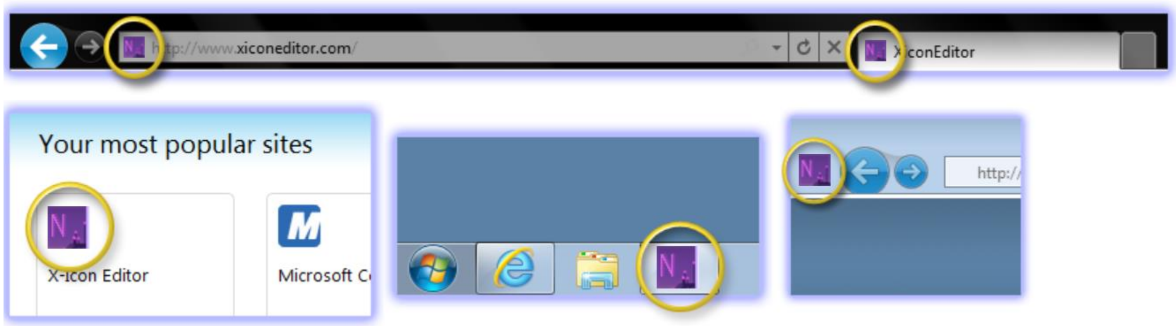


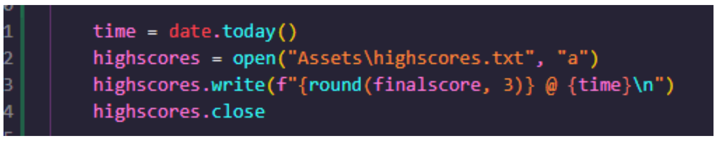
Figure 43: Preview of how the icon will look

COMPLEXITY CONSIDERATIONS

Although I already have some signs of complexity (hardware communication, library usage etc.), I'd like to add more aspects of complexity to the project through storing data, algorithms and data structures.

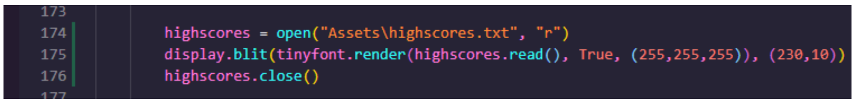
STORING HIGH SCORE AND DISPLAYING ON THE MAIN SCREEN

First adding the code just before the end phase to store the high score:



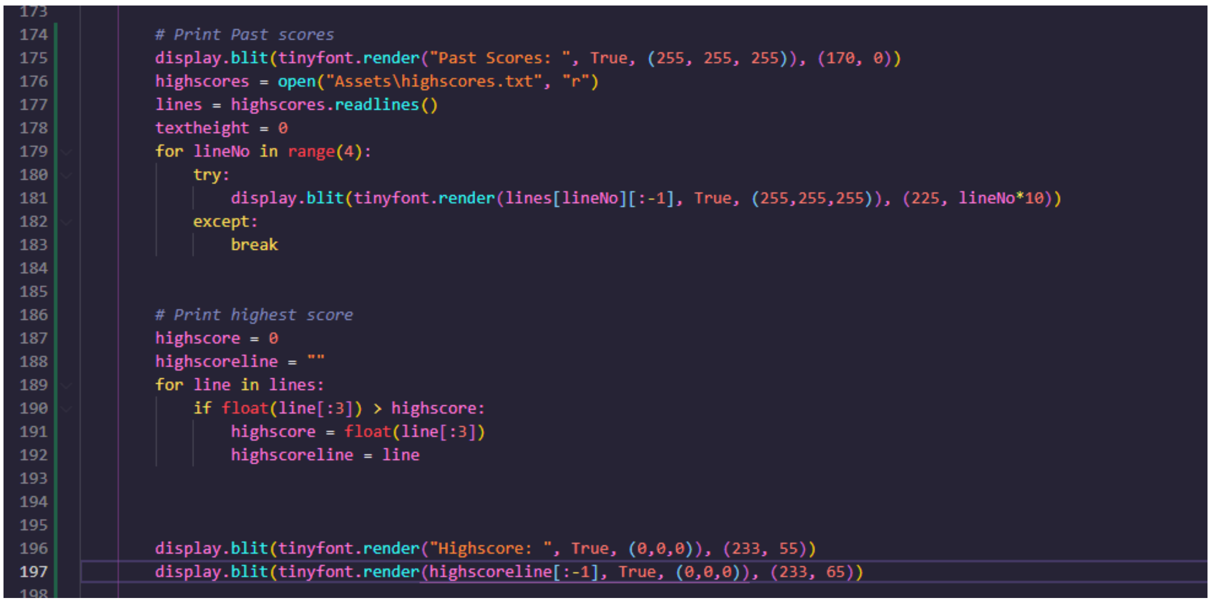
```python
time = date.today()
highscores = open("Assets\highscores.txt", "a")
highscores.write(f"{round(finalscore, 3)} @ {time}\n")
highscores.close
```

And then adding a way to print these scores in the start screen.



```python
highscores = open("Assets\highscores.txt", "r")
display.blit(tinyfont.render(highscores.read(), True, (255,255,255)), (230,10))
highscores.close()
```

Revising this output, making it print the past scores and high score:



```python
# Print Past scores
display.blit(tinyfont.render("Past Scores: ", True, (255, 255, 255)), (170, 0))
highscores = open("Assets\highscores.txt", "r")
lines = highscores.readlines()
textheight = 0
for lineNo in range(4):
    try:
        display.blit(tinyfont.render(lines[lineNo][:-1], True, (255,255,255)), (225, lineNo*10))
    except:
        break


# Print highest score
highscore = 0
highscoreline = ""
for line in lines:
    if float(line[:3]) > highscore:
        highscore = float(line[:3])
        highscoreline = line



display.blit(tinyfont.render("Highscore: ", True, (0,0,0)), (233, 55))
display.blit(tinyfont.render(highscoreline[:-1], True, (0,0,0)), (233, 65))
```

The final screen is as:

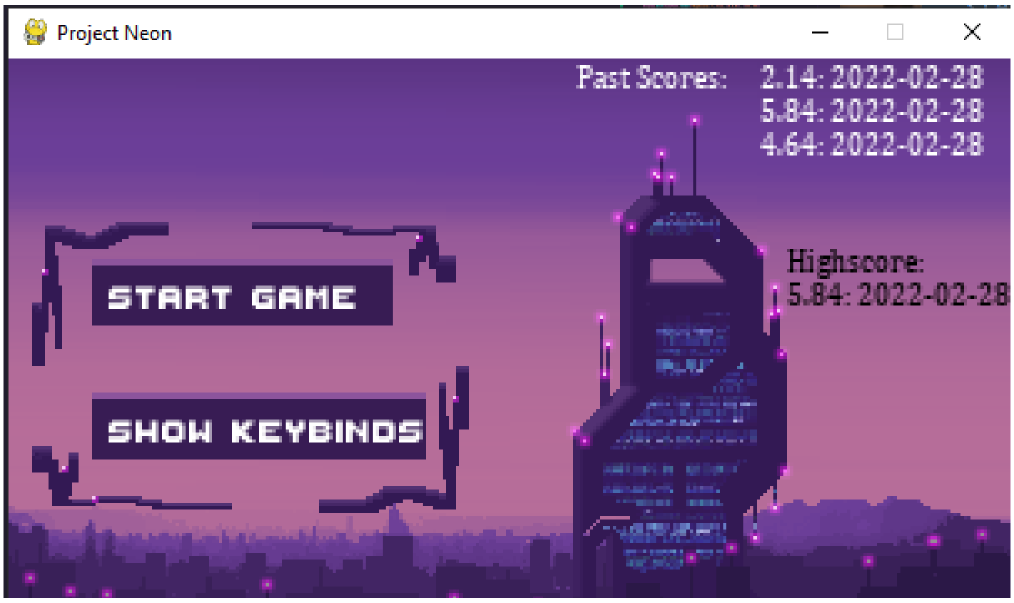


Figure 44: I've deviated from my original font so that this text could fit in smaller areas without compromising quality.

I feel that having this text somewhat bloats the screen and negatively impacts the design. If I'm able, I would like to move these scores onto a separate screen that can be accessed using a new button (which I would probably place in the middle of 'Start Game' and 'Show Keybinds')

EVALUATION

Due to time constraints, I've moved onto the evaluation phase early.

EVALUATION TESTING AGAINST TEST DATA + VALIDATION OF KEY ELEMENTS IN SOLUTION

SCORES

Test Data	Expected Output	Real Output	Pass or Fail
Non-Player entity is killed	Score increases	Score Increases	Pass
Player is killed	Score stays the same	Score stays the same	Pass
Score in game end phase is higher than high score	Score becomes new high score	Score is listed in end screen	Fail, no score system developed

I'm satisfied with the development of my scores, although it's not completely finished (lacking the score system), people can see their scores at the end, and it still serves its purpose – to encourage competition and repeated attempts.

HIGH SCORE NAME

Test Data	Expected Output	Real Output	Pass or Fail
Alphanumerical (letters and numbers)	Valid output, name is added to score list	N/A	Fail, system not built

Non-alphanumerical characters (e.g., special characters)	Invalid output, error message shown	N/A	Fail, system not built

As stated above, this section of the game was unfinished. This was due to the time constraints and also lack of knowledge in setting up/linking the database. In future I should gain an understanding on the topic before attempting to use it in part of my solution.

