a lEVEL

COMPUTER SCIENCE PROJECT

Computer Game [Worlds Collide]

Contents

[Analysis 3](#_Toc121141744)

[The Problem 3](#_Toc121141745)

[Why the program is computational? 3](#_Toc121141746)

[Stakeholder 7](#_Toc121141747)

[Research 8](#_Toc121141748)

[Game Reference/Competition 8](#_Toc121141749)

[Why combat orientated? 9](#_Toc121141750)

[Final Conclusion 12](#_Toc121141751)

[Question and Answers 12](#_Toc121141752)

[Questions 12](#_Toc121141753)

[Features 15](#_Toc121141754)

[Limitations 17](#_Toc121141755)

[Requirements 17](#_Toc121141756)

[Hardware Requirements 17](#_Toc121141757)

[Software Requirements 18](#_Toc121141758)

[Success Criteria 19](#_Toc121141759)

[Design 22](#_Toc121141760)

[Solution to make it computational 22](#_Toc121141761)

[Top-Down Design 22](#_Toc121141762)

[Development methodology 36](#_Toc121141763)

[System Lifecycle 36](#_Toc121141764)

[User Interface 37](#_Toc121141765)

[Main Menu 37](#_Toc121141766)

[Controls 38](#_Toc121141767)

[Pause Menu 39](#_Toc121141768)

[Game over 40](#_Toc121141769)

[In-Game Screen 41](#_Toc121141770)

[Stakeholder feedback [User interface] 42](#_Toc121141771)

[Final User Interface Designs 45](#_Toc121141772)

[[Main Menu] 45](#_Toc121141773)

[[Controls] 46](#_Toc121141774)

[[In-game Screen] 47](#_Toc121141775)

[Usability Features 47](#_Toc121141776)

[Maintainability 48](#_Toc121141777)

[Coding conventions 48](#_Toc121141778)

[Iteration Plans 49](#_Toc121141779)

[Development 50](#_Toc121141780)

[Class Diagram 50](#_Toc121141781)

[Software Testing Methodology 50](#_Toc121141782)

[Black Box testing 50](#_Toc121141783)

[White Box testing 51](#_Toc121141784)

[Alpha Testing 51](#_Toc121141785)

[Beta Testing 51](#_Toc121141786)

[Iteration 1: Framework 51](#_Toc121141787)

[Aim 51](#_Toc121141788)

[Key variables 52](#_Toc121141789)

[Pseudocode 53](#_Toc121141790)

[Test plan table 59](#_Toc121141791)

[Development Diary 59](#_Toc121141792)

[Result of testing plan 71](#_Toc121141793)

[Stakeholder Feedbacks 73](#_Toc121141794)

[Iteration 2: Sprites, animations, movement, camera 74](#_Toc121141795)

[Aim 74](#_Toc121141796)

[Key variables 74](#_Toc121141797)

[Pseudocode 75](#_Toc121141798)

[Test plan table 77](#_Toc121141799)

[Development Diary 77](#_Toc121141800)

[Result of testing plan 88](#_Toc121141801)

[Stakeholder Feedbacks 89](#_Toc121141802)

[Iteration 3: Pause and Quit menus and Enemy AI 90](#_Toc121141803)

[Aim 90](#_Toc121141804)

[Key variables 90](#_Toc121141805)

[Pseudocode 90](#_Toc121141806)

[Test plan table 90](#_Toc121141807)

[Development Diary 90](#_Toc121141808)

[Result of testing plan 90](#_Toc121141809)

[Stakeholder Feedbacks 90](#_Toc121141810)

# Analysis

## The Problem

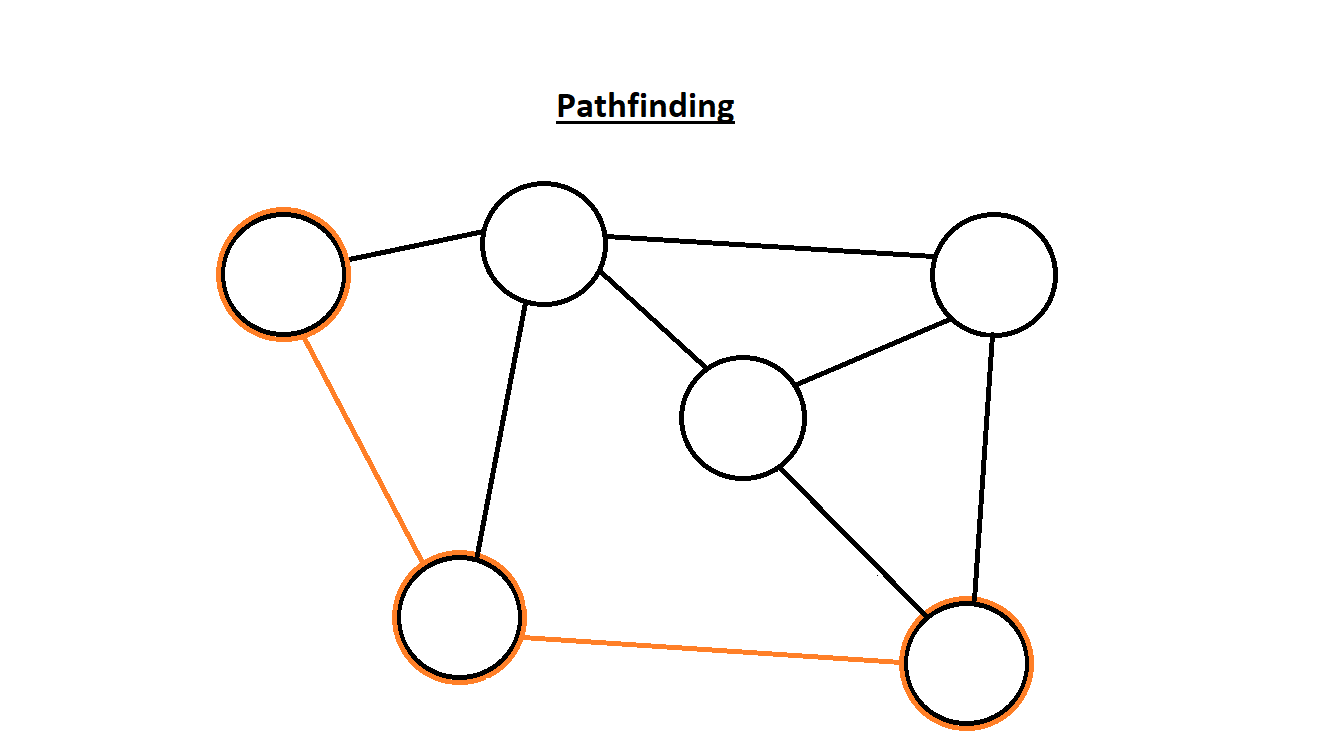
The immersion of games. The concept is given to those who are immersed in games believing that they are within the game itself. When people play games, it allows them to leave the real world. It allows people to forget their struggles and problems. In addition, games are a fantastic way to rethink and take different approaches to real-world struggles. Why? Well because games allow people to take breaks from using their brain; it reduces stress by creating a flow of sensation and satisfaction – similar to meditation. When people are stressed, they tend to take a different perspective of the situation by changing their pace of workflow which is the main reason why I am creating a game. One way to combat stress is to use combat itself. Creating a game that is fast pace and requires the player to focus, allows the concentration to be shifted away from the stressful stimuli. But why a video game? Well, video games are engaging. Unlike taking a stroll in the park, video games allow people to completely forget about stress. Taking a stroll in the park, even though for some it relieves the pressure, for many since there is not anything for the brain to be distracted with, the brain signals out the stimuli, causing the person to be possessed with the stress. The program is going to be a 2-dimensional (2D) ‘Hack and Slash’. This is where combat is emphasized with melee-based weapons. The aim of the game is for the player to defeat waves of enemies, and reach a high score. Therefore, the main purpose of the action-packed and fast pace Hack and Slash will be for the player to be immersed in the realms of the game to mitigate stress.

## Why the program is computational?

Games heavily rely on computational methods. Without computational methods, the idea of creating a game will be so overly complex, that games, in general, will not have a place in the market, let alone exist. Therefore, for many games, different methods are required in order to create a successful game.

**Decomposition (thinking procedurally):**

One way in which my Hack and Slash game utilises these methods is the use of decomposition. By breaking down the functionality, I am able to make sub-functions for the sprite, actions, camera, and many other sub-layers that conclude the foundation of the final game. One main sub function is pathfinding/pathing. This is the concept of entities finding the shortest route to the final destination. For that reason, the enemies are going to be using pathing to track the position of the player.



*(Pathing)*

The illustration [source: Pathfinding] tells us how pathing works. We know that pathing allows objects to follow the shortest route to their destination. Therefore, the illustration shows that the orange outlines are the fastest route to the destination compared to the other possible routes. However, the diagram is abstracted since there could be obstacles blocking the entities from taking different routes. With this in mind, I am going to be implementing a similar design for the enemy pathing.

|  |
| --- |
| *(Top-down design: Key Functions)* |

Decomposition generally makes solving and understanding the functions much easier. This is because it helps the programmer understand the situation and the steps required to achieve the main function. When decomposing, the top-down design is the most suitable since the stages are in a hierarchal format. The top gives an overview of the main functions while the bottom is the sub-functions that are the building blocks to create the main function. The key problems to be solved are stated through visualisation on my top-down design [source: ‘Key Functions’]. Note that the diagram is a simplified and abstracted example, the more comprehensive top-down design diagrams will be stated in the analysis section.

* Controls:
  + The main control consists of using the keyboard. With the main keys being ‘W’/’Space’ (jumping), ‘A’ (moving left) and ‘D’ (moving right). With these keys, the player has the ability to move the character in any position they wish. This makes the game interactive by allowing players to be active rather than being static.
  + The use of the mouse is also crucial. Without the mouse, the player is not able to attack, resulting in the game idea being useless. This is because the main point of the game is for combat.
  + The help/instruction is an option for the player to see what controls they need to use. If the player enters the game without the knowledge of what controls to use, then the game would not be played.
* Graphics:
  + For the design element, I want to create the art aesthetically pleasing. This is because, when the visuals look beautiful it satisfies our senses and does not strain our eyes. The visuals are an important criterion to hit. Since my program is to help alleviate stress, having the visual look suffocating does not relax the mind. This will result in the problem surrounding stress not to dissipate, but elevate.
* Options:
  + Without different options, it will make the player jump into the game without any warning. Having them start the game straight away will cause some confusion since the menu screen allows the user to control the settings. The settings are important as permission is given to the player to change the quality of the setting. This may be due to their low-end computer. But due to the time limit, the implementation of different graphical settings will not be added.

Furthermore, having the play button rather than kick-starting the game is important as it will allow the user to load the game at their own pace. In addition, the controls button helps players understand the main keys to be used when playing the game. Without knowing the main controls, players will be clueless about what to do. Also, the quit button will close the game. If the user decides to play the game in full screen, then the close ‘x’ button will not show. This may leave some inexperienced players confused on how to quit the game. Finally, the choice of pause menu allows the user to pause the game whenever they want.

* Sound:
  + Sound is important for stress. The experience of calm sounds helps ease the player's stress. Music that sounds appealing to the ear can cause many health benefits. When we are healthy, the amount of stress accumulated (if stress had an arbitrary number) falls to near 0. The implementation of satisfying music can help players mollify their stress. This is because of the power music does to our minds. Depending on the tempo of the sound, it can change our pulses, which mends our minds together.

**General Methods (e.g., Calculations, input/output):**

Another computational method that is going to be used is calculations. Without calculations, players will be incapable of doing any basic movement. Calculations are going to be a vital implementation of the game, since it is heavily relied upon calculating the damage done and taken. The use of calculations will make developing the health easier, the score users receive and the movement speed.

Furthermore, the use of input recognition is important. Without the use of inputs, the game will not operate. Inputs are crucial for my game since it is needed for certain actions. For example, the use of the mouse click is vital since it is used to attack enemies. If the left click had no input, there will not be any responds and the game will be a still image, where the user can only watch while the enemies deplete their life. Also, the use of the keyboard allows the user to move. Without these input responses, the user is not able to move around the game.

**Visualisation:**

The use of visualisation is important. Since it is easier to understand the aim of the problem through illustrations. It helps break down and understand complex instructions in a more simplistic format. This is because the use of visual aspects aids people to solve information quicker and more efficiently rather than reading a passage. It has been scientifically proven, that many people who see problems through images, graphs, flow charts, etc solve dilemmas much quicker. However, the use of visualisation is not only to solve problems. Another technique used is analysing and summarising concepts. This is a key feature of visualisation since it can outline unintelligible problems straightforwardly. The way I am going to be implementing the visualisation aspect is by graphically representing large chunks of information. This can be seen in the design section where I break down the main functionality into sub-functions to assist me in understanding the aim of the problem and the required steps to follow to solve the issues.

|  |
| --- |
| *(Design Concept)* |

The [source: ‘Design Concept’] diagram is an example of how users can visualise the layout of the game. Without saying a word, users can understand what is going on and what the game is going to look like. Here, we can see that the health bar for the player is on the top left. The player is represented as a yellow star, while the enemies are pink bolts. There is also going to be a ground layer and a backdrop. With only showing images, people’s brains decode concepts easier and faster than if it was text based.

**Backtracking:**

Backtracking is the use of multiple paths. The use of algorithms to solve problems does not always succeed. Therefore, when the program does not run, we can return to our previous path in order to try a different approach. For my program, the use of backtracking is going to be crucial. This is because, even though I know the different functions required to create the game, I do not know how to construct the code together without a few trials and errors. Therefore, the use of backtracking allows me to return to my previous path of code if the newly produced code does not work.

**Abstraction:**

On top of the other computational methods, the use of abstractions is going to be essential for the game. Without separating ideas from reality, the game will get overly complex. The main reason is that having a complex concept of the game will cause; firstly, the programmer unable to understand what they are coding and secondly, the player unable to understand the concept of the game. Without a good vision of what the game is supposed to do in its most basic form, just the thought of playing it will become useless. By ignoring the unimportant detail and focusing on the most important parts, it will allow me to scope out the functionality and key sections of the game. For that reason, I am going to abstract the ideas of my game from reality. One way that I am going to use abstraction is through games. By allowing the player to immerse themselves within a combat game, rather than in real-life combat, allows physical punches to change into virtual forms. Resulting in abstraction being used. Additionally, I want the player to gain experience in other ‘worldly’ situations. Even though neither I nor them have not entered other realms, in games, I do not have to think about realism. Just leaving it up to the game master helps you go through a new adventure and opens new chapters in your life. This all links to my game as I want everyone who plays to be immersed in my game. I want them to be submerged in my game so their stresses can be relieved.

Furthermore, the use of abstraction is also going to be used for the design element. By abstracting the design, we essentially make the game out of basic shapes. This is powerful since it helps us clearly understand the variables that needs to be added and the visuals of the project. If we decided to design the visuals without any abstractions, then time is could be wasted if changes are required. Additionally, without a clear thought of the design, then designers will be confused about what the visuals are supposed to look like. At the [source: ‘Design Concept’] diagram, I have broken down how the game is going to look in its simplest form and only added the essential features the game is going to have. Once those key features are added, then I am going to improve on them and make them look pleasing. In addition, I will also add some features that have not been added to the image. This is because the additional features are less important than the main features being displayed.

## Stakeholder

The target audience for this program is anyone who is stressed. The game's purpose is to ease the stress whereby it allows the player to think of alternative methods to their real-world issues. However, when thinking of some individual target audiences, teenagers are vulnerable to stress. Therefore, are the most suitable option for targets.

Teenagers are in the most important stages of their lives. From developing physically to taking important exams that their future is dependent on. When teenagers grow up, they go through a phase called puberty. This is the process of transformation when a child's body develops into an adult body. During this time, the human changes physically and mentally. This is stressful since it can increase stress-related dysfunctions[[1]](#footnote-1). Moreover, the addition of exams causes stress as it is an important qualification for their future. All these important life struggles colliding with each other, it causes teenagers to go through a period of their lives of agonising stress.

|  |  |  |  |
| --- | --- | --- | --- |
| Stakeholder | Role | Interaction | Availability |
| Jahangir | University Student | - Functionality ideas  - Additional Feature to add | Weekly |
| Ray | UBS Employee | - Adding depth: different viable options the game offers | Weekly |

Jahangir is a student at a university. He is the stakeholder I choose since he fits the target audience's requirements. Since he is currently studying, he will be suffering from stress. Therefore, he is the most suitable as the feedback will be given on behalf of how they feel about the game.

Ray is an UBS employee. As mentioned before, my game is targeted for those who feel stress. This also includes adults as my game is for any age. Therefore, I picked Ray to be my stakeholder because work can be stressful, and his feedback will be very useful, especially I can gain feedback from an older age group.

When speaking to Jahangir about my plans, he decided that health that regenerates would be important. He told me that having regeneration lets the player play for longer periods of time. Without regeneration, player might die quicker and will not have much play time before they die. This is an important feature to add since stress will increase with frustration. If lower skilled players decide to play, their dodging skills might not be at the optimum level since the game is fast pace. Therefore, they will die quicker which will lead them to be frustrated as they die before they can even get started. Therefore, not adding regeneration will cause frustration, increasing stress which is not the point of my game.

## Research

### Game Reference/Competition

The Hack and Slash game, ‘Castle Crashers’ is a perfect example of what my game ideas revolve around. From the simple graphics to the smooth combat, the game is a perfect example of how many can immerse within the game and unwind. ‘Castle Crashers’ is a 2D Hack and Slash where it contains a simple story. The player has to save the princess and the king’s magic crystal. For this to happen, the player must defeat enemies with their melee-based weapon. On the image [source: ‘Castle Crasher’] we can see the orange playable knight using a melee weapon and partaking in combat. This is the main concept of Hack and Slash, where the player’s goal is to defeat enemies.

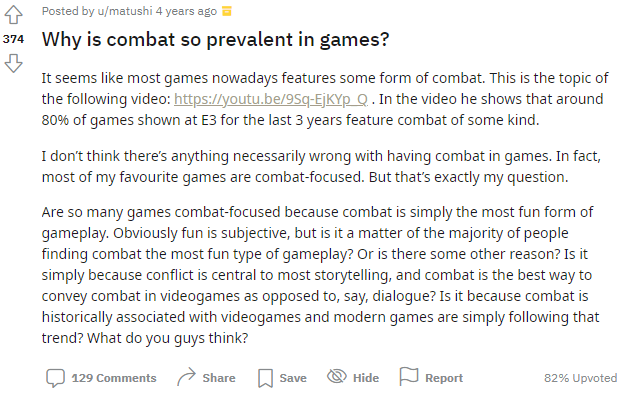
|  |
| --- |
| C:\Users\16AliMo\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\7F563D67.tmp *(Castle Crasher)* |

However, the problem lies within the engaging storyline. ‘Castle Crashers’ has a simple but addictive story. Therefore, the player will be immersed in the game for hours. As a result, they forget about their real-world tasks and deadlines and before they know it, they are stressed once again. When people want to destress, they want to take a brief period away to relax the brain. Usually, just taking 10 – 15 minutes to escape the real world is enough for someone to destress. Therefore, when the player starts playing games with storylines, they submerge themselves into more stress.