INPUTS

Test Data	*Expected Output*	*Real Output*	*Pass or Fail*
MOVEMENT keys (WASD by default)	Moves the character	Moves the character	Pass
Attack keys (LMB + RMB)	Makes the characters attack	LMB and RMB do nothing, however the J key performs an attack	Pass, although LMB & RMB were the original attack buttons, J has replaced that.
Other input	Nothing	Nothing	Pass

The inputs were mostly completed, which is vital as it is one of the most important parts of the game. RMB and LMB were replaced by J for attack as I found it better for the player to have both hands on the keyboard (following inspiration from games like Brawlhalla). Considering the edge case as any other input, the game successfully takes no output from the input of these keys.
The one thing I should work on in the future is the functionality of custom key bind assignment, this would give the player more comfortability in playing with their own preferred controls (meaning that people such as those who are left-handed can change keybinds to suit their dominant hand). I was unable to add this as I delayed focus on this component until the more important portions of the game were complete, if I had better time management, I may have been able to add this component into the game.

BUTTONS

Test Data	*Output/Validity*
Buttons pressed	Script related to button is run
No buttons pressed	Nothing happens

This is the original test data for buttons, however since now I know the buttons in my game, I will go into more detail with a reformed test data table.

Test Button	*Expected Output*	*Real Output*	*Pass or Fail*
Start Game	Game/Main phase starts	Game starts	Pass
Show Keybinds	Keybinds are displayed	Keybinds are displayed	Pass
Back arrow	Goes back to main screen	Goes back to main screen (although for a portion of the button the functionality is void)	Fail, doesn't work across the whole button

Escape key on keyboard	Enters pause screen	Enters pause screen	Pass
Back arrow on pause screen	Enters back to playing screen	Enters playing screen	Pass
Clicking on Score in end phase	Shows full score	Shows full score	Pass
Quit Button	Quits game	Quits game	Pass
No Button is pressed	Nothing happens	Nothing happens	Pass

Most of my buttons work aside from one of the back arrows, which have only a small region in which they don't work (however it's quite hard to hit that specific spot as it's near the right edge), edge cases have also given the correct output. I'm satisfied with the design and construction of my buttons and their functionality. In the future I don't think there was much I would've changed apart from the Back Arrow by messing around a bit with the co-ordinates for interaction.

SPRITE ANIMATIONS

Test Data	*Expected Output*	*Real Output*	*Pass or Fail*
MOVEMENT keys (WASD by default)	Moving animations are played	Moving animations are played	Pass
Attack keys (LMB + RMB)	Attack animations are played	Attack animations are played	Pass
Death of an entity	Death animations are played	N/A No time for animations to play	Fail

Although sprites were squeezed in at the end, I'm happy with how they turned out, they add much more life to the game and also allow the player to see when the character is in its attacking phase. In future I'd have hoped to work on making the attack animation longer therefore making the game a little easier for players.
The death of an entity animation is something I'm okay with skipping as the current state of the game only allows the player to die (the enemy has infinite health as there's only one), and so cutting straight to the end phase is also a viable or appropriate way to handle player death.

SOUND

Due to time restrictions, I hadn't made my way around to sound implementation. I considered game sounds to be one of the least important aspects of the game and therefore left it for last (many players can listen to their own music during gameplay and due to the nature of the game the music doesn't add much except some ambience).
Despite this, I had an audio track ready for implementation in the game, which was created with FL Music Studio and is accessible below:

GAME STATES

Test Data	*Expected Output*	*Real Output*	*Pass or Fail*

Start game button pressed	Game moves from start to playing phase.	Game moves from start to playing phase.	Pass
Game paused during playing phase (using escape)	Game moves from start to pause phase	Game moves from start to pause phase.	Pass
Player dies	Game moves from playing to over phase	Game moves from playing to over phase	Pass
Player ends game	Game moves from paused to over phase	N/A No end button available	Fail
Player selects go back to menu	Game moves from over to start phase	N/A No back to menu option	Fail

The main parts of the game states work flawlessly, with the exception that players cannot manually end the game and there is no option from end to start phase. In the future I should consider adding either a few lines that reopens the game file for a fresh start or moving all the main phase into a function so that it can be easily run without permanently changing external variables due to its local scope.

SOLUTION USABILITY FEEDBACK

This will go over finalised peer feedback on what I have created so far. I will ask from each peer to comment on the following 3 topics: Game difficulty, Game navigation, Game understanding and Game design

GAME DIFFICULTY

This heading will cover how difficult peers considered the game to be.
Questions: How hard was the game to play? Why? How would you recommend I change this?

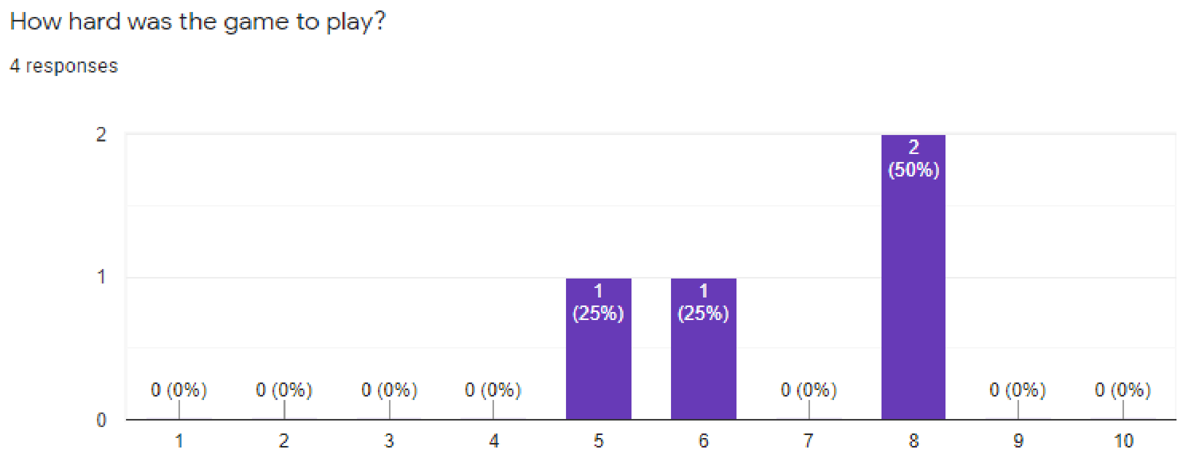
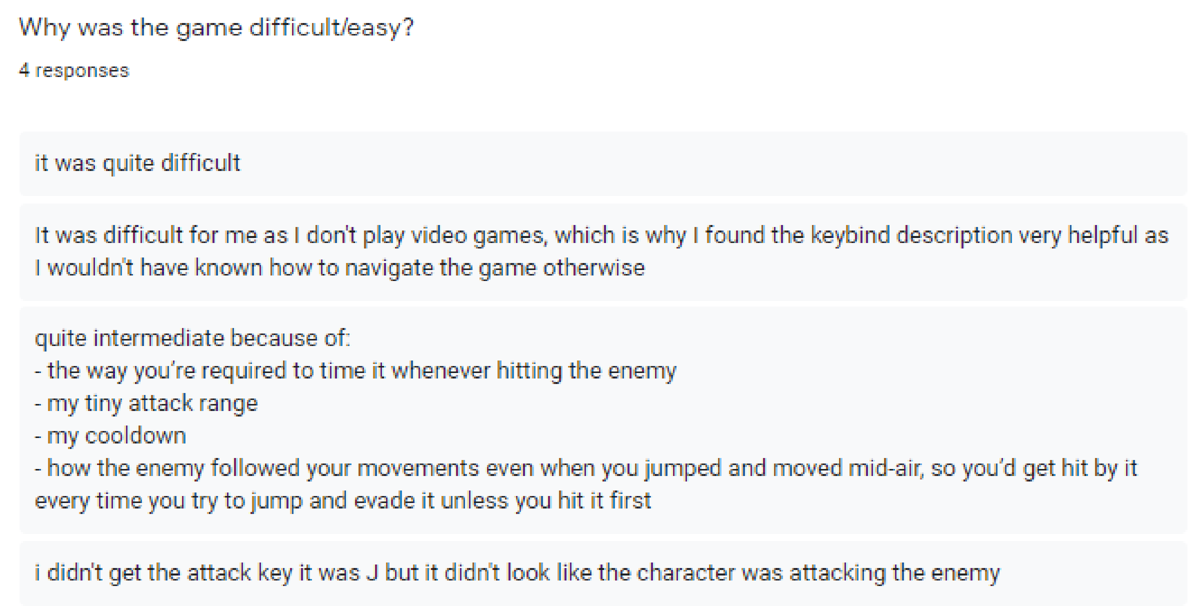


Figure 45: It's evident that the test group found the game to harder on average (1 represents easy, 10 represents very difficult)

These results indicate that the players had a harder time playing the game, which is the goal I was looking for as if the game was easy then there wouldn't be an aspect of competitiveness. In the future I could work on making the game easier through a difficulty setting which changes the speed of the enemy and the player animation swing time.



Why was the game difficult/easy?

4 responses

it was quite difficult

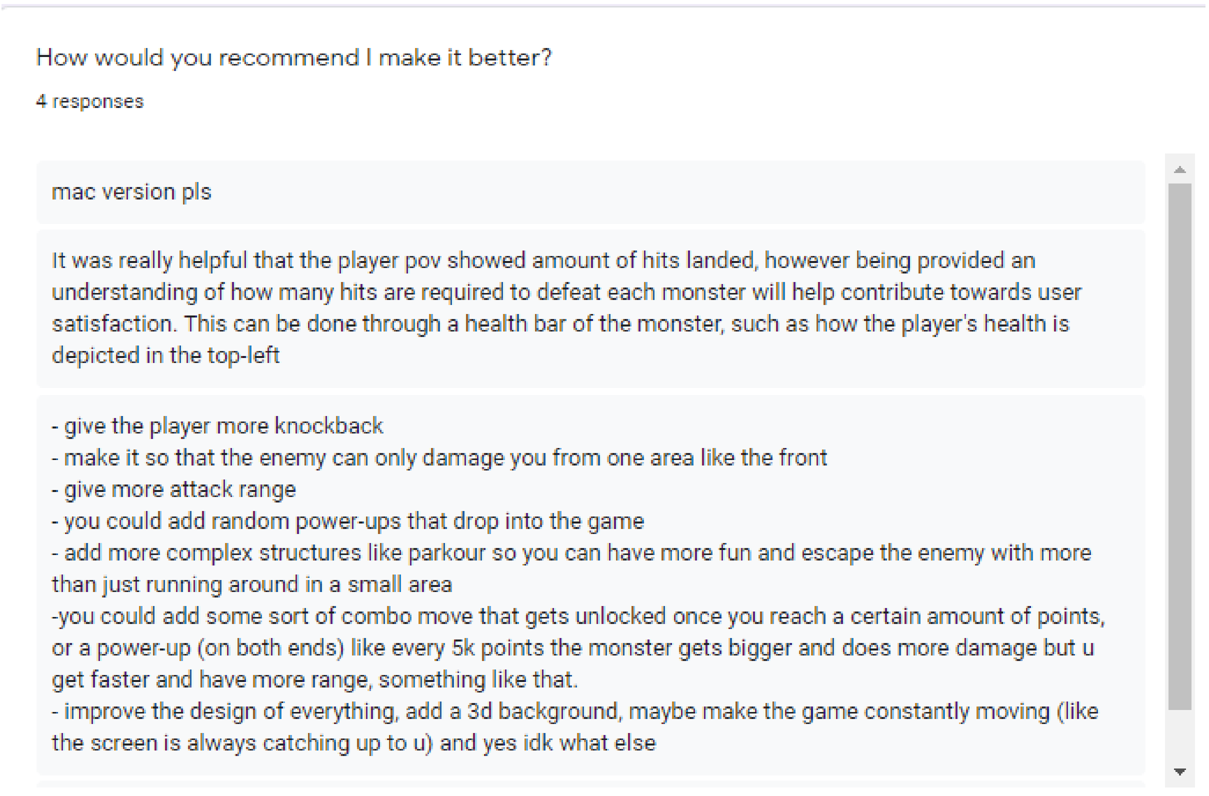
It was difficult for me as I don't play video games, which is why I found the keybind description very helpful as I wouldn't have known how to navigate the game otherwise

quite intermediate because of:
- the way you're required to time it whenever hitting the enemy
- my tiny attack range
- my cooldown
- how the enemy followed your movements even when you jumped and moved mid-air, so you'd get hit by it every time you try to jump and evade it unless you hit it first

i didn't get the attack key it was J but it didn't look like the character was attacking the enemy

From these comments I can tell that that my game difficulty was influenced by player swing times, attack range, cooldown length, enemy behaviour.

These are all variables which are in my capacity to change and therefore something I could change depending on the difficulty of the game



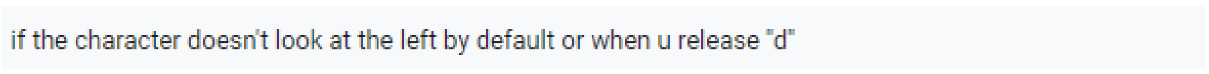
How would you recommend I make it better?

4 responses

mac version pls

It was really helpful that the player pov showed amount of hits landed, however being provided an understanding of how many hits are required to defeat each monster will help contribute towards user satisfaction. This can be done through a health bar of the monster, such as how the player's health is depicted in the top-left

- give the player more knockback
- make it so that the enemy can only damage you from one area like the front
- give more attack range
- you could add random power-ups that drop into the game
- add more complex structures like parkour so you can have more fun and escape the enemy with more than just running around in a small area
-you could add some sort of combo move that gets unlocked once you reach a certain amount of points, or a power-up (on both ends) like every 5k points the monster gets bigger and does more damage but u get faster and have more range, something like that.
- improve the design of everything, add a 3d background, maybe make the game constantly moving (like the screen is always catching up to u) and yes idk what else



if the character doesn't look at the left by default or when u release "d"

Ideas on improvement make it clear that my combat & visual system still needs a lot of work as well the game features. In the future I should work toward having an enemy health bar, powerups, improved map design, variation of attacks, background and combat/movement refinement.

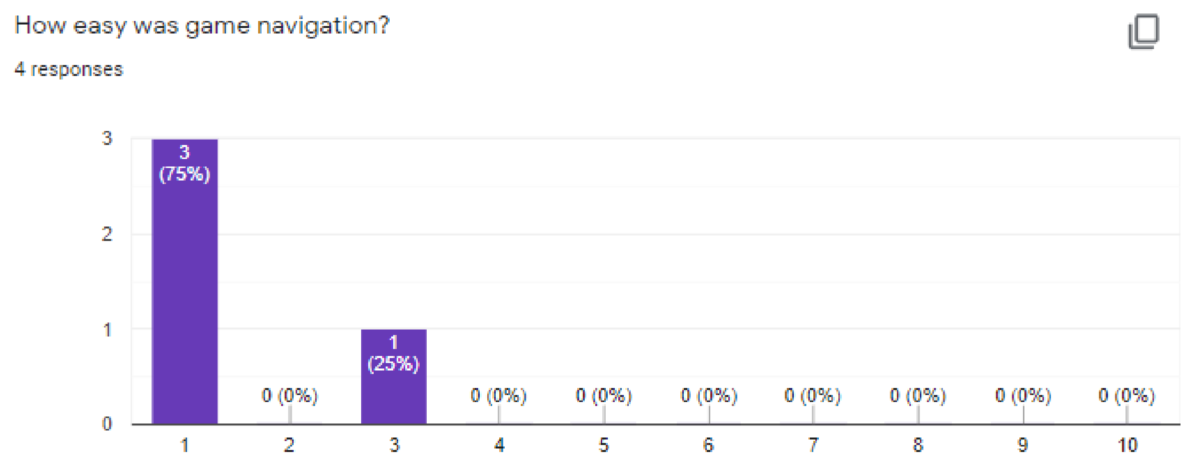
I should also consider that my original plan during development was to have 3 unique enemies that could be killed and would respawn, and that I had to make compromises and end up with one unkillable enemy. In the future I should re-evaluate my enemy plans and whether attempting my original plan will increase, decrease or have no impact on difficulty. Furthermore, I hope to raise the player skill ceiling by adding in wall jumps and dashes, this allows for increased manoeuvrability and will therefore make the game more fun/interactive and a bit easier.

I've also seen that there is a request for my solution to support Mac files, I should be able to do this using online resources that convert my project into .dmg or .app files (similarly to how I used py-to-exe for windows).

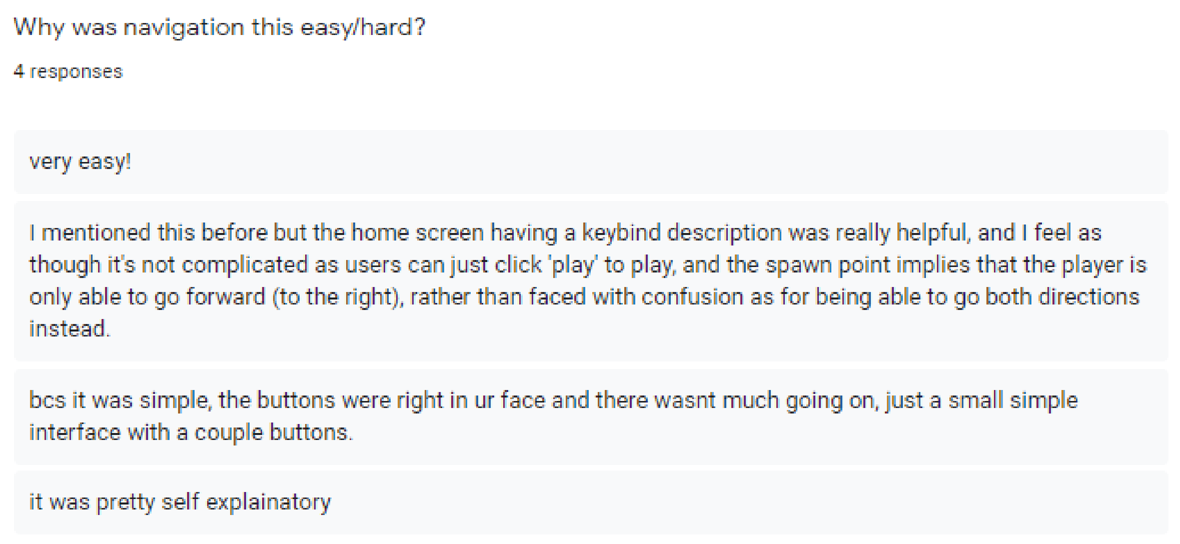
Overall, my game difficulty has a ways to go and should be one of my first priorities during game maintenance.

GAME NAVIGATION

This heading will cover how easily peers could navigate through playing states, controls and extras e.g., pausing.
Questions: How easy was game navigation? Could you start the game, view keybinds/scores, and pause easily? How would you make game navigation easier?