### Why combat orientated?

The program is exclusively going to contain only combat. Having combat as the core mechanic will divert the player's attention away from stress. This is due to the fast pace and engaging environment. A reddit[[2]](#footnote-2) user ‘ElysiX’ replied to a user ‘matushi ’questioning “Why is combat so prevalent in games? [Source: ‘matushi Question.’] Firstly, the question is asking why combat is increasing within games. He stated while watching E3 (Electronic Entertainment Expo), a gaming event for publishers to release new games. That for the consecutive years from 2015, around 80% of games contained combat. ‘ElysiX’ mentions [source: ‘ElysiX Reply’] that combat is easy for the player's mental digestion. He was implying that consuming a combat game allows a mental process that causes the clearing of a person’s stomach, reducing stress. On top of that, he mentions the use of challenge. Comparing puzzles, sports, or racing games also contains a challenge but it is the same basic idea. However, combat is thrilling as no combat is the same. From being faster than the other players/enemies to learning different control and combos to defeat enemies, all the combat is different. Combat as ‘ElysiX’ mentions is *‘mildly engaged to relax after their daily troubles.’* suggesting that combat has the power to grant people relaxation and composure. It allows someone to loosen the clouds of clustered thoughts in their head. For that reason, combat is the optimum choice when someone wants to destress.

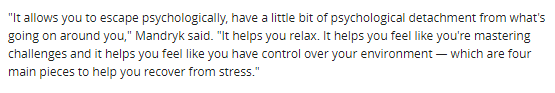
A way I am thinking why the game should be combat orientated is by seeing the enemies as the stress and the real-world problems. Having the player as the standalone character within a pixel word, gives them a satisfying reward since they slash through their problems all by themselves. By doing this themselves, it creates a sense of reward and relief. The achievement of removing a certain challenging task removes the burden, but also you feel ‘right’ when you complete something yourself.

  
 *(matushi Question)*

  
 *(ElysiX Reply)*

In addition, it has been researched that action games in general relieve stress.

During this period, the research was done during the Covid-19[[3]](#footnote-3) pandemic, this was one of the most stressful times for the entire world. One way many coped through the stressful time was through gaming. A professor at the University of Saskatchewan, Regan Mandryk[[4]](#footnote-4) said that *“gaming can help reduce stress and improve mental health.”* They looked at people from the ages of 18 to 55 where Mandryk said that playing video games can benefit people’s emotions and mental health. They gave the example of the game ‘Fortnite’[[5]](#footnote-5), on [source ‘Mandryk Quote’].



(Mandryk Quote)

By going through this passage, they mention that *“it allows you to escape psychologically.”* This is referring to the psychological damage done to the body when stressed. Therefore, when gaming, it allows someone to forget their problems and rethink different approaches to tackle their stress. Additionally, *“mastering challenges”* and *“have control over your environment”* helps individuals overcome their stress. By breaking this down, we can see stress is a problem humans face when they have a challenge that is too difficult to comprehend. Therefore, when they play a simple combat game, it helps them feel like they have control of their surroundings.

### Final Conclusion

After doing much research into what my game is going to be about and the steps required to complete my game, I have concluded that my game is going to be called: ‘Worlds Collide’. My justification of the name is not obvious, but once I explain it, then it will make a lot of sense.

Justification: When it comes to the lore for Worlds Collide, I want the player to create their own lore and story (lore meaning the story known to everyone). I want to leave it up to the player’s imagination. However, I am going to add lore in how the name was chosen. Firstly, when entering the game, the map is going to display 2 worlds colliding. The reason for the worlds colliding is due to alien activities. These alien activities are an underlying meaning for stress, where they represent the whole stress function as a whole. I mention this in the ‘Why combat orientated?’ where I said the enemies the player is fighting is a visualised version of stress. Which is why the aliens is similar to stress in the human body. The worlds and galaxies they exist on have not fully collided, but are in the progress too. The player is not from our world (Earth), where their goal is to save the world from these evil schemes and save the world. If I decide to carry on developing the game, I want to create lore to the game and show that the worlds have collided when the player losses. Now, the true meaning of Worlds Collide is the right at the start of the documentation. *‘The immersion of games**’*, this is the phrase that I will come back to when talking about my goals for my game. The word ‘immersion’ is the main reason for the game name. The underlying meaning for Worlds Collide is that I want to bring the immersion of the real and virtual world merging. I want the player to be immersed within the virtual world of the game through the real world. Therefore, even though the game map is going to be colliding, I also want both the real and virtual worlds to collide. Thus, having the game named Worlds Collide.

## Question and Answers

I am going to be asking my stakeholders and to my target audiences a few questions. The answers to these questions will help me further develop my game. This being a chance to hear people’s opinions and views of what makes a good game, especially a combat game and how to solve certain issues when developing my game.

### Questions

These are the questions that I am going to be asking. These few questions are going to be asked to my stakeholders Ray and Jahangir and to Mohammad a target audience:

1. Do you play games?
2. If yes, does playing games relieve stress/allows you to escape the real world?
3. Do offline or online games destress you more? Why?
4. Have you ever played games containing combat?
5. How do you feel about combat games?
6. Do combat games help you relieve stress/aid you to calm down?
7. What makes combat games satisfying?
8. Have you heard of the ‘Hack and Slash’ genre? If yes, do you enjoy them?
9. Do you still to this day play ‘Hack and Slash’ games? If not, why?

*The answer is from Jahangir who is my stakeholder. He is a university student, where he has an extensive knowledge in games.*

**Q1. Do you play games?**

A1: “Yes”

**Q2. If yes, does playing games relieve stress/allows you to escape the real world?**

A2: “They tend to”

**Q3: Do offline or online games destress you more? Why?**

A3: “Offline games, less competition, allows me to be by myself with my thoughts”

**Q4: Have you ever played games containing combat?**

A4: “I have”

**Q5: How do you feel about combat games?**

A5: “I really enjoy them if the mechanics are made well”

**Q6: Do combat games help you relieve stress/aid you to calm down?**

A6: “Somewhat, it depends on the difficulty”

**Q7: What makes combat games satisfying?**

A7: “Seeing big amounts of damage”

**Q8: Have you heard of the ‘Hack and Slash’ genre? If yes, do you enjoy them?**

A8: “Yes I have”

**Q9: Do you still to this day play ‘Hack and Slash’ games? If not, why?**

A9: “I do, I find them quite stress relieving”

The response from Jahangir, from the answer of A3 given to me, my game being offline will be beneficial to achieve my goal of relieving stress. *“allows me to be by myself with my thoughts”* has suggested that being offline allows the player to control their thoughts and have time for themselves. It made me realise from Jahangir’s answer, having my game being online, will make the player more stressed since the sense of competition will bring adrenaline flowing through their body. This is because of the stress of not losing to their opponent. Therefore, containing enemy AI will not be too difficult to defeat and will help the player gather their thoughts.

*The answers are from Ray who is my stakeholder. They are a UBS employee.*

**Q1. Do you play games?**

A1: “I do. I’ve played them since I was a kid.”

**Q2. If yes, does playing games relieve stress/allows you to escape the real world?**

A2: “They can do when the challenge is at the right level. Games that are too hard can cause me frustration.”

**Q3: Do offline or online games destress you more? Why?**

A3: “I don’t see a real difference between the two in terms of stress relief. I will tend to play co-operative online games though rather than competitive ones.”

**Q4: Have you ever played games containing combat?**

A4: “Yes. Combat games probably make up a 30% of what I play.”

**Q5: How do you feel about combat games?**

A5: “I enjoy them when they have good, and varied combat mechanics.”

**Q6: Do combat games help you relieve stress/aid you to calm down?**

A6: “They can do when the balance between challenge and reward is right. I am not a massive of “souls-like” games.”

**Q7: What makes combat games satisfying?**

A7: “Responsive combat mechanics that you can experiment with, and a strong but not excessive challenge.”

**Q8: Have you heard of the ‘Hack and Slash’ genre? If yes, do you enjoy them?**

A8: “I have, I enjoy them.”

**Q9: Do you still to this day play ‘Hack and Slash’ games? If not, why?**

A9: “I do. Not so much the old school 2d ones but if a good 2d scrolling beat em up style games come along I will give it a go.”

From Ray’s answers of A9, I concluded that the art style is going to be an important factor when designing my game. When doing some research, I found out that the ‘Hack and Slash’ genre is in decline. This is because of the nostalgia gained when playing games falling under the genre.

In digging deeper into the centre of the 'Hack and Slash' games, the main reason for the nostalgic feeling players got from playing, was the old graphics. During that time, games did not look spectacular, like the new RTX graphics (an advanced GPU (Graphic Processing Unit)), but had what looked like a ‘lame’ effort put into them. At that time, the engines to create games did not have much. Therefore, people's expectation for the games to look good was not high. As a result, people did not look at them. In addition, the core gameplay of 'Hack and Slash' was simple. Defeat enemies. The animation did not have much to them and the concept of the game was simple. Now the new games have so much story and overly complicated controls, which is hard to get started with the game.

As a result of Ray's answer of A9, I have gained greater insight into how to beat my competition by concentrating on the user's emotions, and it shined a light on an alternative way to ease stress. Therefore, I have decided to use pixel art as the focus of the game art. This is because I want players to play the game to recreate the sentiments of nostalgia. I think having my game focus on nostalgia will help players relieve stress. This is because the satisfaction gained by revisiting their childhood/old memories brings comfort and happy feelings, which is a way to mitigate stress.

*The answer is from Mohammad who within my target audiences. He is in sixth former, therefore has a great deal of stress.*

**Q1. Do you play games?**

A1: “Yes, I do play games.”

**Q2. If yes, does playing games relieve stress/allows you to escape the real world?**

A2: “I think that playing games does relieve stress to a certain amount because it makes me forget all the worries I have. Also playing games, in my opinion, is one of the best solutions to escape the real world because I can immerse myself into a whole different world and forget all my worries.”

**Q3: Do offline or online games destress you more? Why?**

A3: “This question is quite debatable: games, whether it is online or offline can be subjective due to a person's interest. In my case, I think both are good for destressing me because, if the genre of the game is either fighting or strategic, it can help me forget the real world quite easily.”

**Q4: Have you ever played games containing combat?**

A4: “Yes, I have played games that contain combat.”

**Q5: How do you feel about combat games?**

A5: “I really like combat games because they are really competitive and interesting. However, I don’t like shooting games.”

**Q6: Do combat games help you relieve stress/aid you to calm down?**

A6: “Depends what type of combat game we are talking about. If we are talking about a competitive game, then no, it will not. However, if we are talking about a game that has combat but is not competitive, then yes, it will calm me down.”

**Q7: What makes combat games satisfying?**

A7: “Combat games, in general, are very satisfying for me because killing the enemy/opposition really makes me eager to continue playing.”

**Q8: Have you heard of the ‘Hack and Slash’ genre? If yes, do you enjoy them?**

A8: “I have heard of them and yes, I have played some. I enjoy ‘Hack and Slash’ games because they are very appealing due to their unique style of melee combat.”

**Q9: Do you still to this day play ‘Hack and Slash’ games? If not, why?**

A9: “Unfortunately, I don’t play ‘Hack and Slash’ games anymore. There is no particular reason. I just haven’t heard/seen a good ‘Hack and Slash’ game in a while.”

From everyone's responses, I want to highlight Q7: ‘What makes combat games satisfying’. I have concluded that everyone has a varied response to what makes combat feel pleasing. In Jahangir's response, he says that big damage will make the game satisfying. Through my interpretation, he wants to see enemies defeated relatively quickly. In Ray’s answer, they express that game mechanics are fundamental. They should be smooth and clean, and the input must be responsive to the gameplay. Finally, in Mohammad’s reply, he says defeating multitudes of enemies gets him eager to play more. Which is similar to Jahangir’s point of seeing high damage to enemies. These varied responses, it has opened a new sight, which has allowed me to largen my field of view when it comes to my target audiences. Since everyone is different, I will need to consider a lot of different opinions therefore, containing these answers has made me think of distinctive features required to add.

## Features

Adding various features to the program allows ease of accessibility. Firstly, without features, a program might find it harder to stand out from the rest of the competition. This will cause many problems as money, time and effort would have been wasted on the project. Secondly, features make it easier to use the program. For example, having a game a login screen makes it easier for the user to log in straight away and access the game. That being said, my program is also going to contain many key features.

* GUI (Graphical User Interface):
  + A GUI is an essential feature for modern games. GUIs are not only used for games, but are used everywhere else. This is because GUI allows users to interact with the software by either clicking or tapping the screen. It helps users understand what is happening in the program and what they might do to combat certain situations. Without a GUI, the other alternative is command-line interface (CL). Command-line interface is a text-based interface that can also be used to run programs, interact with the computer, etc. However, this is much harder to decipher since the use of text rather than graphics is difficult to understand. For that reason, I am going to be incorporating GUI into my game. This is because games require GUIs to play and make the user recognise what the tasks are. For example, it will be easier to see combat, health, death, game over, and many other systems rather than seeing them as text. In addition, GUI will relieve stress than using command-line interface since GUI are easier to interpret. Command-line interface will just increase stress since reading massive amounts of text on the screen is not thrilling. But, the use of GUI allows the user to immerse themselves easily into the game since it is easier to interpret what is going on.
* Menu Screens:
  + Including menu screens will make the player able to navigate through the different interfaces. This will make the game easily accessible and interactable. There is going to be 2 menu screens: Main menu and pause menu. Firstly, the player is going to be introduced to the main menu screen. This will give the player the option to be in control of the game. The player will be given 3 options when entering the main menu screen. These will be ‘Play’, ‘Controls’ and ‘Quit’. This will make the game easier to navigate and access. The next menu is the pause menu. When the player is in the middle of the game, they can stop the gameplay and the pause screen will be displayed. When they pause, it should display 2 options: ‘Resume’ and ‘Quit. Having these two options rather than adding nothing makes the menu easy to resume and quit the game when the player wants. If I added only added the title ‘Game Paused’, without adding the designated buttons, it could make the player confused on what to do when trying to quit the game or resume playing. Having these menus makes it more efficient for the player to control the game and understand what is going on.
* Visible Buttons:
  + The controls allow the player to navigate through the game. I am going to make the buttons large so that the players can see them. Having unidentifiable buttons, by either the player not seeing them in general or being hard to click, will make it hard to interact with the interface and will cause the player to be annoyed. For that reason, having the buttons big and well-defined will make it visible for the player to navigate.
* Pause Menu:
  + Allows the player to pause the game when necessary. Having the pause menu allows the user to pause the game at whatever point they want. Since everyone has different situations when playing games, it is a good idea for the player to be able to pause so that they can return back from where they left off. Additionally, it makes it easier for the player to quit the game. This is because including the pause menu makes the quit option more accessible rather than the player force closing the window.
* Music:
  + Adding music will improve the players interest within the game. Without music, the game will be bland and boring. Therefore, adding sound fills in the empty void and makes the game more relaxing.

## Limitations

I think that the main limitation is the time constraint to create different maps, characters, and enemy designs. Pixel art is the main art style for my game. In pixel art, many individual pixels need to be coloured. In addition, the canvas size is 1280x720, and it takes a long time just for a single map to be constructed. Therefore, having multiple maps will take a long time to create, especially having uniquely comprised maps that will make my games stand out. Another issue is the character and enemy designs. Firstly, just coming up with a design for a character causes problem since I would need to think of distinctive characteristics and features that make up the character. This is because when creating and designing a character I need to think about the different components of what makes them visually appealing. After settling down with a design, I would need to create the same variation of the art, but make some small adjustments to the sprite for the different actions. Therefore, creating multiple characters and enemies will require a lot of time and effort to create them.

As I mentioned before, sprites are game assets that represent images. Each character and enemy are made from sprites. Therefore, when creating just a single character or enemy, there are going to be many sprites just for movement. This is the main reason I am making a 2D game since the only sprites that I am required to create is the left and right movement rather than up and down. Having multiple sprites does not only count for movement, it also counts for the attacks, jumping, taking damage and death. Therefore, it requires a lot of different sprites that comprise the final character/enemy. For that reason, the animation of the game is going to be hard to create due to the time needed to develop. Which is the main limitation since I lack the time required to pull off many designs.

However, a limitation to adding multiple maps, different sprites for each character and enemy, and various animations, is the amount of memory required to store the assets. If I do decide to add distinctive abilities for each character and enemy, it is also going to take more memory space since I will have to implement different sprites and animations. In addition, having all the unique designs and maps will require different sounds for each of them. This is a problem since the amount of memory required to store each image does not require an extensive number of bytes, however, having multiple of them is. In addition, the audio files are going to take the most amount of storage because there are going to be multiple. Not to mention, if a user has a low-end computer, it can cause lag, which is frame rate dropped, causing more frustration for the player. This would not reduce their stress.

## Requirements

### Hardware Requirements

Since this is a computer game, the user requires certain peripherals and software to play the game. Here are the hardware and software requirements and justification

* A monitor is required to provide a display of the visuals since the program is going to make use of GUI (graphical user interface).
* The keyboard is required for the player to move around. Using the ‘A’ (left movement), ‘D’ (right movement), and ‘Space’ (for jump).
* The mouse is needed to attack enemies.
* Speakers or headphones are needed since the game is going to output sound.

The internal hardware does not have to be complicated;

* it requires a basic GPU (Graphic Processing Unit) for the game to run. This is because we need a GPU to process information that is graphical.
* The game requires at least 1GB of RAM to play. Without it, poor performance and a loss of frame rates can occur.
* VRAM (video RAM) is required since I am creating a 2D game. The use of a graphical interface allows a smooth execution of the game.
* Having HDD or SSD is required to store sprites, images, animations, and sound since we need it to handle data.

In general, having a decent computer will allow this game to run without issues.