This is good as it shows that players are easily able to navigate my game. (1 represents easy, 10 represents extremely difficult)


Why was navigation this easy/hard?

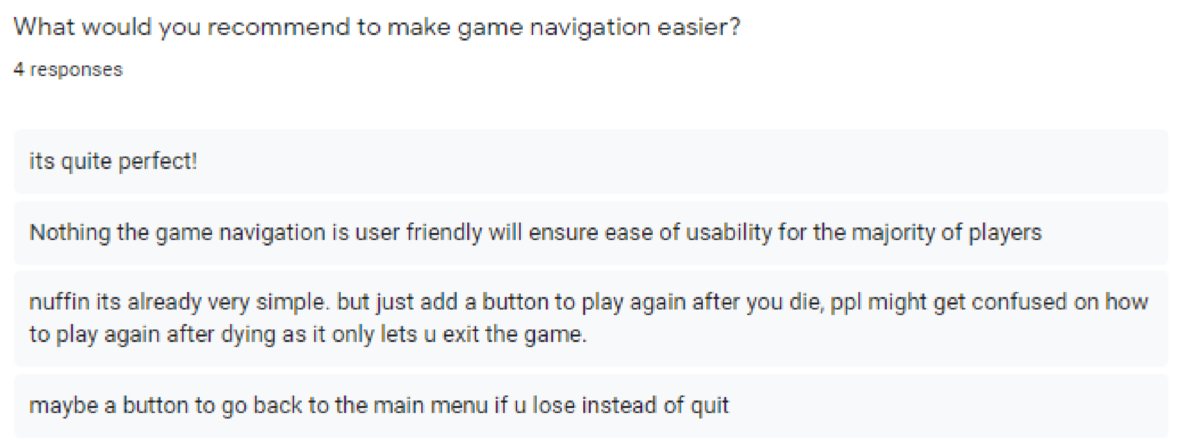
4 responses

very easy!

I mentioned this before but the home screen having a keybind description was really helpful, and I feel as though it's not complicated as users can just click 'play' to play, and the spawn point implies that the player is only able to go forward (to the right), rather than faced with confusion as for being able to go both directions instead.

bcs it was simple, the buttons were right in ur face and there wasnt much going on, just a small simple interface with a couple buttons.

it was pretty self explainatory


What would you recommend to make game navigation easier?

4 responses

its quite perfect!

Nothing the game navigation is user friendly will ensure ease of usability for the majority of players

nuffin its already very simple. but just add a button to play again after you die, ppl might get confused on how to play again after dying as it only lets u exit the game.

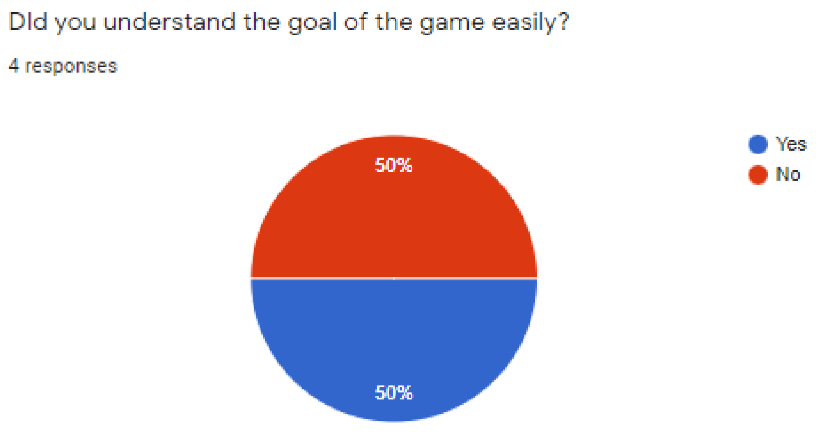
maybe a button to go back to the main menu if u lose instead of quit

The only recommendation for improving the navigation is one that allows the player to go straight from end phase to start phase, I discussed this during the development of my project as something I would focus on in the end as it wasn't too important and didn't hinder game enjoyment too much. However as this is raised as a user concern, I should prioritise this during future maintenance of the project.

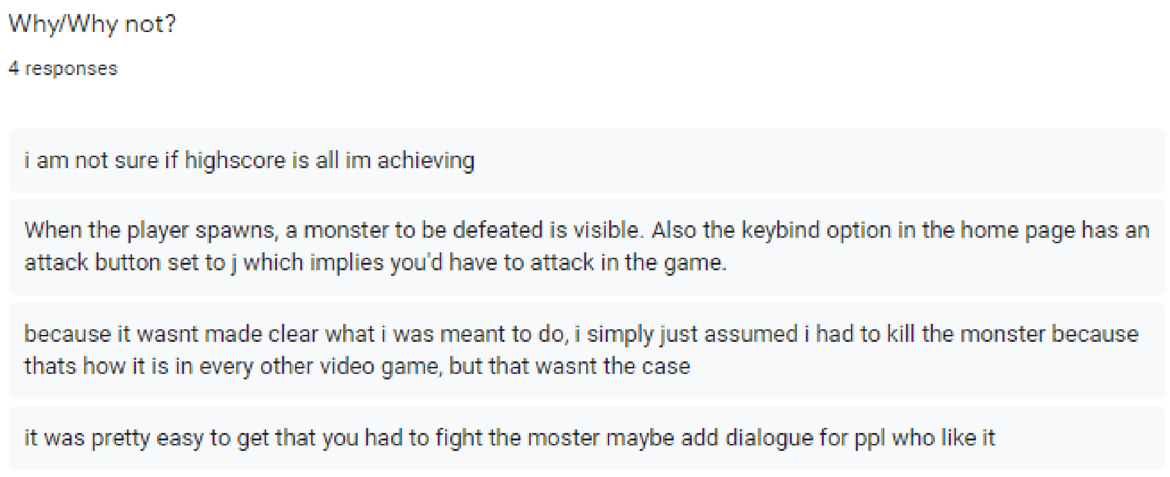
Overall, the game navigation has been coded to a good standard and will not need too much emphasis during maintenance.

GAME UNDERSTANDING

This heading covers how well peers understood the game, and the scenarios they would use the game in. Questions: Did you understand the game quickly? In what scenarios would you use this game (e.g., in your free time alone? With others? While waiting for something?)?



It's apparent that my game goal isn't clear to some, while it is to others.



Why/Why not?

4 responses

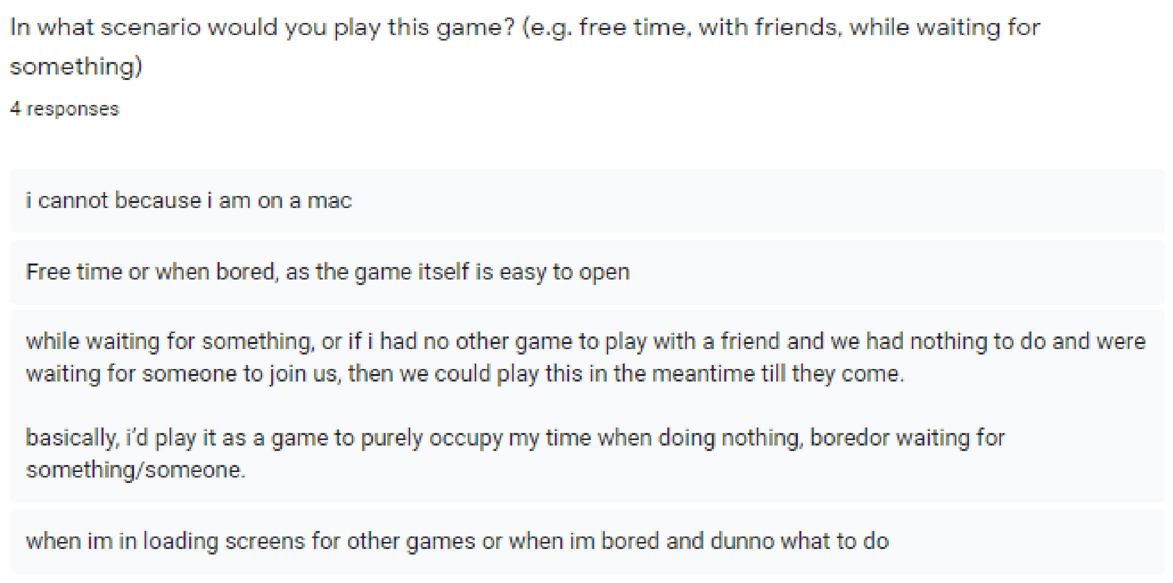
i am not sure if highscore is all im achieving

When the player spawns, a monster to be defeated is visible. Also the keybind option in the home page has an attack button set to j which implies you'd have to attack in the game.

because it wasnt made clear what i was meant to do, i simply just assumed i had to kill the monster because thats how it is in every other video game, but that wasnt the case

it was pretty easy to get that you had to fight the moster maybe add dialogue for ppl who like it

The results from this question align well with my Alpha peer feedback, I should work on a small question mark button which explains the concept of the game and how it is played. This is also a sign that fully developing enemy combat is necessary in order to make the game goal clearer (one response suggested dialogue which is an interesting concept to look into)


In what scenario would you play this game? (e.g. free time, with friends, while waiting for something)

4 responses

i cannot because i am on a mac

Free time or when bored, as the game itself is easy to open

while waiting for something, or if i had no other game to play with a friend and we had nothing to do and were waiting for someone to join us, then we could play this in the meantime till they come.

basically, i'd play it as a game to purely occupy my time when doing nothing, boredor waiting for something/someone.