### Software Requirements

* The user is required to install the latest version python, since the entire program will be running on the python language.
* Pygame is essential for the user to install to their computer. Pygame is a python module that allows ease of creating games such as my game.
* Using the same up-to-date operating system (OS) is also required since the game will only operate on Windows.
* The user should have the appropriate drivers to play the game: keyboard driver, mouse driver, speaker driver, graphical driver

**Main Requirements**

|  |  |  |  |
| --- | --- | --- | --- |
| Number | Requirements | Solves | Justification |
| 1 | Menu Screen | Displays options for the player and allows the player to start when they like | Helps the user understand what they can do when entering the game. |
| 2 | Play button | Lets the user enter the game to play | The player may not be ready, just starting the game without any preparations can cause problems |
| 3 | Controls button | Lets the user to see the controls required to play the game | If the user does not know the controls, then they will not be able to play the game |
| 4 | Quit button | Lets the user quit the game | The game could be played on full screen therefore, there will not be a way to quit the game |
| 5 | Display the map when entering the game | The background of the game is required to make the game aesthetically pleasing | The visuals will help the player ease their stress since the game looks nice |
| 6 | Display the entities | Show the enemies and the character on the screen | If the enemy is not shown then damage will be taken randomly. In addition, if the character is not shown, then the user will not know what they are controlling |
| 7 | Create a camera | Allows the player to the centre of the game | When the player moves, the camera should follow them as it will makes it easier for the player to understand what is going on. |
| 8 | Movement | Allows the user to move in any direction and have the enemy track the players movement | If the player and enemy are static. Then the game will only be an image. Therefore, sprites to move around makes it a game |
| 9 | Animation | Having animations makes the game look good | Without the animations, the game will not look aesthetically pleasing and will look bland |
| 10 | Border | The map should have a border | The entities and map border should have a box around it so that it prevents any entity from leaving the map |
| 11 | Collision | Actions should not collide | When actions of jump and attack should not collide with each other |
| 12 | Health bar | Displays the health of the player | The user should know what health they have so that they can prevent death |
| 13 | Damage | Allows the entities to take damage | This should kill the enemies when their health drops to 0 or display game over when the player dies |
| 14 | Spawn positions | Spawn the entities | Spawn the player in the middle of the map while the enemies should spawn randomly on the map |
| 15 | Game over | Displays game over | When the players health drops to 0, it should display game over |
| 16 | Sound | Make sound for the correct actions | When certain actions are performed, the corresponding sound should be played |
| 17 | Pause | Stops the game | When the game is paused, it should stop the game and allows the user to resume playing or quit. This may be because the player is disrupted midway. |

## Success Criteria

**Essential Features**

Justification:

1. Display the menu screen with ‘play’, ‘controls’ and ‘quit’ options so that players have a choice of what they want to do when entering the game
2. When the ‘play’ button is pressed, it should allow the player to play the game
3. When the ‘controls’ button is pressed, it should display the controls
4. When the ‘quit’ button is pressed, it should quit the game
5. Display the map when entering the game, which shows that the user has entered the game
6. Create a camera which is centred on the player. When the player moves, the camera should follow them.
7. Display sprites for both player character and enemies, which shows that the game has begun
8. Allows the player to move left, right, jump and attack
9. Correspond the appropriate inputs with different actions. When the player either moves or attacks, then the game should respond to those action. E.g., if the player presses ‘A’ it should allow the user to move left
10. Add animation for the different actions performed by the player or enemy (e.g., attack, jump, death, etc). There will be a sprite sheet with all the different sprites that make up the animations
11. The entities should not be able to leave the map. For that reason, adding a border should prevent this
12. A combination of different actions should not collide with each other (e.g., attack and death). This is because having death as well as attacking will cause a problem since the entity should have died
13. Display the health bar for the player to know when they might lose
14. Reduce health of enemy or player when taking damage
15. Enemies should spawn at random positions; this makes it unique and helps the player not be bombarded with enemies. In addition, it reduces the likelihood of entity cramming, where there are too many entities in an area, causing lag. I can prevent this by limiting the number of spawns. I am going to do this by having a new enemy spawn once another enemy dies
16. When health enemy health reaches 0, they should die and a new one should spawn
17. When the player dies, it should display the game over and return to the main menu
18. Allow the user to pause displaying the ‘resume’ and ‘quit’ options
19. Play sound according to the action, so the game will not sound empty
20. Display a score for the player to know how long they have survived and to see their personal best

**Advanced Features (Low priority + cuttable)**

Justification:

1. Restore health when lost after a certain time. When the player loses health, it should return back to full if they have not taken damage after a while
2. Having different playable characters. This helps the game be more enjoyable as there are many different options
3. Add different styles of enemies with unique animations, as it can make the game more worthwhile
4. Having different playable maps for variation as the back drop can get repetitive
5. Implementing a high score function that counts scores for number of kills, which can make the game more competitive
6. Add another option button which represent the high score of players
7. Add special abilities (e.g., pressing the ‘E’ keys allows special attacks) to make combat more fun
8. Add crouching to the game as a dodging mechanic
9. Add stamina so that the player cannot spam attacks and requires the player to be more strategic
10. Give every enemy their own health bar. Similar to the player, each enemy is going to have a visual health bar, that is unique to all of them, which is a visual representation for the player. This is where they can be more strategic with which enemy they will target

# Design

## Solution to make it computational

### Top-Down Design

When describing decomposition on the analysis stage, I described how the top-down design is the most suitable method to solve problems. The main reason is the ease of understanding what the main functions are. By breaking down the main functions into smaller sub-functions, it makes coding the program easier since I know what is supposed to be added. This is known as modular programming where I am going to break down the core functions into smaller independent functions since it is easier to understand the main objectives needed to follow in order to complete the program. Therefore, when breaking down the functions into smaller and smaller sub-functions, we reach many subroutines. With these subroutines, I can understand the steps involved to code the functions to make the whole game.

In addition, the top-down design is more of a visual approach to seeing things. As I stated in the visualisation section in the analysis section, *‘Without saying a word, users can understand what is going on**’*. Even though there are some words within each box, the hierarchal format shows me the steps required to complete sections of program. The use of a visualisation rather than having plain script of instructions makes it easier to understand the key functions without missing a step. For that reason, there are going to be many top-down design diagrams that will unite to make an entire main function. Down below is going to contain the many different subroutines that I am going to be needing. By giving bold headings to my designs, I am able to understand different individual functions required to do.

**[Overall Concept]**

Justification: The diagram represents the main concept of the game. The subsystems will be broken down into even more clear and in-depth functions. This will tell me what the aim and goals are to complete the program. It lets me see the essential components to the game and the steps to achieve them. These main systems that will be broken down include (not in specific order):

* GUI
* Input
* Sound
* Animation
* Menus
* Enemies
* Health System

These all will be justified why they are the main systems and vital for the game.

**[User Interface]**

Justification: The user interface as mentioned in the features section part of analysis, is going to contain a GUI (Graphical User Interface). This is because it is much easier for the user to interact with the program, and easier for the user to destress since having aesthetically pleasing graphics ensures satisfaction to the player.

In addition, the GUI is going to contain colour. Having colour to a game also ensures satisfaction but additionally, a game containing colour allows the player to identify different objects. Having interactable entities and non-interactable objects as the same colour will make the game confusing since players will have to guess what is interactable. An example of this is door colours. In certain games, the use of having different shaded doors indicates that certain doors are interactable while the other ones are there for art purposes. The use of different shades makes the player know that they can interact with certain objects.

* Animations:
  + Animations are important to games. Without them, it is similar to like watching a still image. Adding animations brings life to a game, it allows people to read the character and enemies. In a more in-depth way of explaining, having characters animated brings out their personality, it expresses how the character is like and how the game might play out. For example, having a character tripping over countless of times indicates that they are clumsy and might belong in a simulation game. Moreover, adding animations adds an entertaining addition to the game’s movement. Having clean-cut movement makes the game easier to look at. Games with beautiful animations submerges the user into the game, believing that they within the game itself.
* Main menu screens:
  + Within the interface, it is going to contain the different menus screen. I will not go into detail about the menu screens since I went in-depth about them in the [Menu] top-down design section. However, the different menus are going to be interactable through the GUI and they are going to contain colour. In addition, when navigating through the controls button in the main menu screen, it will take the player to another interface where the player will be able to see the controls.
* Display the game:
  + Display the Map: When the player presses the play button, they should be directed to the game’s GUI. From there, the first thing they should see is the map. The map will contain colour so that it looks pleasing to the eye. This is because we want a background to my game. Having a plain background does not tell a story. Having a complete background tells the player what setting they are in and who the character is. For example, having a post-apocalyptic background tells the player that they are in a world where there is no other civilisation. That the world they are in is dead with no signs of life and that they have to protect themselves from danger. ‘The immersion of games’ is the quote I mentioned in the problem section at analysis where I want the player to feel as if they have been consumed by the game. Having the player consumed means that they have sunken so far into the game, that they have forgotten their real-world problems and have overcome their stress. This is the main problem my game is solving.
  + In addition, the map is going to be containing borders. The border is not going to be visible to the player. Having visible borders will be distracting since the player will keep diverting their eyes to toward the out of place border. Therefore, having them invisible will make it easier for the player to concentrate on the game. One way I will make the player know that they have hit the border is through collision. When the player rectangle (which is a box around the player) touches the border rectangle, then the player is not able to move in that certain direction. This tells the player that there is a border and that they cannot proceed further.
* Display sprites:
  + Along with the map loading, the individual sprites should load. The first sprite that should load is the character sprite, where they will spawn right in the middle of the map. The second sprites that should appear on screen are the enemy sprites. The enemies are going to be spawning from different ends of the map. The character and enemy sprites are going to have unique animation that will be justified in the animation diagram below.

**[Inputs]**

Justification: This diagram represents the different inputs the program should recognise and execute.

* Keyboard Input
  + The keyboard is going to contain 4 keys that the user can press. When these keys are pressed, the program should correspond with the correct actions.



The image outlines the keys that is needed for play the game. The keys ‘A’, ‘D’, ‘Space’ and ‘Esc’ is outlined in yellow indicating that these keys will be used. The ‘A’ key will be used to move the sprite left. The ‘D’ key will move the sprite right while the ‘Space’ key will allow the sprite to jump. The ‘Esc’ key will allow the player to pause the game. When the player needs some time off due to problems outside of the game or they do not want to play, they can pause the game. When the game is paused, it should pop up a pause menu where the user is given the choice to either resume playing or quit the game.

* Mouse Input
  + The mouse will be needed for the player to attack. A mouse is commonly used in games since it easier to use and control. Mouse’s make it easier to respond to certain stimuli.



The image of the mouse shows a yellow outline. This yellow outline highlights the left click. When the player clicks on the left click, then the game should produce a response when the character sprite attacks. The player will know when they have attacked is when the sprite animation occurs.

**[Sound]**

Justification: Sound is an important factor to take in. Without sound in games, it will make them dull and boring. Having sound immerses someone into the virtual world. This is because sound usually tells a story. Depending on the mood of the game, the right corresponding sound or music can play, which can rush emotions into the player as if they are experiencing the characters emotions themselves. This creates a strong attachment to the game since it can provoke a connection for the player, believing that they are in the same wavelength with the game character. This can aid players into having a deep links with the character itself, allowing them to destress when playing that game again. This is ultimately the main goal for my game. Having the correct sound played when the players play so the game can destress them.

* Background music
  + Within opening the game, when the main menu screen is displayed, the music should play. This will allow the player to synchronise in to the game and feel the flow of the game. Once the user hears the music, it creates an immersion where the player will understand the sort of game it is and how it will turn out.
  + When the user enters the game, music should play. The music is going to be an 8-bit fast pace music it will set the player's pace.
* Sprite Sound
  + When the player does certain actions or the enemy AI does an action, which are: attack, jump, take damage or die when HP (hit point) goes to 0, different sound should play that corresponds to each action. Firstly, when the AI or character attacks, it should make a slash sound where it indicates that an attack action has been done. The jump, which only the player character can do should make a ‘boing’ noise that shows that the player has jumped. The damage being taken for both player characters and enemies will sound different from each other. The sound of the damage taken will have a fade in, fade out noise. By having the sound fade in and out, is an easier way to understand that damage has been taken. Finally, the death sound should sound would be different for both player character and enemy. The different change in sound will indicate whether the player died or the enemy died.

**[Animations]**

Justification: The inclusion of animations will bring the game to life. Having animations tells the personality of the world and entities. Therefore, when creating the different animations, I want them to be unique so it brings a powerful expression to the player. In addition, animations will make the program seem smoother, since everything will be more flowing rather than being all choppy. With the animations, I can add realism to the game as everything is graceful similar to the real world.

* Player animations
  + The player animation is going to be comprised into 4 main sprite sheets. These sprite sections include: idle, move, attack, jump, and death. The first animation is the idle animation. This is where the sprite it standing still as there is no user input. When the sprite is at a standstill, the sprite should bop up and down (the sprite should move up and down) at a slow pace, displaying to the player that the sprite is not moving.
  + The second animation is the attack animation. When the player clicks the left click on the mouse, the game should respond with the character sprite doing a slash animation. This is where a white streak will appear showing that the player is attacking. When the enemy is attacked, they should blink white to show that the players attack has gone through. This will indicate to the player that the hitbox of the enemy has been hit.
  + The third animation for the player is the movement. The player should be able to move left and right of the screen. When moving either left or right of the screen, a running animation should be displayed. However, there is a limit to how much the player can run left or right. This is due to the borders. When the player moves to far to the left, they will eventually hit the border. When the border is hit, the user will not be able to move left and the sprite will stand still changing to the idle animation. The same thing will happen to the right movement.
  + The next is the jump animation. When the player presses the ‘Space’ key, the character entity should respond by jumping. When the player jumps, the corresponding animation should be displayed. This is where the sprite bends the knees and then lifts of from the ground. When the sprite jumps, there will be particle effects indicating that the sprite has jumped. However, the sprite will not stay airborne forever. After a few seconds, the sprite will fall back to the surface due to gravity. For the animation of when the character falls, the sprite will have their arm up in the air. Once the character hits the ground, then the sprite will change to the idle animation if no other input processed.
  + The final animation for the player character is the death animation. When the health integer value drops to 0, then that is an indication that the player has died. When the player dies, there will be a sprite animation. The animation will be the sprite falling, where after a second, the sprite will turn into particle flying back into the atmosphere.
* Enemy animation
  + The enemy animation is going to be made of 3 sprite sheets. Within the sprite sheet, the enemy is going to contain attack, move and death. The first animation is the attack animation. When the enemy attacks the player, there will be an attack animation. This attack animation is going to be represented by a red swipe. When the player is hit, they should blink red to indicate that the player has taken damage.
  + The next animation is the move animation. When the enemy moves, they can only move left or right. Usually, I do not have to worry about the enemies having to walk too far too the left or right, since they will be tracking the player’s entity. However, in the circumstances that they move too far to the left or right, like the player, then they will stand still since there will be a collision with the border and the enemies hitbox. The animation of the enemy moving left or right will be a semi-paced run, trying to chase to the player.
  + The final animations of the enemy sprite are the death animation. When the integer value of the enemy drops to 0, then that means the enemy has died. When the enemy dies, the animation that would be displayed them falling to the ground and turning into ashes, floating into the atmosphere until they dissipate.

**[MENU]**

Justification: This diagram represents the different menu systems. By having these menu systems, it ensures that the user is in control of the game, allowing them to start, pause and quit the game whenever they like.

* Main Menu:
  + Firstly, when we head into the ‘Main menu screen’, we have three options to pick from. At the start, there is a ‘Play button’ this allows the user to enter the game. After, there is a ‘Controls button’ which should warp the user to another interface where it displays the keyboard and mouse controls. Finally, on our main menu screen, it should display the ‘Quit button’ where it allows the user to quit the game. These will conclude the main menu and what the user is supposed to expect.
* Pause Menu:
  + Furthermore, there is a pause menu. On the diagram, we can see that ‘Pause menu’ is defined. The use of the pause menu allows the user to pause the game at whatever point they want. When the pause menu pops up, it should display two options. Firstly, it will show the ‘Resume button’ where it allows the user to return to the game from where they left off. Secondly, there is going to be a ‘Quit button’ where it allows the user to quit the game.

This concludes the menus the game is going to include.

**[Enemy]**

Justification: This diagram represents the enemy. This tells me how the enemy is going to be laid out and its key functionality. This will make it easier for me to see the main practicality of the enemy. The enemies are the main component to my game. This is because my game is about combat. Not laying out the important functions of the enemy could make them unpractical. When it comes to the coding, not having the basic layout of the enemy will make it difficult to visuals the key aspects of the enemy. Therefore, this top-down design diagram makes it easier to program without missing any steps.

* Spawning
  + When the game starts, the enemy should spawn at either end of the map. When they spawn, only a set number of enemies can spawn. This is because having too many enemies can result in several problems. The first problem will cause the game to crash. This is because not having a set number of entities on the screen can overload the system and cause frame rate drops and lag. Eventually, the game will crash. The second reason is that even if the game did not crash, it could obstruct the player’s view. This is because if there are too many enemies swarming the player, it could cause too many enemies on the screen at the same time, resulting in the player being unable to see their character. When the enemies spawn, they will spawn at the far left and the far right. This will allow the player to adjust themselves and get ready for the combat. When an enemy die, another enemy should spawn after that where they will spawn at either end of the map.
* Health
  + Health is an important variable for the game and enemies. When setting the health for the enemy, they will have an integer value set with them. The enemy will only lose health when the player attacks them. When the enemy is attacked and the hitbox is hit, then the enemy will take damage. When damage is taken, then the maximum integer value will lose a set amount of health. When the integer value falls to 0, the enemy dies. When the enemy dies, then we return to the spawning mechanic. The enemies will not have too much health since I want the player to defeat the enemies at a relative pace. In a more gaming term, the enemies that take a long time to defeat will be known as ‘tanks’. This means that they have an incredible amount of health and take a long time to kill. If I have enough time, I think the implementation of adding ‘tanks’ will be on the list of my priorities. This is because I think if the player progresses through the game for an extended period, I think introducing stronger enemies and ‘tanks’ will add more levels of complexity. In addition, these ‘tanks’ could be bosses when the player reaches a certain distance into the game. However, as stated before, this will be a lower priority due to the time required to create the game.
* Movements
  + I went into detail about the movement of enemies in the animation diagram. To reiterate, the enemies can move left and right. However, when the enemy move left and right, they cannot move too far to the left or right. This is because of the world border. Since the map is not infinite, the entities cannot move out of bounds. Therefore, when the enemy reaches the border, it will clash with their hitboxes and stop them from proceeding in a certain direction. Another movement is the attack and death. When the enemy attacks, it should have a visual representation that the enemy is attacking. Similarly, there will be a death animation.
  + The most important movement for the enemy is tracking the player's movement. This is vital since I want the enemy AI to seem as if another human is controlling them. As soon as the player clicks the ‘play game’ button, the enemies will spawn and track the player. When tracking the player, it will move towards them in the direction the player is. When the enemy hitbox collides with the player's hitbox, then the enemy will start the attack animation. This will make the game more realistic, since in real life when someone is at a perfect distance to reach someone else, they will engage in combat. For that reason, when the enemy is at the perfect range (when the hitboxes collide) they will attack the player. However, for them to do that, they will have to track the player's movement.

**[Health system]**

Justification: The central part of the game requires a health system. Without a health system, it could result in the game just being an end-to-end fight. This will make the game boring since the only thing the player is doing is fighting without having a main goal. When adding the health system, the player has a goal of not dying. The challenge and thrill of not having their health to 0 and trying to survive as long as possible will bring the desire and passion to defeat enemies. The player must think on the spot about the different actions they need to do to stay alive. Having the enemies have health adds a reward element. This is due to satisfaction, as defeating opponents that the player has been fighting endlessly until the enemy’s health drops to 0 is an achievement in the player’s eye. During the research section in the analysis under the ‘Why combat orientated?’ heading, I mentioned that I see the enemies as real-world problems and stress where the player aims to defeat as many of their problems. Therefore, having enemies that disappear when they defeat them brings in reward for the player. One question might be: “Then why does the player have health, if they die, will that not mean that the player has lost to their problems?”. With that, I respond by saying that: since the game is a fast pace game and has a lot of action, I think the player will be immersed too much into the game to ever think that they have lost to their problems. Allowing them to retry the game as much as possible, ensures that they will not give up. Another way of thinking about this is: losing to the enemies gives the player the passion for finding another way to think of alternative routes and solutions to solve their problems.

* Player
  + The player is going to have a set integer value for their health which will be represented at the top of the screen. When the player takes damage by getting hit by the enemies, then their health will decrease. The integer value given to the player will decrease by the amount of damage taken. Over time, if the player does not take any damage, or is taking significant lower damage, then their health will regenerate. This is where an addition will be done to the integer value, increasing it until the health reaches the maximum value. However, if too much damage is taken, the integer value will drop to 0, meaning that the player has lost. This then should display a game over screen.
* Enemy
  + Enemies also have a similar concept to the player’s health. The enemy will have a set integer value that is represented as the health. However, the difference between the player and the enemy is that the enemy is going to have a different integer value from the player. Also, the enemy is not going to have their health displayed like the player. I think exhibiting the enemy’s health will distract the player since there would be many health boxes that will cover too much of the screen. Additionally, there are going to be enemies respawning once the old enemies die, which can mean implementing new health boxes for each of them. This means that I have to remove the old boxes with the new ones but, due to the time required to create the game, it will need to be added as a low-priority/cuttable feature.

## Development methodology

### System Lifecycle

The development methodology that I will operate with is agile, which is a form of iterations approach. If I used the waterfall methodology, then there would be a high risk. This is because, unlike the agile model, the waterfall model is not suitable for object-orientated projects. Since my code is going to make use of object-orientated code, the waterfall method is not able to handle the complexity of the code. In addition, there is a lack of user feedback. Without the feedback from my stakeholders, it could cause many faults to my game. Stakeholder feedback puts my attention on the essential features that helps me better develop my program into a more user-friendly software. Jahangir, one of my stakeholders, suggested to add health that regenerates when lost. This helped me realise that players want to play the game longer without any disruptions. Having the player constantly loosing health and dying within 5 minutes in the game will stop the player from having fun. For that reason, having health that regenerates allow the player to play the game longer. This suggests that stakeholder feedback is vital for my game's success.

Since I am using a top-down design, many of the problems are going to be broken down into many modules. For that reason, the use of agile is going to be essential. This is because when using agile, each problem is worked through iterations. The use of making multiple iterations helps me test my ideas quickly and improve from the last one. Once I complete an iteration, I can test and review each iteration to ensure that they work. Additionally, feedback from my stakeholder will be given as soon as possible. This is because once I have developed an iteration, receiving feedback will allow me to change any problems within the code. The flexibility of agile is significant since I will be able to alter any idea or change solutions. This is important due to the time constraint. Since my program is going to be large, and since I am the only sole developer of the program, it is going to take much longer to create since I do not have a team of programmers to help me. Therefore, due to the limited amount of time required to create the game, having the flexibility and stakeholder feedback benefits me to progress through my game efficiently.

## User Interface

I will need to make each screen because my game will use a GUI (Graphical User Interface). In order to scope out the fundamental design of the game, I will construct abstracted versions of each interface in the sections below. After that, I will be able to expand on the fundamental concept to create more intricate game designs.

### Main Menu

This is an abstracted view of the main menu screen. It will have the entire interface annotated to see each individual function and why I have decided to include them in the user interface.

A ‘Play Game’ button that allows the user to player the game

This is going to display the name of the game. ‘'z

A ‘Quit’ button that allows the user to quit the game.

A ‘Control’ button that once clicked, display the controls of the game.

It is going to contain the character sprite. This will be a PNG background, meaning that it will be transparent

Play Game

Controls

Quit

Background

### Controls

This is an abstracted interface of the control page. This is going to display the controls to the user. The controls interface is important since without it, the player is not able to understand the keys required to press to play the game.

Display the heading ‘Controls’ to indicate the user is on the controls interface

Return the user back to the main menu

Shows the user that the left mouse click is used

Attack

Display what the controls are for the user to play the game

Esc

Jump

Pause/

Return

SPACE

D

A

Move

Keyboard

Return

Mouse

Controls

### Pause Menu

During the in-game screen, the player has opportunities to pause the game. When they do, a pause menu interface should pop open.

Displays the heading ‘Game Paused’ to indicate that the game is paused

Allows the user to quit the game

Allows the user to resume from where they left off

Quit

Resume

Game Paused

### Game over

When the player health drops to 0, then the player has lost. When the player has lost, then the game over screen should be displayed. When the game over screen pops up, then there should be 2 options for the user can do. Firstly, the user can return to the main menu screen by clicking the ‘Return to menu’ button and the allows the user to quit the game by clicking the ‘Quit’ button.

A button which allows the user to quit the game

A button that allows the user to return to the menu screen

The ‘Game Over’ headings indicates that the player has lost

Quit

Return to menu

Game Over

### In-Game Screen

I already have an abstracted game screen overlay for the game which was mentioned under the visualisation section in the analysis topic. However, I am going to create a new one to stay in line with the other interface designs. When the game is loaded up, the player is going to have their character sprite loaded up. There is going to be a back drop so that the game does not look bland and has life to it. The enemies are going to be spawning from either end of the map. The player’s health bar is going to be at the top left of the screen which will indicate how much health the player will have. There is going to be a score system, where the users will receive scores on the number of enemies they defeat. It will increase until the enemies are defeated.

This displays the score the player receives when defeating enemies

This is the player’s health bar which represents the health of the player

Backdrop

This is the ground surface where the player and enemies stand

The ‘sun’ sign represents the player

The ‘no access’ sign represents the enemies, which are at both ends of the game

### Stakeholder feedback [User interface]

When speaking to my stakeholder (Ray), he gave me some feedback for my user interface.

**Important Feedback:**

* Interface Design Justification:
  + The first feedback that was given to me was implementing a game over GUI when the player dies. When speaking to my stakeholder, I completely forgot about adding a game over screen, but once Ray gave me feedback, I came to the realisation to add the game over screen. The game over screen is under the ‘User interface’ heading.
  + Another feedback given to me for my interface was the main menu screen. The feedback that was given was to make the game button a little to smaller. This is because in modern games the buttons are small to compensate the background image behind. He said that having the background allows the player to understand what type of game they are getting into and what the game might be about without playing it.

However, I am not going to be incorporating this feedback. This is justifiable since I want the buttons to be a decent size for usability. Having the buttons too small like modern game, might make it hard for certain players to navigate through the menu. Since my game is about stress relieving, I do not want certain players with health problems to suffer with miniature buttons.

* Mock-up of the game:
  + When displaying the drafts of the in-game screen and the character design, Ray suggested to make a mock-up of the designs together. When I think about, this feedback is super useful. This is because I have not thought of how the character design and the map will go with each other. I had to think whether the colours are distinctive for the player, if I used the right colour palate, if the character size is relative for the map size and many other underlying problems there might be. The question that I now asked myself was: “Do the designs fit?” when designing my game.

Some adjustments I made when doing the mock-up of the game are:

* + - Changing the pixel size of the character from 98x98 pixels to 80x80 pixels. This is because the character size was too big compared to the other objects in the game.
    - Changing some of the colours. I have adjusted some of the hue and saturations of the background so the player character does not blend into the background of the game.

Down below are the drafts for the design in which Ray and Jahangir both approved.

**Cuttable Feedback:**

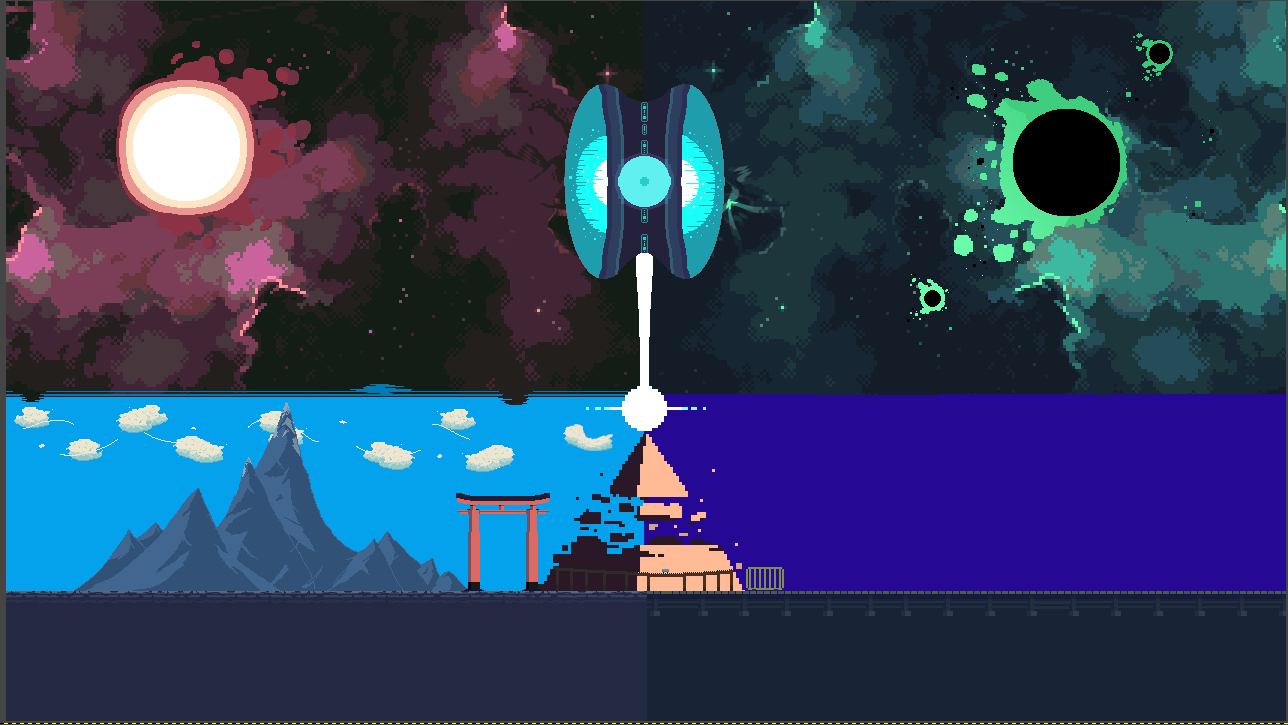
This is features that will not be added unless I have the necessary time.

* World Progression:
  + When speaking to Ray, he asked: “How would the world play out”, this got me thinking as all I thought about was how the player is going to be defeating enemies and getting a personal score for themselves. However, due to Ray’s question, it gave me a thought. Since my game name is Worlds Collide, I think eventually I will change the aim of the game for the player to save the worlds from colliding. The player will have to save the world and reach a final boss, once reached, then the player has successfully saved the world. I will visually show that the player has won by making the worlds turn into a beautiful scenery. If they lose, then right before the game over screen, I would make the game map collide and fuse, showing that the player has lost.

These drafts are subject to change. I have stated that to both Ray and Jahangir.

**Character Design [Draft]**



**Game map design [Draft]**

### Final User Interface Designs

#### [Main Menu]

Justification: This is going to be the main menu screen. I decided to change the user interface to make it more usability friendly. Having the buttons at the middle of the screen rather than the right of the screen will make it visibly easier for the player to comfortably look at. In the basic draft of the menu, I made the buttons get smaller as they descend. This will prove an issue to those with OCD (obsessive compulsive disorder) since not having the button aligned might cause some problems. Additionally, the text size might be too small to read for those with blurred vision. Therefore, considering many problems, I think it will be suitable for the game button to be centred. In addition, I have removed the character pixel image on the right due to the time. Having such limited time, I do not want to main focus the game art of the main screen, but on the in-game screen. Therefore, I decided to scrap the idea.

The buttons do not include any content such as: ‘Start Game’, ‘Controls’ and ‘Quit’ because I am going to be implementing the buttons using the Pygame module. This will make the game easier to create since I do not have to waste time creating art for the buttons and the different colour changes once clicked.

#### [Controls]

Justification: The control panel looks like this. The interface for the prototype has remained the same. The return button is the only component of the interface that is missing. As previously indicated, I will implement the button using the pygame module to save time on creating it for the art. The controls interface is complete and shows the controls for the user to learn besides that.

#### [In-game Screen]

Justification: This is the final design of the background of the game. This is the game's background's final layout. The player will be the subject of a camera that will zoom in and focus. This prevents from the player seeing the entire map and makes it more able to see the sprites. Two sides from various parts of the world are colliding within the map. My game's premise is for the player to prevent the world from clashing. Thus, get the name World Collides. Given that the player's goal is to save the world, this will immerse them in the game, which will help them relax because their attention will be diverted from the stressful stimuli they were experiencing.

## Usability Features

Usability features makes the game easier to understand. If the game is complex and the player is confused on what to do, it could cause stress. As previously mentioned, my game is about relieving stress. I do not want players to feel stress just when entering the game. For that reason, I am going to make the game easy to learn. When the player logs back into the game for the second time, they should be familiarised with the layout of the game. They will easily memories the controls, user interface and how to navigate through it because including images and visual aspects will make the player understand the criterions and purpose of the game. An example is the controls interface. If I decided to add text on what the controls are rather than highlighting the controls through images, it could make the controls less memorisable and inefficient. However, highlighting the left click on a visual mouse and the keys for the keyboard, will make it facile to remember and efficient for the player to learn.

The game’s plot is basic. The player has to defeat swarms of enemies without dying. Therefore, I think the player will not have difficulties when playing the game. In addition, when thinking about the long-term, players will see game play, reviews and play throughs of the game to get the general gist of the game. The use of tutorials will not be included since the game is a simple concept. I will reiterate which was under the research heading, ‘*80% of games contained combat*’, this tells me that players will not have any confusion when it comes to combat games. This is because I am assuming that the player has knowledge from other games which contain combat. I think for those whose first time it is playing a combat game, the difficulty of remembering the controls will not affect them, since the controls are straightforward to learn. I think when it comes to playing the game, the important thing is for the player to play the game. Combat is not a rare game style so; I think players just playing will give them a simple understanding of what the game is. The controls are not too hard to remember, and it only matters whether the player tests out the game.

The use of colour is an important feature to add. Adding colour makes the program more visually appealing to users. But the reason colour is added is to aid the player to distinguish objects between each other. If the character and the background were the same colour, it will be impossible for the player to differentiate between the background and character. By including colour, players use less effort when playing the game as they do not have to squint their eyes to process the different accepts of the game.

## Maintainability

Maintainability ensures that my program is readable and understandable to other programmers and myself. By adding these coding conventions, the program can be maintained since if I decided to return back to the project in the future, I can tell where I left off and begin programming. Additionally, if another programmer decided to continue my project, they will easily understand each individual part of the code. This means that they can either modify, improve or debug the project without breaking the program since they understand what each class and function does.

### Coding conventions

* **Comments:** The use of comments ensures that others and myself understand each line of code. This is because comments are written in plain English so, it is a brief explanation of what the code is. That means even non-programmers will understand what each line is doing. By adopting comments into my program, everyone will simply understand each line of code as it will be represented in a clarified explanation.
* **Object Orientated Programming:** When programming my project, I am going to utilise classes. This makes the program modular-based, making it easier to modify and edit the code when necessary. It also makes it easier when including methods, for me to call certain variables more easily. Additionally, I intend to inherit certain classes, which helps reduce memory used due to the re-usability of the same code. This will be done when creating different sprites.
* **Constant naming conventions (i.e., variables, classes, functions):** When it comes to naming my variables, classes, functions, etc, they must be constant. What I mean by this is having names which make sense to developers. If a future programmer or I decide to continue the project in the future, we must be able to understand what each name is about. This is because having them randomly named will cause confusion about what each line is about. Variables will be named like this: firstName, where the first word is not capitalised and the following words are. For classes, the first letter is going to be capitalised: Button. Finally, functions are all going to be lowercase, where if it is two-worded, it will contain an underscore (\_) to separate them: sprite\_sheet.

## Iteration Plans

|  |  |  |
| --- | --- | --- |
| Iteration Number | Iteration Name | What it will include |
| 1 | Iteration: Framework | This is the main iteration where I will add the basic framework of the game: It will consist of the main menu with buttons, a controls interface and the allows the user to quit the game. |
| 2 | Iteration: Display and move sprites, add camera | In the second iteration, I will add the sprites and assign the movement keys for the player to move around the map. There is going to be a camera that will be following the player’s movement. |
|  |  | Also, I will be adding a pause menu for when the player wants to stop for a brief moment of time. |

# Development

## Class Diagram

## Software Testing Methodology

### Black Box testing

When running my code, I am going to see if the inputs register the expected outputs. I am going to be checking the functionality of the program without looking at the code itself. What I will check are:

* Whether the movement inputs register to the corresponding response on the screen. ‘W’/’SPACE’ will jump, ‘A’ moves left, ‘D’ moves right and the left click will attack.
* Checks if clicking the buttons on the different menus sends the player to the correct page/ do the intended functions
* The pause menu opens when the ‘ECS’ keys are pressed
* Game over will display when the player loses

### White Box testing

I am going to be looking at the code itself. I should be able to trace through the code with ease by looking at it with great detail and understanding how the code will work. For that reason, I am using object-orientated programming (OOP) which will help the code function more smoothly and not be messy. As modifications can be made rapidly, when necessary, it will increase flow and flexibility.

### Alpha Testing

This is testing that will be done by me. Throughout creating the game, I am going to be continuously testing to ensure that the intended functions and gameplay are correct and working. These will be based on my success criteria in my analysis section, which I would follow religiously so that the game can be on track and completed before the end of the deadline. Also, it ensures that I can finish the program as I have proposed and avoid any bugs and logic/syntax errors within the code.

### Beta Testing

This testing is given to end users to test and give feedback. One type of user that will be running the program is my stakeholders. Since they are my target audience as well as my stakeholders, it allows them to gain first-hand experience with the program and check if there are any problems, which they are not happy with. Since my stakeholder is my target audience, this is a perfect chance to see if the program's intentions of relieving stress have been fulfilled. The valuable feedback given by the beta testers helps me reduce any bugs I have not spotted or any changes to the interface/code itself. For my stakeholder, Ray and Jahangir, opinions on the game will give me greater insight for improvement as their quick feedback will improve the quality and user experience for the game.