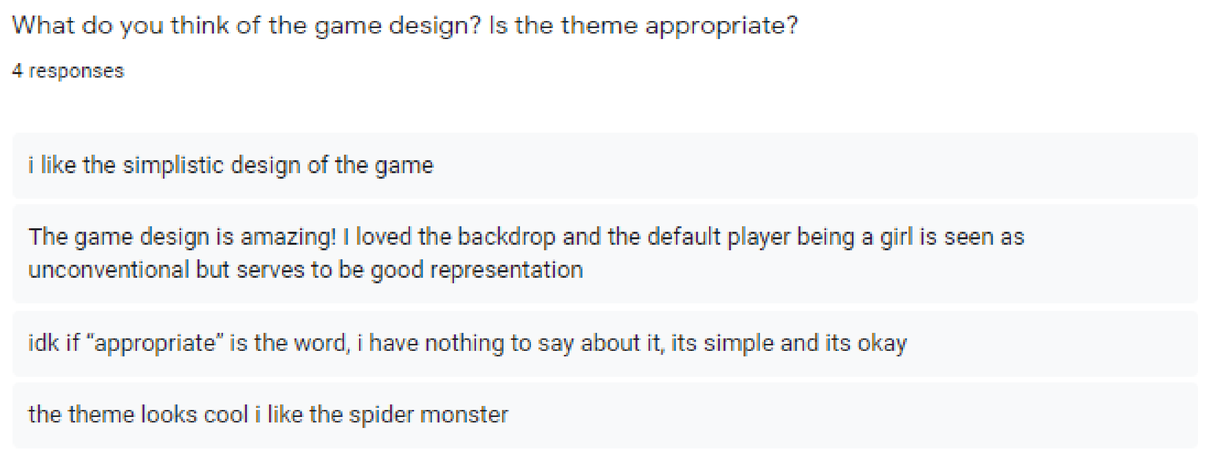
when im in loading screens for other games or when im bored and dunno what to do

These results prove that my game fulfils the requirement for which it was intended for, a game that is easy to pick up and play for those who are bored. That can run on any system (given that the operating system lets you run exes).

Overall, the understanding of the game is satisfactory, if there's anything I should work on in the future it's a tip in the main screen that explains the game concept and goal.

GAME DESIGN

This heading covers peers' opinions on the design and style of the game.
Questions: What do you think of the game design? Is the theme appropriate? What would you change?



What do you think of the game design? Is the theme appropriate?

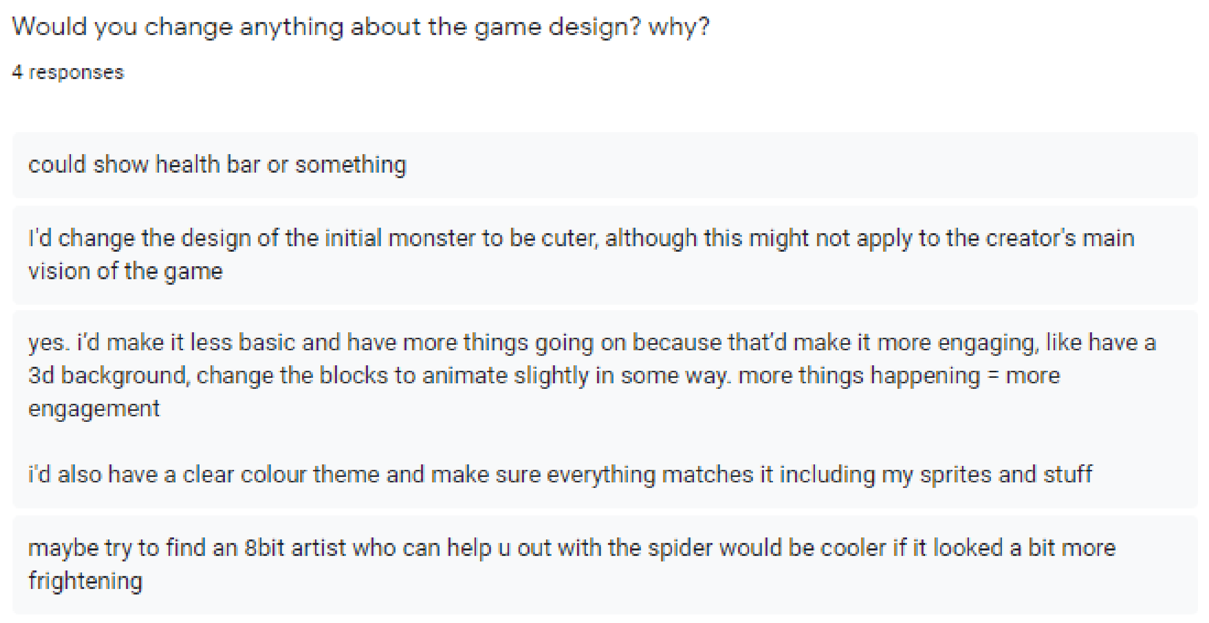
4 responses

i like the simplistic design of the game

The game design is amazing! I loved the backdrop and the default player being a girl is seen as unconventional but serves to be good representation

idk if "appropriate" is the word, i have nothing to say about it, its simple and its okay

the theme looks cool i like the spider monster



Would you change anything about the game design? why?

4 responses

could show health bar or something

I'd change the design of the initial monster to be cuter, although this might not apply to the creator's main vision of the game

yes. i'd make it less basic and have more things going on because that'd make it more engaging, like have a 3d background, change the blocks to animate slightly in some way. more things happening = more engagement

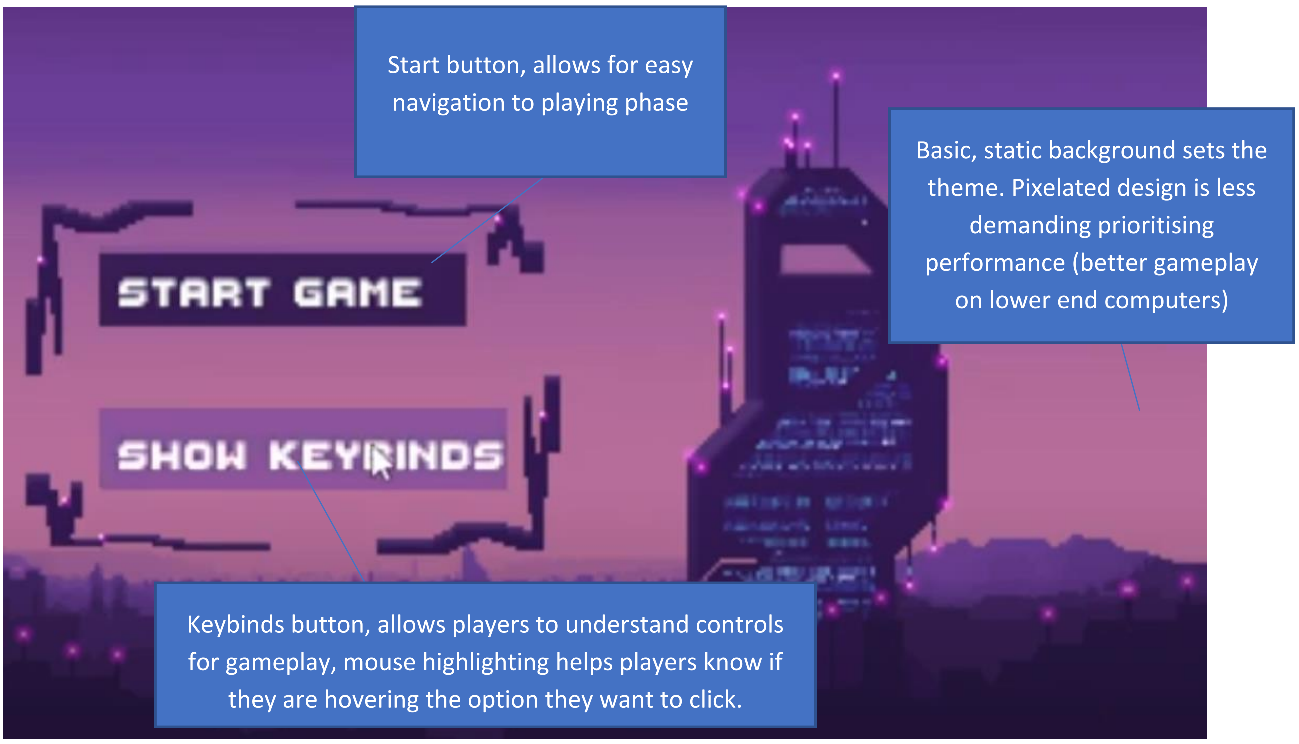
i'd also have a clear colour theme and make sure everything matches it including my sprites and stuff

maybe try to find an 8bit artist who can help u out with the spider would be cooler if it looked a bit more frightening

Overall feedback on game design is positive, in the future I should aim to make the game a little more appealing. Key things to work on are the enemy's design (which is still currently the test model as I had never the time to work on a final design), 3d background and extra animations to bring the game to life.