I am going to be using the Pygame documents[[6]](#footnote-6). This contains all the functionality and references which will help me develop my game.

## Iteration 1: Framework

### Aim

The aim of this iteration is to start up the framework of the program. Creating the framework is a basic building block of code to help me achieve the construction for the rest of the program. Having the base layout will make it easier for me to code my game as it saves time when creating the other iterations. Therefore, I am going to display the menu screen and the different buttons. When the buttons are clicked, they will warp the player to their respective destinations. When the ‘play game’ is clicked, it will display a blank screen and the character sprite. When the player inputs the specific keys required for movement, the sprite on the screen should move to with the corresponding actions. The black screen is a placeholder map as it will be done at a later iteration when I develop the game. The main success criteria for this iteration are:

1. Display the menu screen with ‘play’, ‘controls’ and ‘quit’ options so that players have a choice of what they want to do when entering the game
2. When the ‘play’ button is pressed, it should allow the player to play the game
3. When the ‘controls’ button is pressed, it should display the controls
4. When the ‘quit’ button is pressed, it should quit the game
5. Display the map when entering the game, which shows that the user has entered the game

### Key variables

|  |  |  |
| --- | --- | --- |
| **Name** | **Data Type** | **Justification** |
| Game | class | This is the main class which will contain different methods to run the entire program and access various windows |
| WIDTH | constant | A key that is constant so no one can change the width of the window |
| HEIGHT | constant | A key that is constant so no on one can change the height of the screen |
| FPS | constant | A key that sets the games frames per second. This prevents the game from losing frames. |
| event | function | The function allows the game to run and quit. It a continuous loop to play the game, until QUIT is met |
| Button | class | This is a button class that load buttons which the user can interacts with |
| clicked | function | Contains the parameters ‘pos’ and ‘pressed’, which register with the program if the mouse left click has been pressed and what position the mouse is at. |
| main\_menu | function | A menu function which displays the main menu and transfers the player into different interfaces |
| control\_screen | function | Displays the control interface when the controls button is clicked |
| draw | function | Creates the interface, which shows all the interface for the game |
| new\_game | function | Creates a new game when starting the game, which displays all the sprite and backgrounds necessary |
| return\_button | function | A function which calls the return button for the player to return back to the main menu screen. |

### Pseudocode

Within the pseudocode I will be maintaining it by adding comments. I think adding comments to pseudocode is really important. This is because it will help me and other programmers understand what each class, function and variable will do. Therefore, I will be including comments using // in green

NOTE: Since this is pseudocode, I will be placing 0s as placeholders since I do not have a general gist of where each image, button and hitbox are within the game interface. There is going to be a lot of trial and error; therefore, I will be placing 0s instead of the actual positions, as it will be hard to guess without having the actual interface to tamper with. E.g., self.screen.blit(self.rightScreen, (0,0)

**//main.py file**

//Import necessary libraries

import pygame

import sys

import random

from sprites import \* //Imports every content from the sprite.py file to the main.py file

from settings import \* //Imports every content from the settings.py file to the main.py file

class Game():

//Initialises the game so it can run and display necessary sprites and backgrounds. All the methods will contain self, where I am able to recall variables with self to other methods

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

pygame.inti() //initialises pygame in order to use

self.screen = pygame.display.set\_mode((WIDTH,HEIGHT)) //Sets the window screen size to WIDTH and HEIGHT form setting file. The width and height of the screen is stored in a tuple. This is so that it does not change the screen size as tuple are unchangable

self.title = pygame.display.set\_captions(‘Worlds Collide’) //Adds a name to the window bar

self.clock = pygame.time.Clock() //This is the FPS (Frame per second) which synchronises the game. This tells us the number of images/pixels constantly being displayed in a single second

self.running = True //Set the running state to true, indicating that the program is running

//The convert\_alpha() fucntion matches the window screen. It does calculations which makes it easier for us to load the image without visual glitches

//Loads spritesheets

self.entity = pygame.image.load('Graphics/...').convert\_alpha() //Opens the character sprite sheet

self.enemySprite = pygame.image.load('Graphics/...).convert\_alpha() // //Opens the enemy sprite sheet

self.surface = SpriteSheet('Graphics/Surface.png')

//Loads the image background of the game

self.background = pygame.image.load(‘Graphics/Game Screen.png’)

self.leftScreen = pygame.image.load(‘Graphics/ Left Screen.png’)

self.rightScreen = pygame.image.load(‘Graphics/ Right Screen.png’)

self.trainStation = pygame.image.load(‘Graphics/Train Station.png’)

self.pyramid = pygame.image.load(‘Graphics/Pyramid.png’).convert\_alpha() //.convert\_alpha() changes the pixel format in order to not lose performance

//Moving Images

self.train = pygame.image.load(‘Graphics/Train.png’).convert\_alpha()

self.train\_rect = self.train.get\_rect(topleft = (0,0)) //Creates a rectangle of the train image, where the top left of the train image at coordinate 860, 530 and move the train left of the screen.

//Menus

self.mainMenu = pygame.image.load(‘Graphics/Main Menu.png’)

self.control\_GUI = pygame.image.load(‘Graphics/Controls.png’).convert\_alpha()

//self.pauseMenu = pygame.image.load(‘Graphics/’)

//self.gameOver = pygame.image.load(‘Graphics/’)

//Method which starts a new game

def new\_game(self):

//Starts a new game

self.playing = True //If the player quit, then it should destroy the interface

//Creates the sprites layers and stores the intended sprites within the pack. Therefore, it allows easier recall of that sprite when needed

self.all\_sprites = pygame.sprite.LayeredUpdates() //An object containing all the sprites in the game making it easier to update the game

//A function that run the entire game

def event(self):

for event in event.pygame.get(): //gets the event from the Pygame module

if event.type == pygame.QUIT: //if the event type is quit, then the game will stop running and the game window will close

self.running = False //Sets the running state which was true to fasle to indicate the game has stopped running

self.playing = False //Sets the playing state to false, indication that all the sprites will stop loading. The new\_game method will stop functioning.

//A function that updates the game

def update(self):

self.screen.fill(BLACK) //Set the background colour to black to help sprites and background fill

self.all\_sprites.draw(self.screen) //Gets all the sprites from the all\_sprite group and draws it on the screen

self.clock.tick(FPS) //Sets the FPS of the game back to 60

pygame.display.update() //Updates the display of the game to the latest

def draw(self): //A function that draws everything from the screen

self.screen.fill(BLACK) //Fills the screen background with black

self.all\_sprites.draw(self.screen) //Draws all the sprites from the all\_sprites group to the screen

self.game\_screen() //Calls the game\_screen function

self.controls\_screen() //Calls the controls\_screen function

self.clock.tick(FPS) //Sets the FPS of the game to 60

pygame.display.update() //Updates the display

//Methods which displays the content of the game screen

def game\_screen(self):

self.screen.blit(self.background, (0,0)) //Draws the background of at the 0, 0 coordinated

self.screen.blit (self.rightScreen, (0, 0))

self.train\_rect.x -= 2 //Moves the rectangle of the train to the left of the x coordinate

if self.train\_rect.right < 100: //Once the trains rect reaches the 100 x coordinate

self.train\_rect.left = 1000 //It should move bck to the 1000 x coordinate

self.screen.blit(self.train, self.train\_rect)

self.screen.blit(self.leftScreen, (0,0))

self.screen.blit(self.trainStation, (0,0))

self.screen.blit(self.pyramid, (0,0))

//A function that keeps on looping the game while the game is running

def loop(self):

while self.playing: //loops through code while self.playing is true

self.event() //Calls the event function

self.update() //Calls the update function

self.draw() //Calls the draw function

//The main\_menu function

def main\_menu(self):

main = True //Variable set to true to indicate that the game is running

playGameButton = Button('Start Game', 0, 0, 0, 0, WHITE, LIGHT\_GREEN, 0) //The playGameButton variable is set to the Button class, where the parameters are in use

controlsButton = Button('Controls', 0, 0, 0, 0, WHITE, YELLOW, 0) //The controlsButton variable is set to the Button class, where the parameters are in use

quitButton = Button('Quit', 0, 0, 0, 0, WHITE, RED, 50) //The quitButton variable is set to the Button class, where the parameters are in use

self.returnButton = Button('Return', 0, 0, 0, 0, WHITE, BLACK, 0) //A return button which allows the user to return back to the main menu screen

while main: //While main is set to True, it will keep on looping

for event in pygame.event.get():

if event.type == pygame.QUIT:

main = False

self.running = False

//Gets the mouse and from the sprties.py, it gets the mouse position and pressed

mouse\_pos = pygame.mouse.get\_pos() //Gets the mouse position

mouse\_pressed = pygame.mouse.get\_pressed() //Checks if the mouse has been pressed

if playGameButton.clicked(mouse\_pos, mouse\_pressed): //Checks whether the mouse position is on the play game button and is the mouse clicks on the play game button

playGameButton = Button(‘Start Game’, 0, 0, 0, 0, WHITE, GREY, 0) //Changes the button to grey once clicked

main = False //main set to false tells us the main menu should close

if controlsButton.clicked(mouse\_pos, mouse\_pressed): //Checks whether the mouse position is on the controls button and is the mouse clicks on the controls button

controlsButton = Button('Controls', 0, 0, 0, 0, WHITE, GREY, 0) //Changes the button to grey once clicked

main = False

if quitButton.clicked(mouse\_pos, mouse\_pressed):

quitButton = Button('Quit', 0, 0, 0, 0, WHITE, GREY, 0)//Changes the button to grey once clicked

self.playing = False

self.running = False

main = False

//Draws images blit is a function from the Pygame module the draws images to the screen

self.screen.blit(sself.mainMenu, (0,0))

//Draws the buttons of the game and their hitbox

self.screen.blit(self.playGameButton.image, playGameButton.rect) //Draws the play button image and hitbox

self.screen.blit(self.controlButton.image, controlButton.rect) //Draws the control button image and hitbox

self.scren.blit(quitButton.image, quitButton.rect) //Draws the quit buttom image and hixbox

self.clock.tick(FPS)

pygame.display.update()

//Converting class into object code

game = Game() //Creates the game object

game.new\_game()

game.main\_menu()

while game.running:

game.loop()

//Stops the game from running

pygame.quit

sys.exit

**//sprite.py file**

//Importing libraries

import pygame

from settings import \* //Imports every content from the settings.py file to the sprite.py file

import math

import random

//This is a class names SpriteSheet which will helps load in the sprite sheet from a file

class SpriteSheet:

//A function that will initialise the sprites.py containing the parameters self and filename for the file

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

self.filename = filename

self.sheet = pygame.image.load(filename).convert() //Loading in the sprite sheet files

//A function that gets the sprites from the sprite sheet

def sprite\_get():

sprite = pygame.Surface([width, height]) //On the surface of the sprite, it gets the height and the width

sprite.blit(self.sheet, (0,0), (x , y, width, height)) //Draws the sprite on the position 0,0 and gets the x and y coordinates and the width and height of that position

sprite.set\_colorkey(BLACK) //Sets the sprites colour key (background) to black

return sprite //returns the sprite to draw on the screen

//A class: Contains the no parameters - extracts files from their location and allows them to be used

class SpriteSheet:

//A \_\_init\_\_ (initialise) method: Containing the parameters self, filename - sets the variables

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

self.filename = filename

self.sheet = pygame.image.load(self.filename).convert\_alpha() //Loads the game faster by loading the files quickly

//Sprite\_get method: Contains the parameters self, x, y, width, height - Cuts out individual sprites from the spritesheet and gets the corresponding sprite

def sprite\_get(self, x, y, width, height):

sprite = pygame.Surface([width, height]) //Creating the surface of the sprites with the width and height

sprite.blit(self.sheet, (0, 0), (x, y, width, height)) //The third parameter creates a cut out of the sprite sheet so that only the selected sections are used to make the sprite animated. Blit is a function to draw an image loaded in

sprite.set\_colorkey(BLACK) //Setting the sprite background colour to black

return sprite //Returns the sprite to draw on the screen

//Button class - no inheritance needed

class Button:

//\_\_init\_\_ (Initialises) method - Contains the parameters self, info, x, y, width, height, fg, bg, fontsize - Stores all the variables for the button

def \_\_init\_\_(self, info, x, y, width, height, fg, bg, fontsize):

self.font = pygame.font.Font('Pixeltype.ttf', fontsize) //Sets the font and font size for the game

self.x = x //Sets the x cords of the button

self.y = y //Sets the y cords of the button

self.width = width //Set the width of the button

self.height = height //Sets the height of the button

self.fg = fg //Sets the fore ground for the text

self.bg = bg //Sets the background colour for the button

self.info = info //Sets the information which contains the text in the button

self.image = pygame.Surface((self.width, self.height)) //Calls image variable which sets the width and height

self.image.fill(self.bg) //Fills the image rectangle with the background colour

self.rect = self.image.get\_rect() //Creates a rectangle around the image

self.rect.x = self.x //Creates a rectangle hitbox for the x coordinate

self.rect.y = self.y //Cretaes a rectangle hitbox for the y coordinate

self.text = self.font.render(self.info, True, self.fg) //Renders the font, which is stated at the self.font line and the content(info). The True indicates whether anti-aliasing should be on. This makes the font smoother

self.text\_rect = self.text.get\_rect(center = (self.width/2, self.height/2)) //Puts the text in the middle of the button, being exactly half the button size

self.image.blit(self.text, self.text\_rect) //Draws the image text and rect

//Clicked function -containing self, pos being the position of the mouse and pressed which checks if the mouse has been pressed

def clicked(self, pos, pressed):

if self.rect.collidepoint(pos): //Gets the position of the mouse and checks whether it is collding with the button

if pressed[0]: //Checks whether the left click (which is 0) is pressed

return True //If pressed returns True

return False //If not pressed return False

return False //If the rectangle of the mouse does not collide, then return false

**//settings.py file**

WIDTH = 1280 //A constant for width of the screen

HEIGHT = 720 //A constant for the height of screen

FPS = 60 //The frame rate per second

TILESIZE = 80 //The box of the SPRITE\_SIZE

//Speed

PLAYER\_SPEED = 5

//Colour (red, green, blue)

BLACK = (0, 0, 0)

WHITE = (225, 225, 225)

GREY = (188, 188, 188)

LIGHT\_GREEN = (19, 203, 118)

YELLOW = (203, 173, 19)

RED = (203, 19, 41)

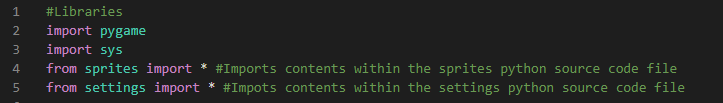
### Test plan table

|  |  |
| --- | --- |
| Test Description | Expected Results |
| Display the main menu screen | The main menu screen should be visible to the player when entering the game |
| Click on the ‘Play Game’ button | When clicking the button, it should display the screen to show that I am in the game |
| Click the ‘Controls’ button | When clicking this button, the game should display the controls button |
| Click the ‘Quit’ button | When clicking the button, it should quit the game by destroying the interface |
| Clicking ‘Return’ button on the control interface | When the player is on the controls interface, there will be a return button. When clicked it should return the user back to the main menu |

### Development Diary

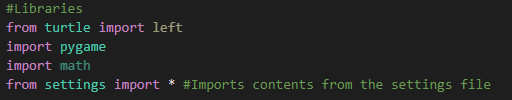
**Date: 4/11/2022**

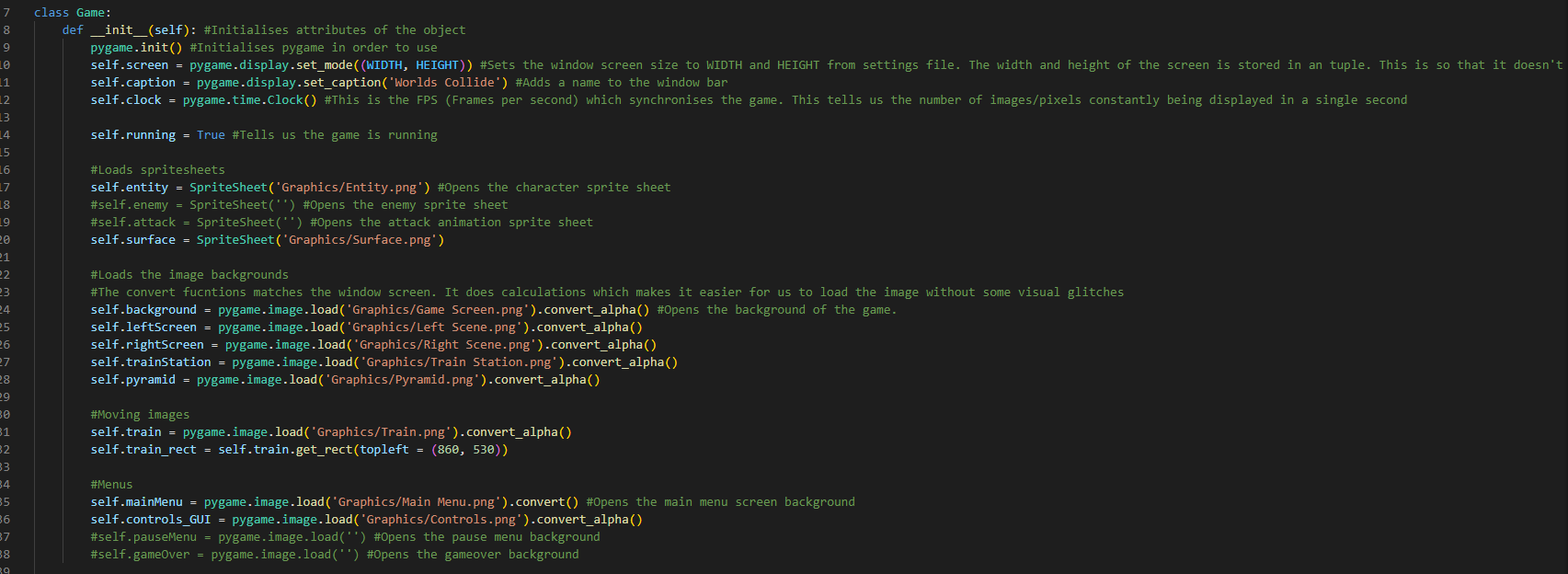
**Session Aim:** This is the first day of which I started programming. The first section of the program I would like to implement is the main menu screen. I want the various buttons on the main menu screen to perform the specific functions assigned. Firstly, I would like to implement the framework of the program so that the game window can open. When the window is opened, I am going to be displaying 3 buttons, which are 'start game’, ‘control’ and ‘quit’.