SOLUTION IN COMPARISON TO SUCCESS CRITERIA & USER REQUIREMENTS

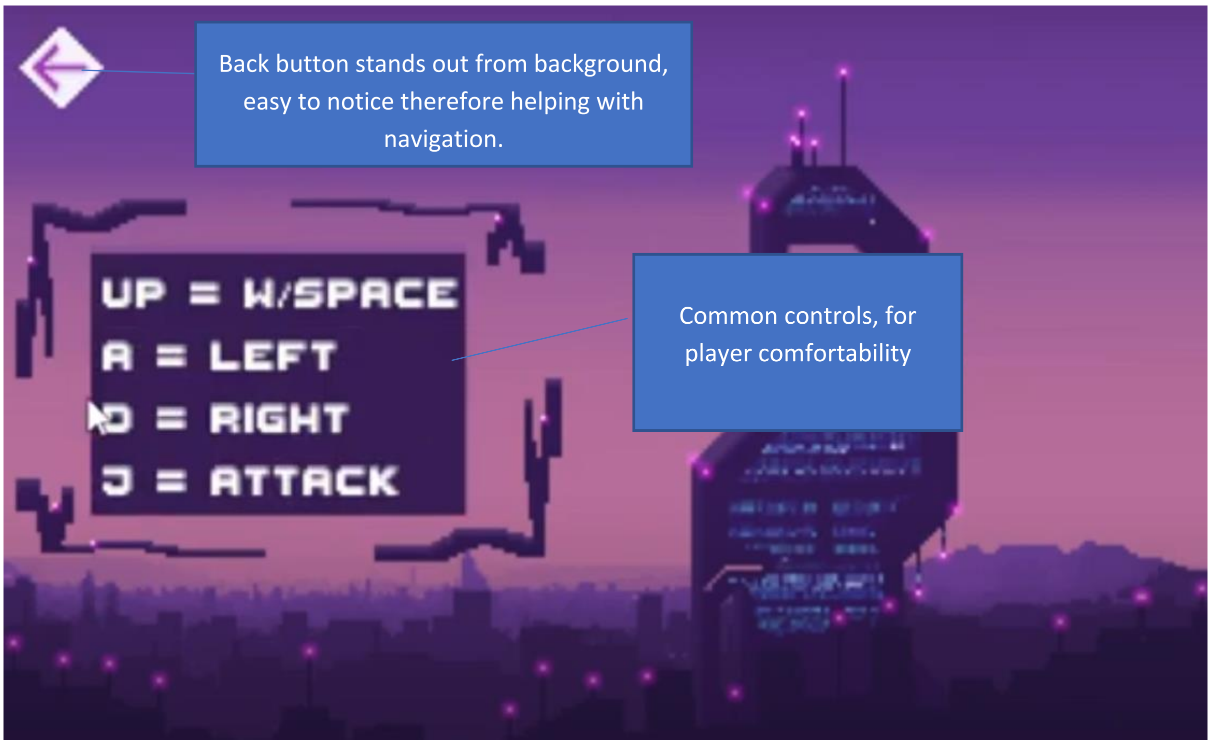
Here I will record a sample of me playing the game, and annotate my final solution and show how they meet user requirements and other information. (Note the score isn't visible here as it's a clean start meaning there are no past scores stored).



Start button, allows for easy navigation to playing phase

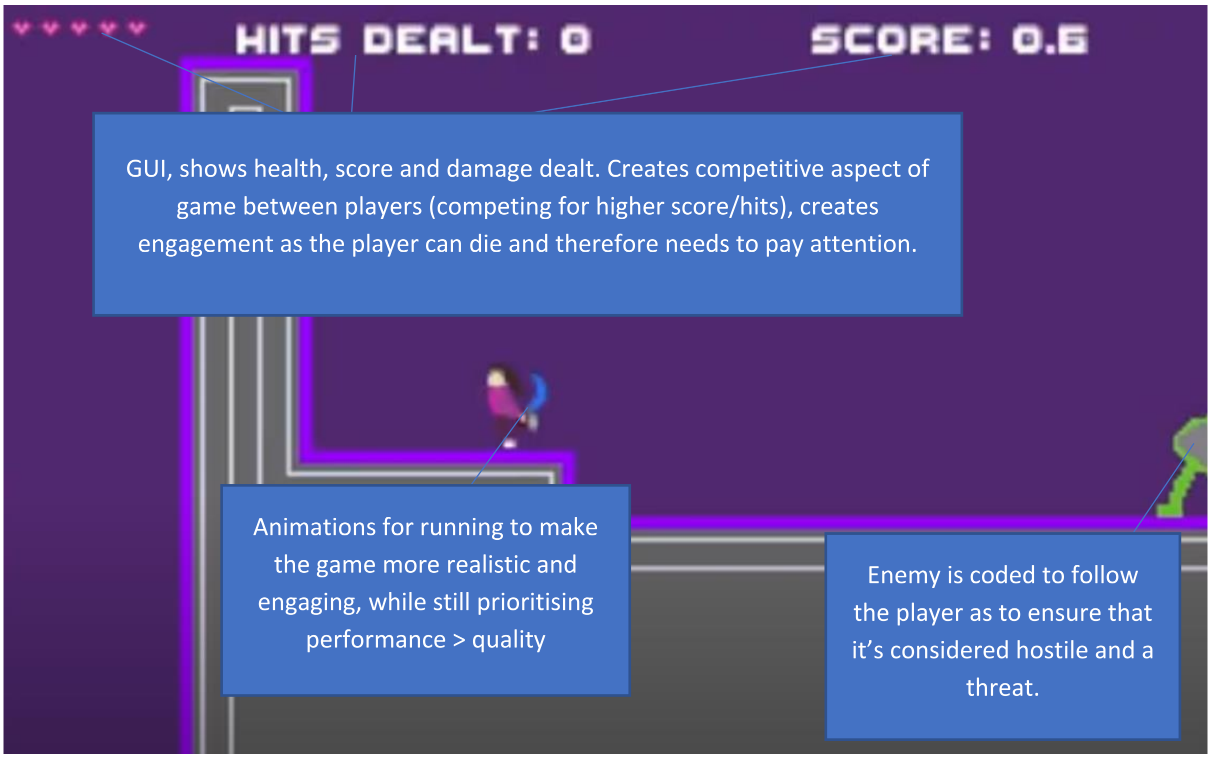
Basic, static background sets the theme. Pixelated design is less demanding prioritising performance (better gameplay on lower end computers)

Keybinds button, allows players to understand controls for gameplay, mouse highlighting helps players know if they are hovering the option they want to click.



Back button stands out from background, easy to notice therefore helping with navigation.

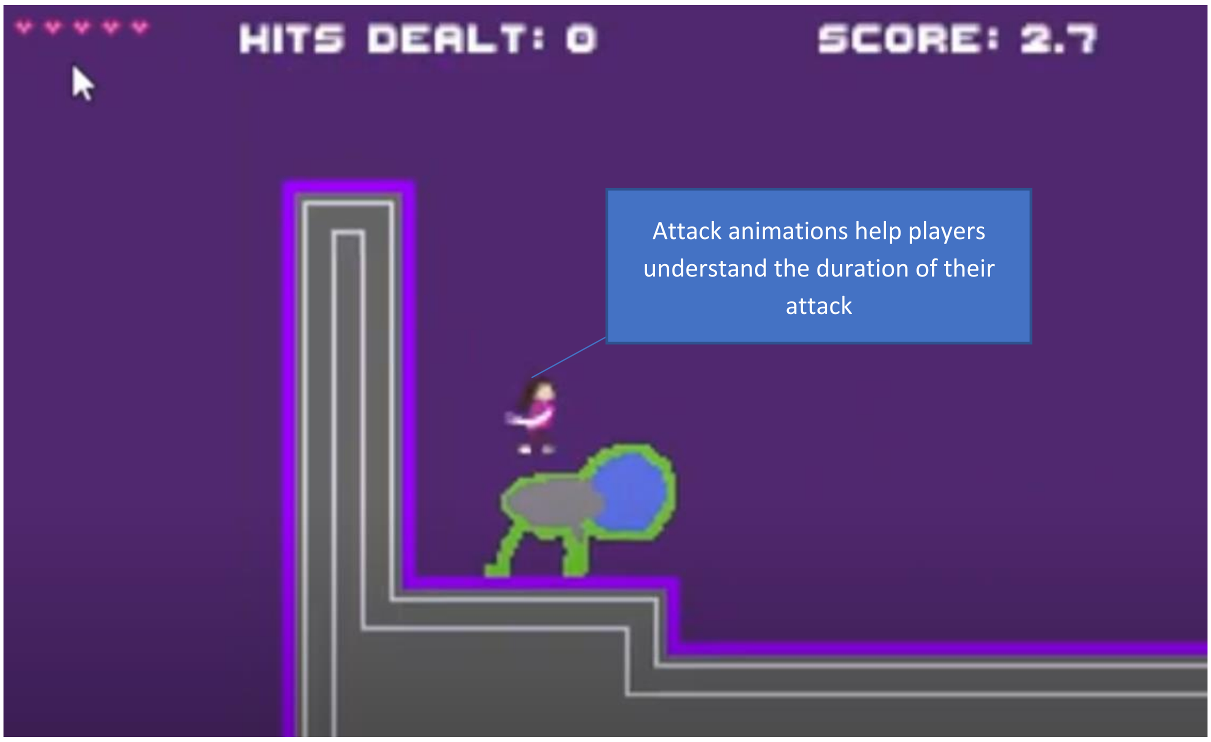
Common controls, for player comfortability



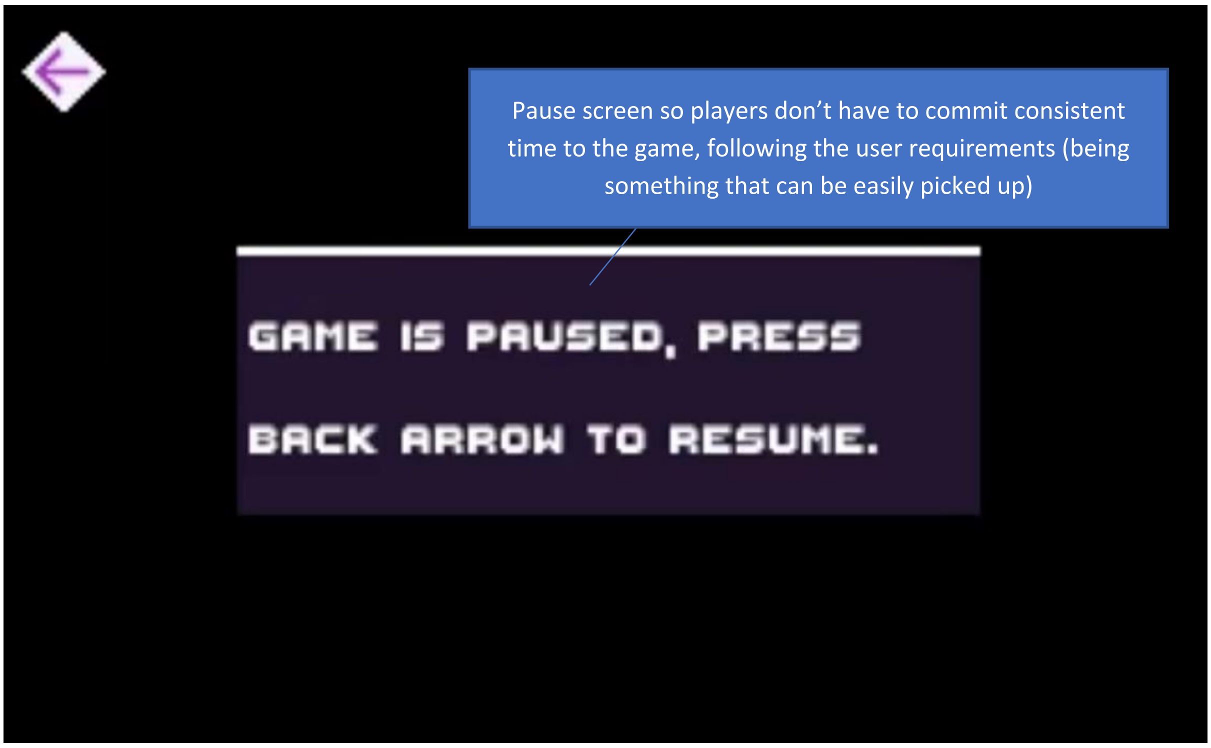
GUI, shows health, score and damage dealt. Creates competitive aspect of game between players (competing for higher score/hits), creates engagement as the player can die and therefore needs to pay attention.

Animations for running to make the game more realistic and engaging, while still prioritising performance > quality

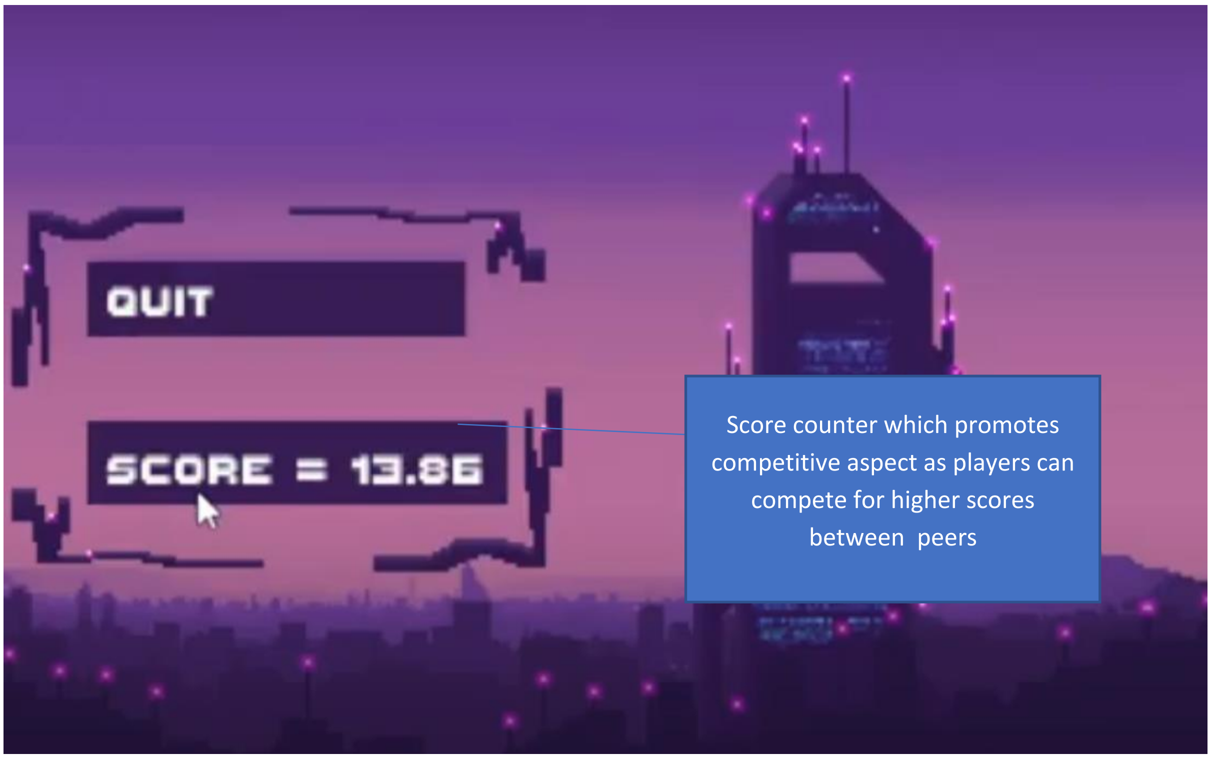
Enemy is coded to follow the player as to ensure that it's considered hostile and a threat.



Attack animations help players understand the duration of their attack


Pause screen so players don't have to commit consistent time to the game, following the user requirements (being something that can be easily picked up)

GAME IS PAUSED, PRESS BACK ARROW TO RESUME.



Score counter which promotes competitive aspect as players can compete for higher scores between peers

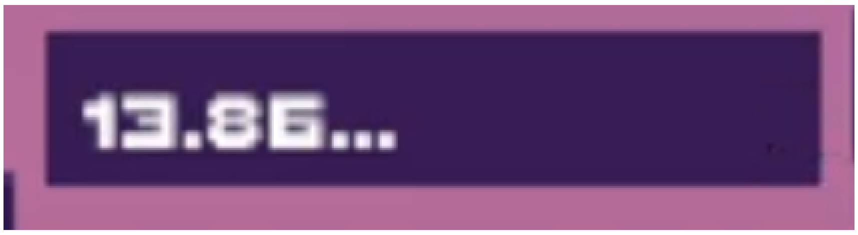


Figure 46: Clicking on the score button shows the full score (if players have that many points) to maintain competitive aspect among the best players too.

My solution meets all the basic user requirements, such that it is a not resource intensive and works with no connection. I've put life into all but some of the ideas created in my design phase (missing features include database storage, sound, enemy design etc.)

SUCCESS CRITERIA REVIEW

I will restate the success criteria here and discuss how effectively I have met the criteria, or what I could do to ensure I meet it in the future.

The essential features of the game are as such that it must be:

- *Runnable on practically any device*
    - *This means that the user should be able to run the game regardless of how weak their computer is, and that the game should run smoothly without any frame drops or bad quality gameplay*
        - *This is so that it meets the needs of the target market, which is that it can be run-on low-end devices that may not have the best hardware.*

I have met this requirement as the game can run on any pc, I know this as I have tested the game on weak laptops and some of my testing peers have also ran the game on weak computers. However it is entirely possible that there exists a computer that might run just well enough that it can function but not run my game, this is unfortunately a problem that cannot be solved as is an issue with memory/extremely outdated hardware.

- *Runnable without internet*
    - *This means that the user can run the game regardless of location, and won't need any sort of connection to the internet. (The user may require internet for the initial download of the game; however, this is a one-time requirement. There are ways this requirement can be circumvented e.g., Selling the game on a USB locally such that a user just needs to plug in the USB and run the game)*
        - *Similar to above, this is so that the game is available to all who have a computer, therefore meeting the needs of the target market*

My game meets this requirement as the entire exe can be run with no requirement from the internet, however internet will be required to download the game (unless it is sourced from someone else's storage device).

I will also discuss the limitations and how they impact my maintenance/future improvement.

The game has limitations such that it will not be able to be:

- *Run with online multiplayer compatibility.*
    - *This means that players will not be able to play/see each other in their games. And that there will be no way for players to connect with each other through my game.*
        - *This is because my main target market will likely not have access to internet, however I will consider making the game function online as a bonus option after the main criteria has been met.*

Although my game is currently in this state, I would like to attempt local multiplayer in the near future, to increase competitiveness and allow the game to bring people together.
Regarding online multiplayer, although I haven't a good idea how to attempt it yet. It is possible for me to include this such that those with internet can use online multiplayer, but the single player mode remains for

those who don't have internet. This may impact the size of the game and may indirectly cause lag if not handled efficiently, so I would only attempt this if the demand for an online multiplayer is large enough.