**Libraries**

For the first iteration, these are the libraries I am going to be importing. With these libraries, I can use the Pygame module, which this game depends on. Also, I will be using separate files to maintain my code more easily, therefore I will be importing all the contents within those .py files to the main.py file. This makes it easier for me to manage, change and handle my work flow.

This library is from the sprites.py file. This file contains different libraries from the main.py file.



**Game: Class**

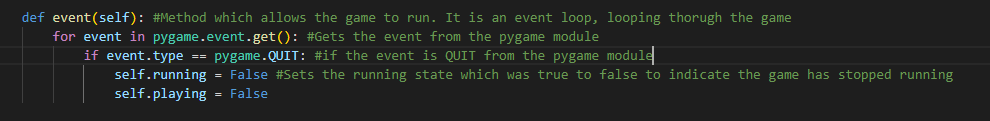
Here is the Game class, which will start off the game. It sets the display screen which will size all and display the interfaces. It uses a tuple meaning that the user is not able to change the size of the window. This is because I have not created the propriate art for the other window sizes due to the time constraint. There is also caption which sets the caption of the game name on the game bar, telling the players what they are playing. There is also a clock system. This synchronises the game to make the performance at the same rate. This makes the game smooth which makes the game easily to play without losing performance.

A running state is set to True, which tells the program that the game is running. If this state is set to False, this will stop the game from running.

The rest of the class in this iteration contains images and sprite sheets which are loaded into the game. This is the graphics of the game for the player to enjoy the game and leave a visually appealing look to the player's eye.

**Method: new\_game**

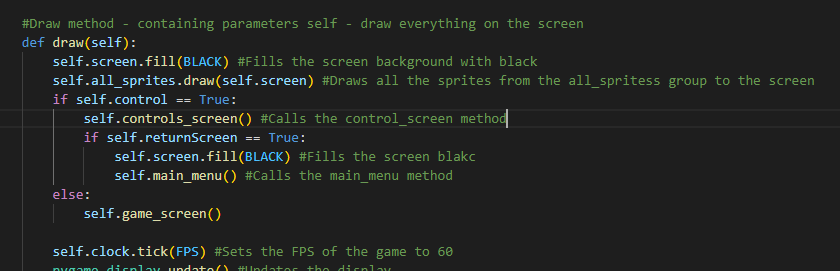
This is the new\_game method. This makes the game run since the playing state is set to True. If the playing state is set to False, it destroys the interface meaning the player has quit. After there is a layered sprite group. This is where all the sprites, borders, enemies and attack animation are stored in all sprite group. This makes it easier to call the program and update when required.

**Method: event**

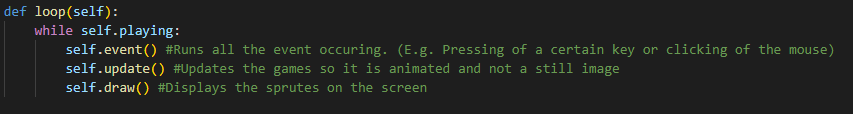
This is the event methods. This method makes the game run by looping continuously until a condition is met. This condition is the QUIT. By using pygame.event.get, which is part of the Pygame module, when QUIT is met, then self.playing and self.running is set to false. This tells the program that the game is no longer running which closes the windows.

**Method: update:**

This is the update method. This updates all the sprites within the game. It is also in the all\_sprites group which I defined in the new\_game method.

**Method: draw:**

The draw method is a function which draws all the screens, entities and all the screen gimmicks on to the screen. The first thing that the function does is that it fills the screen black. This is because rather than closing the interface and loading in a new one every time a new interface is called, the program draws the interface on top of each other.

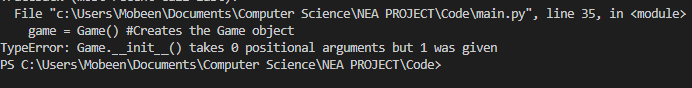
**Method: loop**

The loop method loops through all the other methods while the game is playing. It calls the event, update and draw method which makes it easier for the code to call at once rather than being all called separately. Having these methods within one method wastes less memory and more efficient use of code.

**Justification: Types of errors and solutions:**

When running the entire code for the first time, I came up with some syntax errors. These were mostly easy to fix.

**Error [TypeError]:** An error that occurred was that I forgot the to add self within the argument missing. This is a simple error to fix.

**Fix:** This is the fix to the [Type Error], self has been added to init (initialise) function.

**Error [AttributeError]:** Another error that occurred was an [Attribute Error]

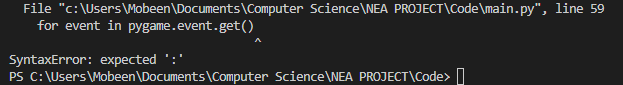
The game was not running since the loop() function was not within the while game.running:. This prevents the game from running since the loop() function contains the event(), update() and draw() functions which allows the game to run. This was an easy fix.

**Fix:**

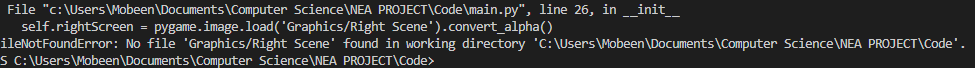


**Error [NameError]:** For this error, I forgot to add self before the screen. This caused pygames unable to understand since screen was not a variable as I used object-orientated code (OOP).

**Fix:** All I needed to do was add self, and the game was running properly again.

**Error [FileNotFoundError]:** Another error I encountered was forgetting to put .png at the end. Without putting .png, it causes the code unable to find the file I was looking for.

**Fix:** This was a simple fix.

**Error [SyntaxError]:** In this syntax error, I forgot to add the colons at the ends of the for event loop.

**Fix:** I just added colons at the end of this statement

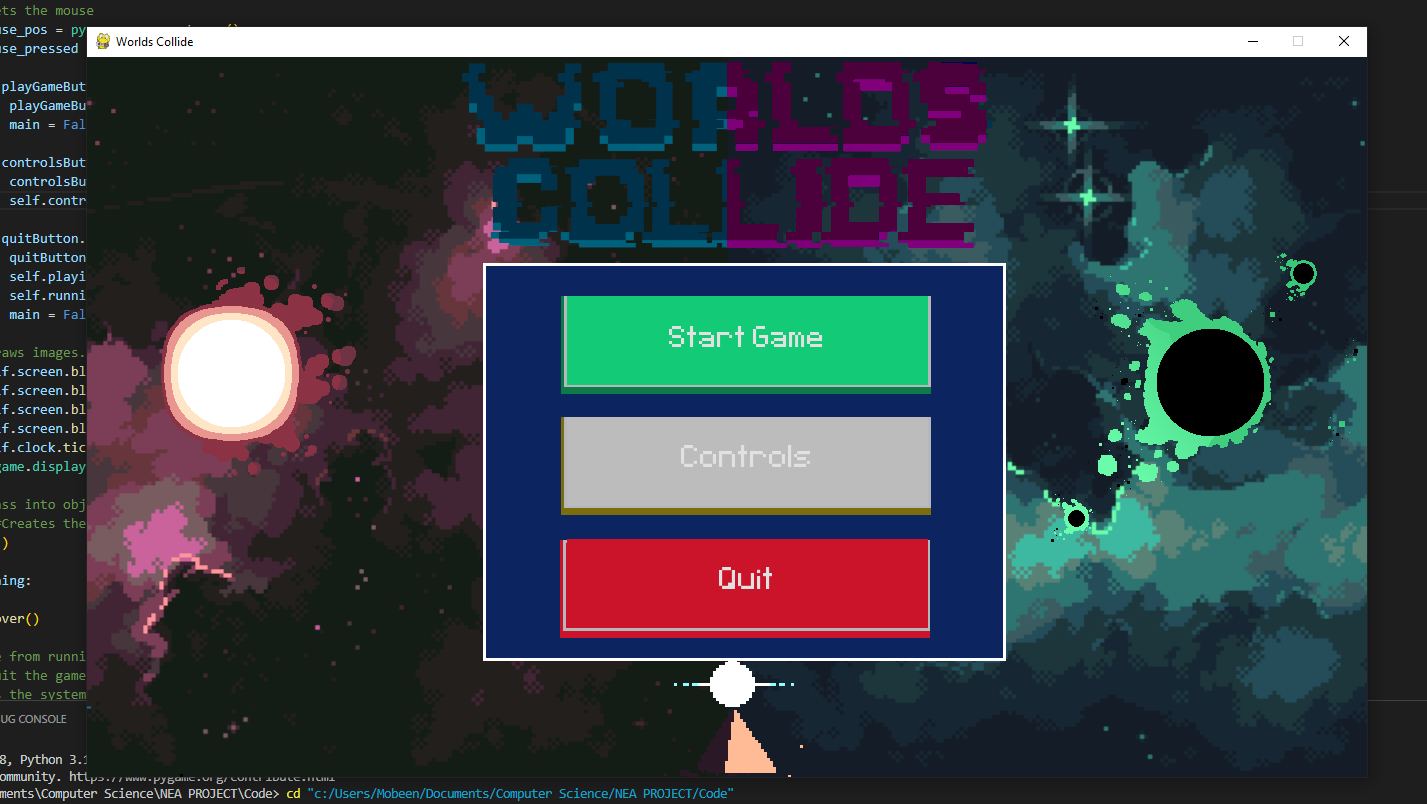
**Outcome:**

This is the final development of the code for today. After removing all the errors from above, the interface is fully complete. I have completed it by adding the interface of the main menu. So far in this iteration, I have made the main menu screen adding the background. I also made it so that the program is running and when the close button (‘x’) is pressed, it will destroy the window. Next session I would like to add the button functionality and finish the first iteration.

**Session Aim:** During this session, I would like to finish of the entire menu screen. The players should be able to return back to the main menu screen when on the controls interface. This would be done by clicking on the return button.

Within the sprites.py, I am going to be coding the buttons, so that when they are pressed, the designated interface will show up.

A problem I kept running into was the ‘start game’ and the ‘control’ buttons clashing with each other. When the 'control’ button was clicked, it kept on opening the in-game interface, which was an error that needed to be addressed.

The first idea that sprang into mind was to not set main = False, so that the program will understand that the main menu has not been left. Instead, I will introduce the ‘blit’ function (self.screen.blit(self.controls\_GUI, (0,0))) which will open the controls GUI rather than the in-game screen. However, when trying out the plan, whenever I clicked the ‘controls’ button it would freeze and there would be no respond other than the button turning grey (which is the intended colour once clicked).

For that reason, I scraped this idea and moved on to the next.

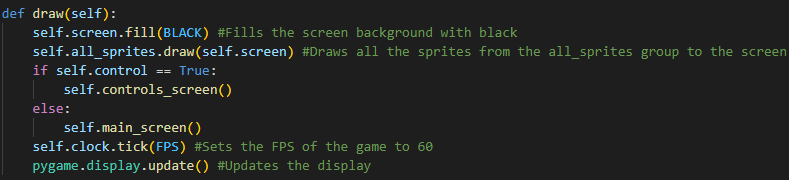
Another idea I had to solve this problem was to ‘blit’ the controls interface before the in-game screen so that the program will read the controls interface first. But this caused another problem where when ‘Start Game’ was pressed, then it would display the controls interface, which was not the intention.

This solution fixed the problem of the controls interface not showing. Firstly, in the main\_menu function, I added a new variable called self.control and set it too False.

Having the control set to false tells the program that the control interface has not been clicked. After this, when the ’controls’ button was clicked, I set the self.control to True. This will tell the program that the ’controls’ button was clicked. The reason I included self rather than keeping it control like ’main = True’ was because I wanted to call it again in another function. The main reason for the ’controls’ button and the ‘start game’ colliding with each other was due to draw function. When drawing the different interfaces, the draw function took priority over the in-game screen, which was the reason they clashed. Therefore, I added a few lines where when the ’controls’ button was clicked, it if self.control was True, then it would display the controls interface and if not then the in-game screen.

With adding this if statement, it resolved the problem and the main menu screen was completed.

Therefore, I made some changes to the pseudocode.

**Note:** The changes are as followed in the colour BLUE:

def draw(self):

self.screen.fill(BLACK) //Fills the screen background with black

self.all\_sprites.draw(self.screen) //Draws all the sprites from the all\_sprites group to the screen

if self.control == True:

self.control\_screen()

else:

self.game\_screen()

self.controls\_screen()

self.clock.tick(FPS) //Sets the FPS of the game to 60

pygame.display.update() //Updates the display

//New function added in order to stop the controls GUI and in-game GUI from clashing

def control\_screen(self):

self.screen.blit(self.control\_GUI, (0,0))

def main\_menu(self):

main = True //main is a variable set to true

self.control = False //control variable is set to false

playGameButton = Button('Start Game', 480, 239, 361, 89, WHITE, LIGHT\_GREEN, 49) //The playGameButton variable is set to the Button class, where the parameters are in use

controlsButton = Button('Controls', 480, 360, 361, 89, WHITE, YELLOW, 50) //The controlsButton variable is set to the Button class, where the parameters are in use

quitButton = Button('Quit', 480, 482, 361, 89, WHITE, RED, 50) //The quitButton variable is set to the Button class, where the parameters are in use

self.returnButton = Button('Return', 1150, 670, 105, 45, WHITE, BLACK, 45)

while main: //while main is set to True, it will keep on running

for event in pygame.event.get()

if event.type == pygame.QUIT:

main = False

self.running = False

//Gets the mouse and from the sprite.py, it gets the mouse position and pressed

mouse\_pos = pygame.mouse.get\_pos() //Gets the mouse position

mouse\_pressed = pygame.mouse.get\_pressed() //Checks if the mouse has been pressed

if playGameButton.clicked(mouse\_pos, mouse\_pressed): //Checks whether the mouse position is on the play game button and is the mouse clicks on the play button

playGameButton = Button(‘Start Game’,480,239,361,89,WHITE,GREY,49) //Changes the button to grey once clicked

main = False //main set to false tells us the main menu should close

if controlsButton.clicked(mouse\_pos, mouse\_pressed): //Checks whether the mouse position is on the controls button and is the mouse clicks on the control button

controlsButton = Button('Controls', 480, 360, 361, 89, WHITE, GREY, 50) //changes the button to grey once clicked

main = False

self.control = True

if quitButton.clicked(mouse\_pos, mouse\_pressed):

quitButton = Button('Quit', 480, 482, 361, 89, WHITE, GREY, 50) //Changes the button to grey once clicked

self.playing = False

self.running = False

main = False

//Draws images blit is a function from the Pygame module the draws images to the screen

self.screen.blit(self.mainMenu, (0,0))

self.screen.blit(self.playGameButton.image, playGaemButton.rect)

self.screen.blit(self.controlButton.image, controlButton.rect)

self.scren.blit(quitButton.image, quitButton.rect)

self.clock.tick(FPS)

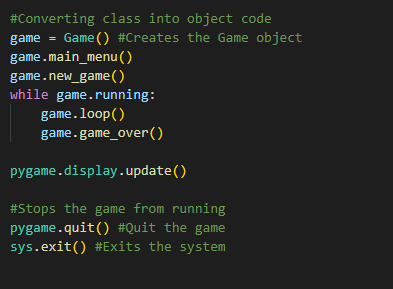
pygame.display.update() //When the menu screens stop, then the in-game screen appears

**Outcome:**



An issue that I kept of running into was the frame loss when the player clicks the ‘Start Game’ button. Whenever the player started the game, the sprites will lose frames for a few seconds. I knew about it when the train sprite moving from one side of the screen to the other was dropping frames when entering the game. At first, I thought about leaving the frame lose for a bit, however, this would cause problems when the character and enemy sprites load into the game. Having more sprites within the game will makes the performance even worse. Therefore, I decided to find the solution.

Within the code, I added: pygame.display.update() at the end of the code. This ensures that all the game display is updated and that the correct entities such as buttons are updated, which improves overall performance of the game.



When entering the controls interface, there was no way to return back to the main menu without closing the entire game. Therefore, I created a return button. However, there was an issue when implementing the return button. The problem was the main menu and control interfaces were clashing with each other. This meant that I could not return back to the main menu. Therefore, I decided rather than the user returning back to the main menu, there would be a direct click for the game to start. My thought process was that players usually checks the controls in order to understands the controls and then play the game. Therefore, rather than returning back to the main menu, the game would jump start them into the game.

### Result of testing plan

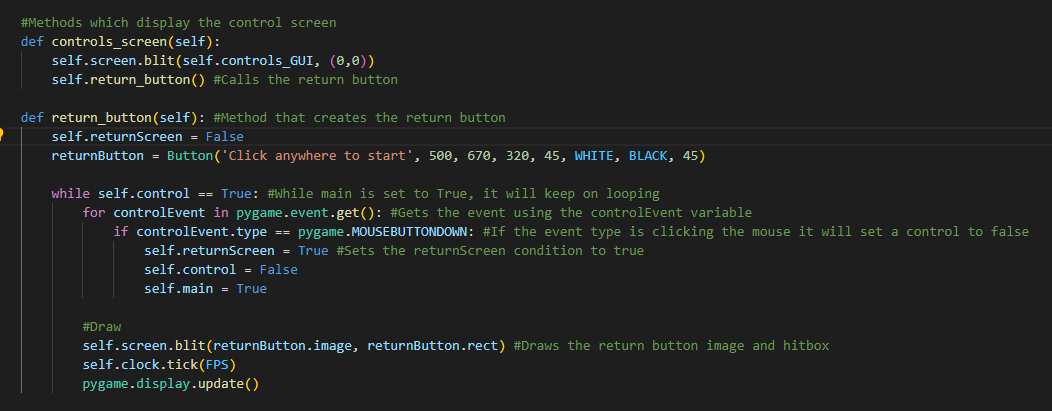
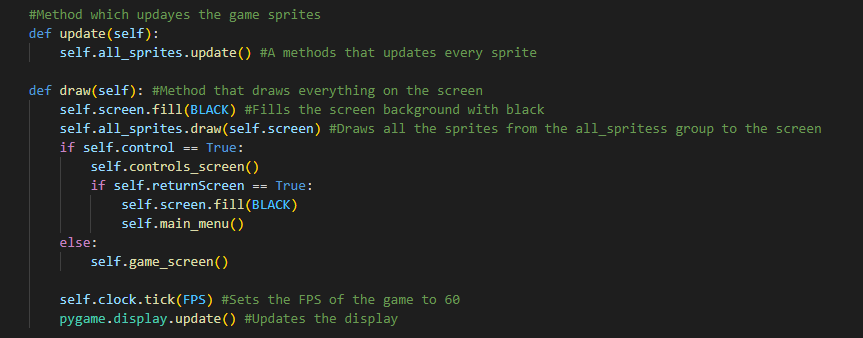
|  |  |  |  |
| --- | --- | --- | --- |
| Test Description | Expected Results | Type | Actual Results (Success/Fail) |
| Display the main menu screen | The main menu screen should be visible to the player when entering the game | Normal | Success |
| Click on the ‘Play Game’ button | When clicking the button, it should display the screen to show that I am in the game | Normal | Success |
| Click the ‘Controls’ button | When clicking this button, the game should display the controls button | Normal | Success |
| Click the ‘Quit’ button | When clicking the button, it should quit the game by destroying the interface | Normal | Success |
| Clicking ‘Return’ button on the control interface | When the player is on the controls interface, there will be a return button. When clicked it should return the user back to the main menu | Normal | Fail |

For the return button, I could not succeed on allowing the player to return back to the main menu screen when clicking on the return button. Firstly, for some reason when clicking the return button, there would be no response on the control screen. I tried to change the algorithms and put the return button in its own separate function, but I did not succeed.