- *Graphically intense (realistic)*
    - *This means that the game will aim to avoid straining the GPU. The game will not look flashy or detailed as a result of this (although a level of basic design would still be necessary in order to make the game enjoyable)*
        - *This is because my target market won't have the strongest computing power.*

My game is not graphically intense due to the nature of how its been designed, however I could make the game look prettier by using more detailed textures, this wouldn't affect the performance of my game as I can compress these images (using lossless compression) and the script would be loading images anyway (I may have misunderstood the concepts of 3D rendering and 2D rendering at the time because now I understand how 3d rendering will strain the GPU in a different way than 2D rendering). Because of this I may work on a retexture of the game. Overall, my game doesn't strain the GPU as it is extremely small and changing the textures of the game to look better shouldn't have a significant impact on that.

- *Run without a keyboard/mouse*
    - *This means the game will depend on inputs from a keyboard or/and a mouse*
        - *This is because the fundamental movement requires input from keyboard and menu selection may require input from mouse (I may consider making the game entirely* keyboard as this feature can reach even more of my target market)

My game demands input from a keyboard and mouse in order to run, this is a core concept of anything traditional computer game. However, it is possible for me to redo the start/end screen such that it can be navigated using a keyboard, which would make my game only require a keyboard to run. However, this isn't something I feel I should work on as most computers will be used with a mouse and I am not appealing to anyone within my target audience by making the game not require a mouse.
When I mention mouse here I am also referring to the use of a trackpad, if the user is on a laptop.

The checkable/measurable success criteria of my game is:

- A 2 dimensional fighting game
    - Can be checked by simply playing the game

This criteria has been met, as visible in the screenshots above.

- Carries on endlessly (unless the player dies)
    - Can be checked by trying to last as long as possible (or editing code to make player invincible) and seeing if the game keeps going

This criteria has been met, as I am able to continuously play the game for extended amounts of time with no crashing (See above video for a short example)

- Has a score counter
    - Can be checked in the end phase, score should be visible if a score counter was in place.
- Has a player that can fight enemies
    - Can be checked by playing the main phase and attempting to interact near enemies (if they exist)

MAINTENANCE

The code has been annotated in order to allow future developers/myself to come back to the script and improve/fix any concepts. I have also used GitHub to manage and update my project meaning that it is possible to backtrack/view branches in order for a more complete understanding of how the game was developed to its current state. Because of this my program is easier to use for development during future maintenance. I could improve on the state of the program by moving more code blocks into functions for isolated testing and to support a wider range of programmers.

My solution is one that is not too adaptable, the concept of the game is relatively simple and there is not much I can do maintain it's relevance in a possible market (apart from adding abilities in rotation, extra movement etc.), because of this, the best way to reintroduce this solution (given that all current problems are fixed and the game is in a complete state) would be to reinvent the game as a sequel.

CURRENT MAINTENANCE

There is currently no maintenance/updates system built in the game. Practically every professionally distributed game has maintenance features, and it is wise for me to follow through, ideally, I would inform players of recent updates through social media. If I ever develop some sort of method for the game to connect to a central server then I could place updates there and notify every user playing the game (with an internet connection) to update. However, the issue with this is that my players will likely not have a consistent internet connection, the best I can do to account for this is to establish a monthly/bi-monthly update text in the game, making all users aware that there will be a message released every month stating whether there is an update for the game and what it includes.

As my project is extremely new, it would be worth considering rigorous testing for the game to ensure that all current physics and other features in the game cannot be exploited, and then performing new checks at every update. This would ensure that the foundation of my game is strong, and so any maintenance doesn't have unintended effects on the core functionality of my game.

FUTURE DEVELOPMENT

There are many things that people have suggested I add to the game, and features that I have had plans to work on as well, which have been labelled throughout the Evaluation Testing as well as Solution Usability. Given that as I work on these features and the game gains popularity, I will also receive requests for more features, it's plausible to say I could maintain the game over an extended period of time. However, there will undoubtedly be limitations that may stop me from accomplishing certain goals (e.g., online server hosting and detailed 3d graphics will go against the idea of my game despite it being recommended by peers, meaning that I'm limited by my own requirements unless I split the game into lower end and higher end versions/settings).

Some future development plans based on user feedback are as follows:

- Refined combat system
    - Through my survey I have found that many users found my gameplay rather difficult, I should look to refine my combat by adding more manoeuvrability such as invincibility dashes, wall jumps and charged attacks
        - This is high priority in my overall development plan as it fits in line with my original plan during my design phase, and would be one of the first things I work on.
- Powerups and complex parkour structures

  - Similarly, other peers have stated that my combat system could be improved by adding powerups and parkour structures which would make my main phase more random and therefore more interesting.
    - This is medium priority as it is more of a bonus feature, and will take a lot longer to implement as the map will need to be entirely redone and enemy behaviour will need to be reconfigured so that it is smart enough to also do parkour (to avoid players farming points forever by staying on an elevated platform)
- Add a "Restart" button
  - I originally had a restart button, however replaced it for quit since the functionality was buggy. Since users have shown an interest in having a restart button I should revisit this module and try an implement this.
    - This is low-medium priority as although it's requested, it doesn't significantly hinder game performance as players can still restart except it'll take them a few seconds longer.

Further future development plans have been included in my limitations section below, including how I would attempt to implement the said features.

LIMITATIONS

There are current limitations in my game that, if removed, can greatly improve the user experience.

- Adding sounds to the game
  - This makes the game more immersive and provides audio cues to help with synchronising attacks
    - This could be coded with my current knowledge using the respective python modules, and shouldn't take too long to implement; however, it would need re-iterative development and feedback from peers which can be time consuming. I consider it best to do this alongside working on another implementation.
- Multiple enemies (when one dies another is spawned)
  - This creates variation in my game so that players are less likely to get bored from fighting the same enemy with the same behaviour pattern over and over.
    - This would be done by copying the way I had added my first enemy (or even better creating a class and placing all the enemy variables under the class initialisation).
- Boss enemies
  - This gives players something to work towards when playing the game, and creates variation in enemy styles and therefore more to learn/master about the game
    - This would be done in an analogous manner to above, however with a subclass and a decreased spawn rate, unique attack patterns as well as physical properties (size, speed, strength etc.)
- Local two player (versus)
  - This allows players to go against each other if they are on the same computer, further enforcing a competitive environment hence further sustaining the player count.
    - This would be done by creating a second player, setting new variables for that player and setting up its movement keys differently. (It is worth noting that I had attempted this during my beta Cycle 2 however it caused the entire game to stop working, which I suspect is because of problems with character blitting which therefore crashed the overall display and stopped the game from working) I would need to do a little bit of research on how others had attempted this before tackling the problem myself

There are many other limitations such as 3d graphics, multiplayer online etc. However, these do not fit my user requirements nor my end goal for this game. If I were to add such features, it would most likely serve under a duplicate game or with optional settings.

PERSONAL REFLECTION

Coming to the end of my project, it's important to review how well the overall process went for me as an individual developer/documenter.

I feel that my analysis phase was quite smooth and I had adequately gotten data through surveys and reviewing similar solutions. However I wish I had spent time looking at failed solutions, and 2d games with the same vision that were infamous, this would've helped me understand what my game should avoid and how I could add features that would keep my target audience satisfied.

My design was done to a standard I was satisfied with, and I felt my charts/diagrams made it clear to me what the problem was, and which modules I should attempt to work on first, as well as how I could utilise computational thinking to make the workload easier. In hindsight, this phase could've been better if I had taken the time to analyse each phase in more detail to help deal with more complex problems in development e.g., I should've made a diagram for the playing phase in more detail, highlighting the enemy entities, and hopefully realising that an efficient solution for multiple enemies would involve classes and creating enemies with instantiation.

My development phase was rather rough, I had spent a lot more time on it than I expected to and came across many lethargic days. In the future I would look at the possible technologies that could potentially better my solution before coding e.g., looking into how AI would make my enemy movement more unique, or how infinite randomised terrain generation would hopefully prevent my target market getting bored from the game as easily.

FINAL GAME FILES + CODE

My project was maintained on GitHub.. The 'Game' folder holds the python files and assets, all of which I had personally developed during my execution phase. The "Executable Game" compressed package is the file which I had sent to peers, as it allows the game to run on windows machines that do not have python, or any libraries installed (I had creating this .rar using a library to py-to-exe)

In order to run the EXE, the Main.exe file needs to be run, this is located "Executable Game/Main.exe".

BIBLIOGRAPHY

Google. (2021, October). *Google*. Retrieved from Google Dinosaur Game: https://www.google.com/search?q=google+dinosaur+game&rlz=1C1VDKB_en-GBGB989GB989&oq=google+dino&aqs=chrome.0.69i59j46i20i263i433i512j69i57j0i67l2j0i20i263i512j0i67l3j0i512.2012j0j1&sourceid=chrome&ie=UTF-8

IGN. (2021, October). *IGN*. Retrieved from Google images: https://www.google.com/search?q=hollow+knight+combat&tbm=isch&ved=2ahUKEwiAgeKKwdj3AhWugc4BHaJNBuMQ2-cCegQIABAA&oq=hollow+knight+combat&gs_lcp=CgNpbWcQAzIFCAAQgAQyBggAEAgQHjIECAAQGDIECAAQGDIECAAQGDoECCMQJzoKCAAQsQMQgwEQQzoECAAQQzoHCAAQsQMQQzoECAAQHlCfAlji

izmamonke. (2022, March). *python-forum*. Retrieved from Gravity doesn't works, and my character just moves right and ...: https://python-forum.io/thread-34529.html

Tutorials, C. (2021, February). *Youtube*. Retrieved from Paint a pixel art city / wallpaper / game background in photopea (free online photoshop) 8-bit style: https://www.youtube.com/watch?v=7KDZ1GjJlsY&t=1472s&ab_channel=CGITutorials

All external sources are hyperlinked to their source.