Later I changed the code where rather than the player needing to click return button, they would have to click anywhere else on the screen to return back to the main menu. However, this has caused a problem where the different screen was clashing with each other. This is stated above. This was a problem because I could not figure out what the main logic error of the game. The game was working as intended, but for some reason, when clicking on the screen, it would always merge the screens together. Therefore, the final solution was when the player clicks on the screen, it would just start the game. So instead of having a return button, I have created a text on the screen ‘click anywhere to start’ which would take the player to the game. I think this makes sense because when players look at the controls, their intentions are to understand the controls and then enter the game. Therefore, having the player click anywhere on the controls interface to enter the game will make it more efficient.

**Date:** **14/11/2022**

When taking Jahangir’s feedback into consideration (in stockholder feedback), I realised that before calling self.main\_menu() to draw the menu interface onto the screen, I should fill the screen black and then draw the main menu in. This is because my game involves drawing the interface on top of each other. This makes it easier to code since rather than closing the interfaces, I draw the interfaces on top of each other. However, before the interfaces are drawn on top of each other, I use self.screen.fill(BLACK) to fill the screen black (within my draw method/function) and then add the other interface. This gives it an illusion that the other interfaces are closed and the current interface is opened alone.



**Final Outcome:**

Therefore, the player would click anywhere on the screen to return to the main menu rather than clicking anywhere on the screen to start the game.



|  |  |  |  |
| --- | --- | --- | --- |
| Test Description | Expected Results | Type | Actual Test (Success/Fail) |
| Clicking on the control screen to return back to the main menu | The player is able to click anywhere on the controls screen in order to return back to the main menu | Normal | Success |

However, I ran into another logic error. When the clicking on the controls screen to return back to the main menu, if the user hovered over the buttons from the main menu screen, then the code will register that the player clicked the button on the menu. This directs the user to either in-game screen, controls screen or quits. Obviously, this is not the intended function and is a bug. However, due to the time limit, I will find the solution to this bug once the entire code is complete.

### Stakeholder Feedbacks

I gave the first iteration to my stakeholders. Jahangir had no problems with the program and was fond of the design layout. However, one problem that Jahangir did not like was not including a return button once in the controls interface. I explained to him that I was not able to implement the return button as the screen kept on merging when the return button was clicked. However, during 14/11/2022, I fixed this problem.

Fortunately, I used agile methodologies because the use of stakeholder feedback has helped me increase my field of view of what to improve.

Ray did not have any issues with the program. This was mainly due to him testing out the program when I incorporated Jahangir’s feedback.

## Iteration 2: Sprites, animations, movement, camera

### Aim

The aim of this iteration is to add movement within the game. The player should be able to travel across the map by moving left, right, and jumping when they initially access the in-game interface. Additionally, the player should collide with the boundary that prevents them from leaving the game map. A player centred camera will also be included. As a result, the player will be the centre of attention, and the camera will follow the player's every action. Since it will be challenging for the player to see what is happening, the camera will zoom in on them rather than being set to the screen's size. This will improve usability because players will not have to strain their eyes to see the game. I'll implement the character sprite that will replace the white box once the movement and the camera are finished. I will be including the animations for all entities when I implement the sprites.

1. Create a camera which is centred on the player. When the player moves, the camera should follow them.
2. Display sprites for both player character and enemies, which shows that the game has begun
3. Allows the player to move left, right, jump and attack
4. Correspond the appropriate inputs with different actions. When the player either moves or attacks, then the game should respond to those action. E.g., if the player presses ‘A’ it should allow the user to move left
5. Add animation for the different actions performed by the player or enemy (e.g., attack, jump, death, etc). There will be a sprite sheet with all the different sprites that make up the animations
6. The entities should not be able to leave the map. For that reason, adding a border should prevent this
7. A combination of different actions should not collide with each other (e.g., attack and death). This is because having death as well as attacking will cause a problem since the entity should have died
8. Display the health bar for the player to know when they might lose
9. Reduce health of enemy or player when taking damage

These are the success criteria I would like to complete during this iteration. However, I would not be able to complete adding the enemy sprites (criteria 2) as I do not plan to add the enemies within this iteration. Therefore, I will be altering this criterion to only the player.

### Key variables

|  |  |  |
| --- | --- | --- |
| **Name** | **Data Type** | **Justification** |
| SpriteSheet | class | This is the sprite class. This class will load the image file and allow me to crop out the sprites from the sprite sheet. |
| Player | class | This is the player class and will load everything of the player: movement, sprite, collision, attack |
| Enemy | class | This is the enemy class that will store and load all the functions of the AIs. |
| movement | function | This is the movement function that will contain the controls for the player to move |
| Sprite\_animation | function | This function calls the ANIMATION list of the entities and draws them onto the screen |
| animation | function | This function runs the animation, where a certain action has a corresponding animation |
| action\_update | function | It prevents old animation from clashing from the new animation by checking if a new action has been done |
| update | function | Updates the entire sprite, so that they can move |
| attack | function | This allows the player to attack by adding an invisible attack hitbox |
| health\_bar | function | A function that draws the health bar of the player. It alerts the player that their health is dwindling |
| CHARACTER\_DATA | list | This is a list that contains all of the characters attributes: sprite size, scale. |
| ENEMY\_DATA | list | This is a list that contains all of the enemy’s attributes: sprite size, scale, offset |
| CHARACTER\_ANIMATION | list | A list that contains all of the characters animation, where each index is called depending on the animation done |
| ENEMY\_ANIMATION | list | A list that contains all of the enemy’s animation, where each index is called depending on the animation done |

### Pseudocode

**//main.py file**

//Loads spritesheets

self.entity = pygame.image.load('Graphics/Entity.png').convert\_alpha() //Opens the character sprite sheet

self.enemySprite = pygame.image.load('Graphics/Enemy.png').convert\_alpha() //Opens the enemy sprite sheet

self.player = Player(605, 420, CHARACTER\_DATA, self.entity, CHARACTER\_ANIMATION) //Creates the player from the player class and loads the character data. Spawns the player at the middle of the screen.

x\_spawn = random.randint(0, WIDTH - 20) //A random function which selects any part of the map for the enemy to spawn

self.enemy = Enemy(x\_spawn, 0, ENEMY\_DATA, self.enemySprite, ENEMY\_ANIMATION) //Creates the enemy form the enemy class and loads the enemy data

//event method - It is an event loop, looping through the game. This allows the game to run continuously until a condition is met

def event(self):

...

self.player.animation() //Calls the animation of the player

self.enemy.animation() //Calls the animation of the enemy

//game\_screen method - display the content of the in-game screen

def game\_screen(self):

...

self.health\_bar(self.player.health, 0, 0) //Draws the players health bar at coordinates 0, 0

//self.health\_bar(self.enemy.health, 0, 0)

//Draws and moves the enemy

self.enemy.draw(self.screen) //Draws the enemy on the screen

self.enemy.movement(self.player) //Allows the enemy to move, where the player is the target

//Draws and allows the player to move

self.player.draw(self.screen) //Draws the player on the screen

self.player.movement(WIDTH, HEIGHT, self.screen, self.enemy) //Allows the player to move, and do certain action by calling the movement method

input = pygame.key.get\_pressed() //Sets input variable to a pygame function to check is a key is pressed

if input[pygame.K\_ESCAPE]: //Checks if the key pressed is ESCAPE

self.pause\_screen()

//health\_bar method: containing the parameters self, health, x and y - it creates the entity health for both player and enemies.

def health\_bar(self, health, x, y):

health\_ratio = health / 200 //A ratio, where the health is divides by 200. This makes it easier to see the visuals of the player losing health

pygame.draw.rect(self.screen, BLACK, (x - 4, y - 4, 0, 0)) //Draws a black outline of the health bar

pygame.draw.rect(self.screen, WHITE, (x, y, 0, 0)) //Draws the health bar

if health >= 150: //Checks if health is greater than 150

pygame.draw.rect(self.screen, LIGHT\_GREEN, (x, y, 400 \* health\_ratio, 0)) //Draws a green health bar

elif health >= 100: //Checks if health is greater than 100

pygame.draw.rect(self.screen, YELLOW, (x, y, 400 \* health\_ratio, 0)) //Draws an orange health bar

elif health >= 1: //Checks if health is greater than 1

pygame.draw.rect(self.screen, RED, (x, y, 400 \* health\_ratio, 0)) //Draws a red health bard

else: //If neither condition is met

pygame.draw.rect(self.screen, WHITE, (x, y, 0, 0)) //Draws the original white health bar

**//sprites.py file**

//A class: containing no parameter - It creates the player

class Player:

//A \_\_init\_\_ (initialise) method: Contains the parameters: self, x, y, data, sprite\_sheet, animations - This sets the main variables that will be continuously used

def \_\_init\_\_(self, x, y, data, sprite\_sheet, animations):

self.x = x //Defines x

self.y = y //Defines y

self.x\_change = 0 //Sets the value of the player movement of the x coordinate to 0

self.y\_change = 0 //Sets the value of the player movement of the y coordinate to 0

self.velocity = 0 //Creating velocity and setting it to 0, which can change

self.gravity = 2.2 //Creating gravity and setting it 2.2

self.jump = False //Creating jump state and setting to false

self.attacking = False //The player attacking is set to condition False

self.attack\_cooldown = 0

self.facing = 'idle' //Creating a facing state, telling us the which way the player is facing. For now, it is set to idle

self.targetLock = False //Facing the enemy is set to false

self.run = False

//Gets the data from the CHARATER\_DATA in settings.py

self.size = data[0] //Gets the character size from position 0 from the list CHARACTER\_DATA

self.scale = data[1] //Gets the character scale from position 1 from the list CHARACTER\_DATA

self.animation\_list = self.sprite\_animation(sprite\_sheet, animations) //Makes a list of animation where it runs through the sprite sheet

self.action = 0 //The specific action done by the sprite: 0:idle, 1:Left/Right run, 2:Jump, 3:Attack

self.frame\_index = 0 //Sets a frame\_index to 0, indicating the entites animation frame in order to loop through the animation

self.image = self.animation\_list[self.action][self.frame\_index] //The image stores the animation list of the action and frame index

self.time = pygame.time.get\_ticks() //From the pygame module, where it sets a timer. This is used for the animation loop

self.rect = pygame.Rect((x, y, 80, 80)) //Drawing the image react for the player

self.health = 200 //Sets the health value to 200

self.hit = False //Sets hit as false

self.life = True //Sets the life to true

//Update method which updates the other functions

def update(self):

self.movement() //Calls movement method

//sprite\_animation method: containing the sprite\_sheet and animation method - This method draws the animation from the list on to the surface of the screen

def sprite\_animation(self, sprite\_sheet, animations):

animation\_list = [] //The main list of the animations

for y, animation in enumerate(animations): //A nested for loop that goes down the sprite sheet columns to load the sprites

sprite\_list = [] //A list containing the sprites

//This loop goes through the rows of the sprite sheet

for x in range(animation): //A loop that goes through the animation of the rows

sprite = sprite\_sheet.subsurface(x \* self.size, y \* self.size, self.size, self.size) //Gets the single image for the sprite

sprite\_list.append(sprite) //Appends sprites onto the sprite list

animation\_list.append(sprite\_list) //Appends the sprite list into the animation list

return animation\_list //Returns the animation list

//animation method - This method run the animation of the player during certain actions

def animation(self):

if self.life == True: //Checks if the player is alive

if self.run == True: //Checks if the run state is true

self.action\_update(1) //Updates the action to 1

elif self.jump == True: //Checks if the player is jumping

self.action\_update(2) //Does the action of index 2

elif self.attacking == True: //Checks if attacking is true

self.action\_update(3) //Does the action of index 3

elif self.hit == True: //Checks if the player has been hit

self.action\_update(4) //Does action 4

if self.health <= 0: //Checks if the players health is less than or equal to 0

self.health = 0 //Sets the health to 0

self.life = False //Sets the life to false as the player has died

self.collide = False //Sets collide to false, so the player is not colliding with the enemy

self.action\_update(5)

elif self.health <= 0: //Checks if the players health is less than or equal to 0

self.health = 0 //Sets the health to 0

self.life = False //Sets the life to false

self.action\_update(5) //Does action 5

else:

self.action\_update(0) //Updates action 0

else:

self.hit = False //Sets hit to false as the player is no longer hit

self.collide = False //Sets the collide to false as the player is no longer is colliding

cooldown = 100 //A cooldown timer set to 150

self.image = self.animation\_list[self.action][self.frame\_index] //Sets the image to the animation list with the action and frame index

if pygame.time.get\_ticks() - self.time > cooldown: //A tick system where it checks if the tick subtracted by time is greater than the cooldown

self.frame\_index += 1 //Increments the frame index by 1

self.time = pygame.time.get\_ticks() //Sets the time to tick

if self.frame\_index >= len(self.animation\_list[self.action]): //Checks if the frame index is greater or equal to the length of the animation list and action

if self.life == False: //Checks if the player is alive

self.frame\_index = len(self.animation\_list[self.action]) - 1 //Sets the frame index to the length of the animation list and the action

else:

self.frame\_index = 0 //Sets the frame index back to 0

if self.action == 3: //Checks if the action is attack

self.attacking = False //Sets attacking false

self.attack\_cooldown = 10 //Sets the cooldown to 10

if self.action == 4: //Checks if the action is 4

self.hit = False //Sets hit false

self.attacking = False //Sets attacking false

self.attack\_cooldown = 10 //Sets the cooldown back to 10

//movement method: containing the parameters self, width, height, surface and enemies - dedicated for all the movement function of the game.

def movement(self, width, height, surface, enemies):

input = pygame.key.get\_pressed() //input is assinged to the pygame module, where it checks if keyboard keys are pressed

self.run = False

//Prevent other actions from colliding from each other

if self.attacking == False and self.life == True:

if input[pygame.K\_a]: //Checks if the input is 'a' key

self.x\_change = -PLAYER\_SPEED //The player movement of the x coordinate is subtracted by the player speed, making the sprite move left

self.run = True

self.facing = 'left' //Tells the code the sprite is facing left

else:

self.x\_change = 0 //This stops the player from continuously moving, so it is set to 0

if input[pygame.K\_d]: //Checks if the input is the 'd' key

self.x\_change = +PLAYER\_SPEED //The player movement of the x coordinate is added by player speed, making it move right

self.run = True

self.facing = 'right' //Tells the code the sprite is facing right

if input[pygame.K\_p]: //Checks if the key pressed is p

self.attacks(surface, enemies) //Attacks the enemies on the screen

if input[pygame.K\_w] and self.jump == False: //Checks if the input is the 'w' key and if the the self.jump condition is false

self.velocity = -10 //Changes the velocity from 0 to -10

self.facing = 'up' //Tells the code that the sprite is facing up

self.jump = True //When 'w' keys is pressed, changes the jump state to true

//The same setup as the 'w' key, but for SPACE

if input[pygame.K\_SPACE] and self.jump == False:

self.velocity = -11

self.facing = 'up'

self.jump = True

self.velocity += self.gravity //Adds velocity to gravity, so that gravity can pull down the sprite back to the ground surface

self.y\_change += self.velocity //Adds the y change to velocity to tell the code the change in the y coordinate

self.rect.x += self.x\_change //Adds a rectangle to the player x change

self.rect.y += self.y\_change //Adds a rectangle to the player y change

//Checks for collision with border

if (self.rect.left + self.x\_change) < 0: //Checks whether the left rectangle added with the x change is less than 0

self.x\_change = 0 - self.rect.left //When the sprite's left rectangle side touches the border, it stops the player from moving off the screen

self.rect.left = 0 //Resetting the left rectangle to 0 to stop player going past border

if (self.rect.right + self.x\_change) > width: //Checks whether the right rectangle added with the x change is less than the screen width (1280)

self.x\_change = width - self.rect.right //When the sprite's right rectangle side touches the border, it stops the player from moving off the screen

self.rect.right = width //Resets the right rectangle back to width to prevent player going past the border

//Checks if the player is jumping

if (self.rect.bottom + self.y\_change) > height - 130: //Checks if the bottom of the player rect plus the change in y coordinate is greater than the height - 130(the ground level)

self.velocity = 0 //Sets the velocity back to 0

self.jump = False //Sets the jump condition back to false

self.y\_change = height – 130 - self.rect.bottom //Sets the y change to height subtracted by the 130 and the players bottom rect

//Player faces the enemy

if enemies.rect.centerx > self.rect.centerx: //Checks is the enemy x center rect is greater than the player center x rect

self.targetLock = False //If true, player is facing right of the enemy

else:

self.targetLock = True //If false, player is facing left of the enemy

if self.attack\_cooldown > 0: //Check if the cooldown is greater than 0

self.attack\_cooldown -= 1 //Subtracts the cooldown by 1

//Attack method: Contains self, surface and enemies as parameters. Allows the player to attack the enemy

def attacks(self, surface, enemies):

if self.attack\_cooldown == 0:

self.attacking = True //Sets the self.attacking to true

self.attack\_rect = pygame.Rect(self.rect.centerx - (self.rect.width \* 0.3 \* self.targetLock), self.rect.y, self.rect.width \* 0.8, self.rect.height) //Sets the attack rectangle, where the attack\_rect is at the center of the player rect. This is subtracted by times 0.5 width and target lock to flip the sprite of where the enemies are. The self.rect.y is then called and self.width is multiplied by 0.3 to create the width of the attack\_rect. The height is also called

if self.attack\_rect.colliderect(enemies.rect): //Checks whether the self.attack\_rect is colliding with the enemy rect

enemies.hit = True

enemies.health -= 20 //Reduces the enemy health by 20

//action\_update method: Containing the self and new\_action parameters - It checks whether there is a new action done

def action\_update(self, new\_action):

self.new\_action = new\_action //Defines new\_action as self

if self.new\_action != self.action: //Checks if new\_action is not equal to action

self.action = self.new\_action //action is set to the new\_action

self.frame\_index = 0 //Sets the frame index to 0

self.time = pygame.time.get\_ticks() //Resets the timer

//draw method: Containing the parameters self and surface - Draws the player on the screen.

def draw(self, surface):

playerLock = pygame.transform.flip(self.image, True, False) //This flips the player (self.image) dependng on the way the player is facing. True indicates the player is flipped and false stops the player and False preventing the sprite flipping up or down

if self.facing == 'left':

surface.blit(playerLock, (self.rect.x, self.rect.y))

else:

surface.blit(self.image, (self.rect.x, self.rect.y)) //Draws the playerLock with the rect x and y on the surface

//Enemy Class - no inheritance

class Enemy:

//A \_\_init\_\_ (initialise) method: Contains the parameters: self, x, y, data, sprite\_sheet, animations - This sets the main variables that will be continuously used

def \_\_init\_\_(self, x, y, data, sprite\_sheet, animations):

self.x = x //Defines x as self

self.y = y //Defines y as self

self.x\_change = 0 //Sets the change in the x coordinate to 0

self.y\_change = 0 //Sets the change in the y coordinate to 0

self.targetLock = False //A condition set to false where it checks if the enemy is facing the player

self.run = False

//Gets the data from the ENEMY\_DATA in settings.py

self.size = data[0] //Gets the size of the enemy from index position 0

self.scale = data[1] //Gets the scale of the enemy from index position 1

self.offset = data[2] //Gets the offset of the enemy from the index position 2

self.animation\_list = self.sprite\_animation(sprite\_sheet, animations)

self.action = 0 //0:idle, 1:run\_right, 2:run\_left

self.frame\_index = 0 //Setting the frame index to 0

self.image = self.animation\_list[self.action][self.frame\_index] //Sets the image to the animation list where it contains the action and frame index

self.time = pygame.time.get\_ticks() //From the pygame module, where it sets a timer. This is used for the animation loop

self.rect = pygame.Rect((x, y, 60, 30)) //Drawing the image react for the enemy

self.collide = False

self.health = 100 //Sets enemy health to 0

self.hit = False

self.life = True

//sprite\_animation method: containing the sprite\_sheet and animation method - This method draws the animation from the list on to the surface of the screen

def sprite\_animation(self, sprite\_sheet, animations):

animation\_list = [] //The main list containing the main animations

for y, animation in enumerate(animations): //A nested for loop that goes down the sprite sheet columns to load the sprites

sprite\_list = [] //A list containing the sprites

//This loop goes through the rows of the sprite sheet

for x in range(animation): //A loop through the animation of the enemy

sprite = sprite\_sheet.subsurface(x \* self.size, y \* self.size, self.size, self.size) //Gets the single image for the sprite

sprite\_list.append(pygame.transform.scale(sprite, (self.size \* self.scale, self.size \* self.scale))) //Gets the single image from the sprite sheet

animation\_list.append(sprite\_list) //Appends the sprite list into the animation list

return animation\_list //Returns the animation list

//animation method - This method run the animation of the player during certain actions

def animation(self):

if self.life == True:

if self.run == True: //Checks if the run state is true

self.action\_update(1) //Updates the action to 1

elif self.collide == True: //Checks if attacking is true

self.action\_update(2) //Does the action of index 3

elif self.health <= 0: //Checks if the enemy health is less than or equal to 0

self.health = 0 //Sets the health to 0

self.hit = False //Sets hit false

self.life = False //Sets the life to false

self.action\_update(4) //The enemy does action 4

elif self.hit == True: //Checks if the enemy has been hit

self.action\_update(3) //Does action 3

else :

self.action\_update(0) //Updates the action to 0

cooldown = 100 //A cooldown timer which countown from 100

self.image = self.animation\_list[self.action][self.frame\_index] //Sets the image to the animation list with the action and frame index

if pygame.time.get\_ticks() - self.time > cooldown: //A tick system where it checks if the tick subtracted by time is greater than the cooldown

self.frame\_index += 1 //Increments the frame index by 1

self.time = pygame.time.get\_ticks() //Resets the timer

if self.frame\_index >= len(self.animation\_list[self.action]): //Checks if the frame index if greater or equal to the length of the animation list and action

if self.life == False: //Checks if the enemy’s life is false

self.frame\_index = len(self.animation\_list[self.action]) - 1 //Sets the frame index to the length of the animation list and the action all subtracted by 1

else:

self.frame\_index = 0 //Sets the frame index back to 0

if self.action == 3: //Chekcs if the action by the enemyt is 3

self.hit = False //sets hit false

self.attacking = False //Sets attacking false

if self.action == 2: //Checks if the action is 2

self.collide = False //Sets the collision to false

if self.action == 4: //Checks if the action is 4

self.life = False //Sets life to false

self.collide = False //Sets collision to false

self.hit = False //Sets the hit to false

//movement method: Containing the parameters self and player: This allows the enemy to move

def movement(self, player):

self.attack(player) //Attack the player

if player.rect.centerx > self.rect.centerx: //Cheks if the player centre rect is greater than the enemy rect

self.targetLock = False //Sets targelock to false

else:

self.targetLock = True //Sets targetlock to true - faces the player

//A methods which allows the enemy to attack

def attack(self, player):

if self.rect.colliderect(player.rect): //Checks if the enemy rectangle collides with the player rectangle

self.collide = True //Sets the collide variable to true

if self.collide and self.life == True: //Checks if collide is true and the enemy is alive

player.hit = True

player.health -= 2 //The player loses 2 health

else:

player.health -= 0 //Player loses no health

//action\_update method: Containing the self and new\_action parameters - It checks whether there is a new action done

def action\_update(self, new\_action):

self.new\_action = new\_action //Defines new\_action as self

if self.new\_action != self.action: //Checks if new\_action is not equal to action

self.action = self.new\_action //action is set to the new\_action

self.frame\_index = 0 //Sets the frame index to 0

self.time = pygame.time.get\_ticks() //Resets the timer

//draw method - Containing the parameters self and surface - Draws the enemy on the screen and locks the enemies sight on the player

def draw(self, surface):

enemyLock = pygame.transform.flip(self.image, self.targetLock, False) //Flips the enemy sprite to face the enemy. False preventing the sprite flipping up or down

surface.blit(enemyLock, (self.rect.x - (self.offset[0] \* self.scale), self.rect.y - (self.offset[1] \* self.scale))) //Draws the enemy on the screen, where it can flip, sets the offset to the scale of the x and y coordinate

//Surface class: containing the parameter pygame.sprite.Sprite. This is part of the pygame module which allows manipulation of sprites - Makes it load the in-game surface for entities to stand on

class Surface(pygame.sprite.Sprite):

//\_\_init\_\_ method: Contains the parameters self, x, y - stores the variables

def \_\_init\_\_(self, x, y):

self.x = x \* SPRITE\_SIZE //Multiplies x with the sprite size (in settings.py)

self.y = y \* SPRITE\_SIZE //Multiplies y with the sprite size (in settings.py)

self.width = SPRITE\_SIZE //Sets the width to sprite size of 80

self.height = SPRITE\_SIZE //Sets the height to sprite size of 80

self.image = self.game.surface.sprite\_get(300, 600, self.width, self.height) //Gets the image and places to the game surface

self.rect = self.image.get\_rect() //The image is set to self.rect

self.rect.x = self.x //Sets the rect of x to self.x

self.rect.y = self.y //Sets the rect of y to self.y

**//settings.py file**

SPRITE\_SIZE = 80 //The box of the SPRITE\_SIZE

CHARACTER\_SIZE = 80 //Gets the character sprite size

CHARACTER\_SCALE = 1 //Sets the character scale to 1

CHARACTER\_DATA = [CHARACTER\_SIZE, CHARACTER\_SCALE] //A list containing the character data to be called

ENEMY\_SIZE = 80 //Gets the enemy sprite size

ENEMY\_SCALE = 3 //Sets the character scale to 3

ENEMY\_OFFSET = [30, 36] //Changes the offset of the enemy when scaling. This makes the enemy stand on the surface/platform layer

ENEMY\_DATA = [ENEMY\_SIZE, ENEMY\_SCALE, ENEMY\_OFFSET] //A list containing the enemy data to be called

//Speed

ENEMY\_SPEED = 5

//Animation - How many individual sprites there are within the sprite sheets that makes up the animation

CHARACTER\_ANIMATION = [4, 8, 4, 4, 3, 10]

ENEMY\_ANIMATION = [9, 6, 12, 5, 23]

//Colour (red, green, blue)

LIGHT\_GREEN = (19, 203, 118)

YELLOW = (203, 173, 19)

RED = (203, 19, 41)

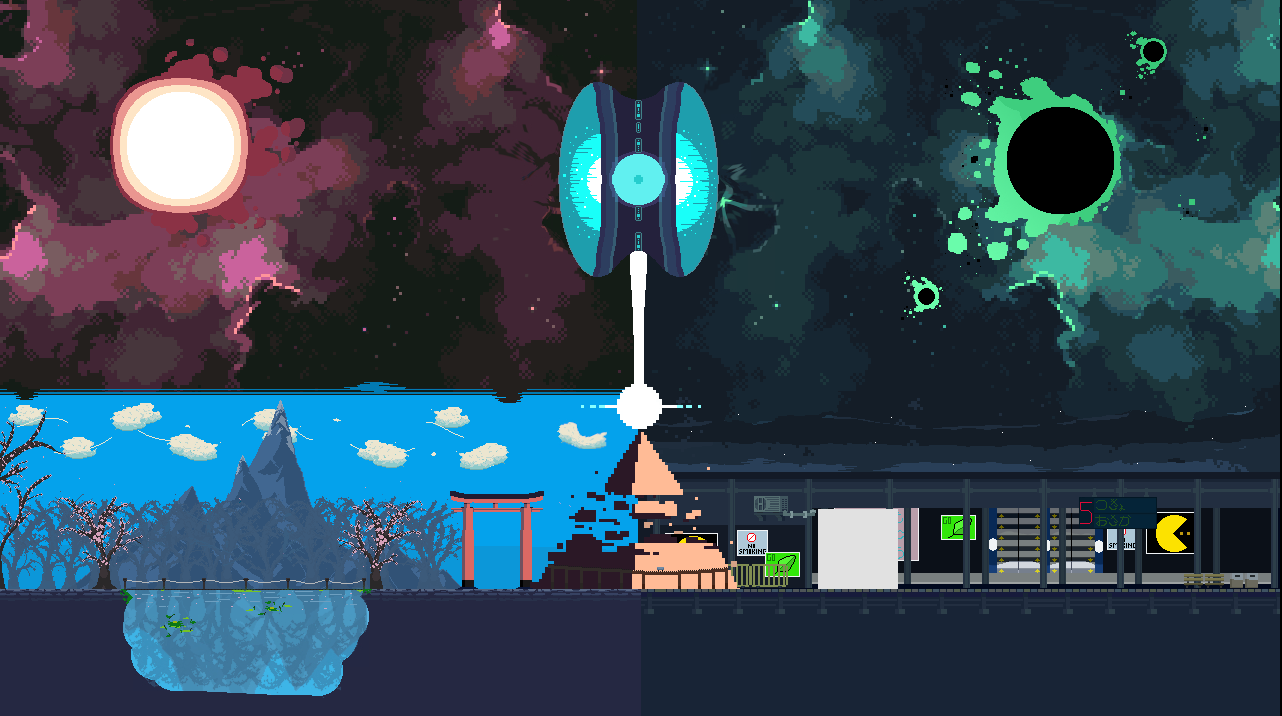
### Test plan table

|  |  |
| --- | --- |
| Test Description | Expected Results |
| Movement | The player should be able to move left, right and jump by pressing the right keys |
| Pressing the ‘a’ key | The player sprite should move left |
| Pressing the ‘d’ key | The player sprite should be able to move right |
| Pressing the ‘w’ or ‘SPACE’ key | The player sprite should jump |
| Attacking by pressing the ‘p’ key | It will create an invisible attack rectangle. When the correct key is pressed, then the attack will register. |
| Load Sprites | I would like to replace the white and purple boxes with the correct sprite visuals. With the implementation of the sprite, I can animate the entities |
| Camera | The camera should be focused on the player, so when the player moves, the camera should always be centred to the player |

### Development Diary

**Session Aim:** For this aim, I would like to implement movement within the game. I am going to draw in a white box, which is the placeholder for the actual character sprite. With this white box, I will be testing the movement by checking if the speed, velocity and gravity all work. Also, I will prevent the player from going off the screen by colliding with the borders.

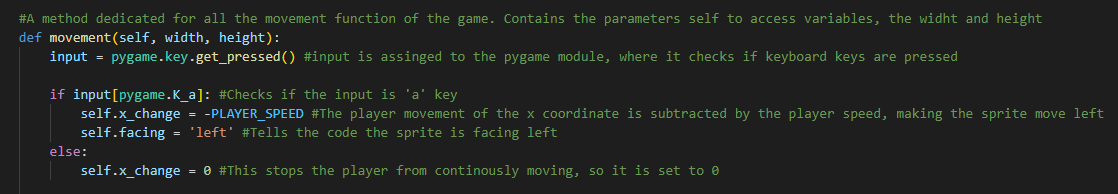
ADD CODE SNIPITS

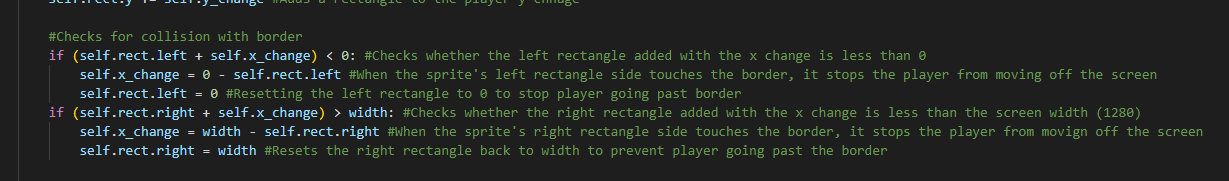


I have added the white box as mentioned in the aim. The white box is the placeholder of the character sprite (highlighted in a yellow circle). So far, I have added the movement where the player is able to move in all directions as intended.

As the player can move left, right and jump, two issues were not intended after the initial setup. The first issue was the player continuously moving left or right. Whenever I pressed the ‘a’ key, it would continuously move the character left and the same for the right movement. The second issue was even though I made a player collision system with the border, when the player kept pressing the left or right key and kept on pushing towards the border, the sprite would go past the border. Once the player let go of the keys, it would bring the sprite back onto the game screen. This was an issue, as when implementing a camera, I do not want the player to see the void and cause a visual problem.

**Outcome:**

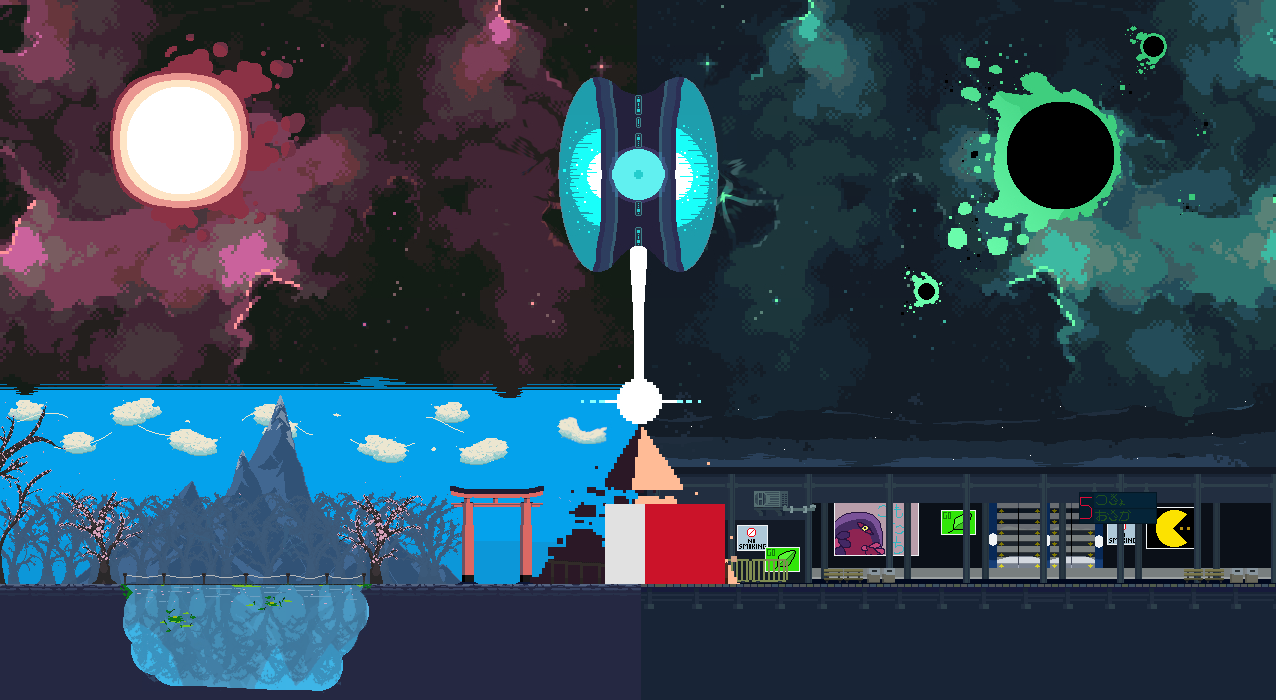
**Fixing the continuous movement:** For the continuous movement, it was a simple fix than I thought it would be. I had to add self.x\_change and set it to the value 0. This was added at the end of the ‘a’ key input. This was the solution because x\_change is the change in the coordinate system of the x value. This means as the player continuous to move left or right, the x\_change would change depending on the players current coordinates. However, for the player to remain still on the spot, the x\_change would have had to be set to 0. This is because the game registers that the sprite is no longer moving, so it stops the sprite from moving. Once the player stopped pressing the movement keys, this ended the continuous movement and stopped the sprite.

**Fixing the sprite going beyon the boundary:** To fix the issue of the player crossing the border, I set the rect.left to 0. This is because the self.x\_change = -self.rect.left. This line of code tells us that x change of the sprite is equal to the negative left rectangle. For the sprite to stop going beyond the boundaries, I will need to change the self.rect.left to 0. This is so that once the sprites left side touches the border, which is at coordinate 0, it will stop the player moving from past the border. The same applies for the width, as the right side of the sprite's rectangle collides with the end of the screen (the width (1280)), it will stop the sprite from moving past the right border.

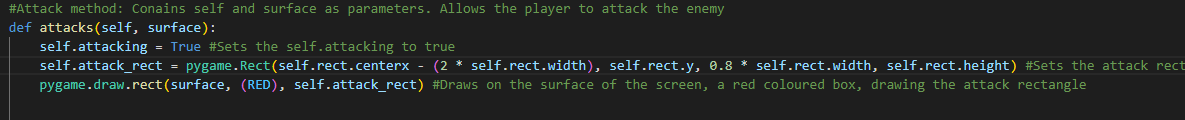
Now the player is able to move all directions and stop once the key has stopped being pressed.

ADD THE JUMP CODE

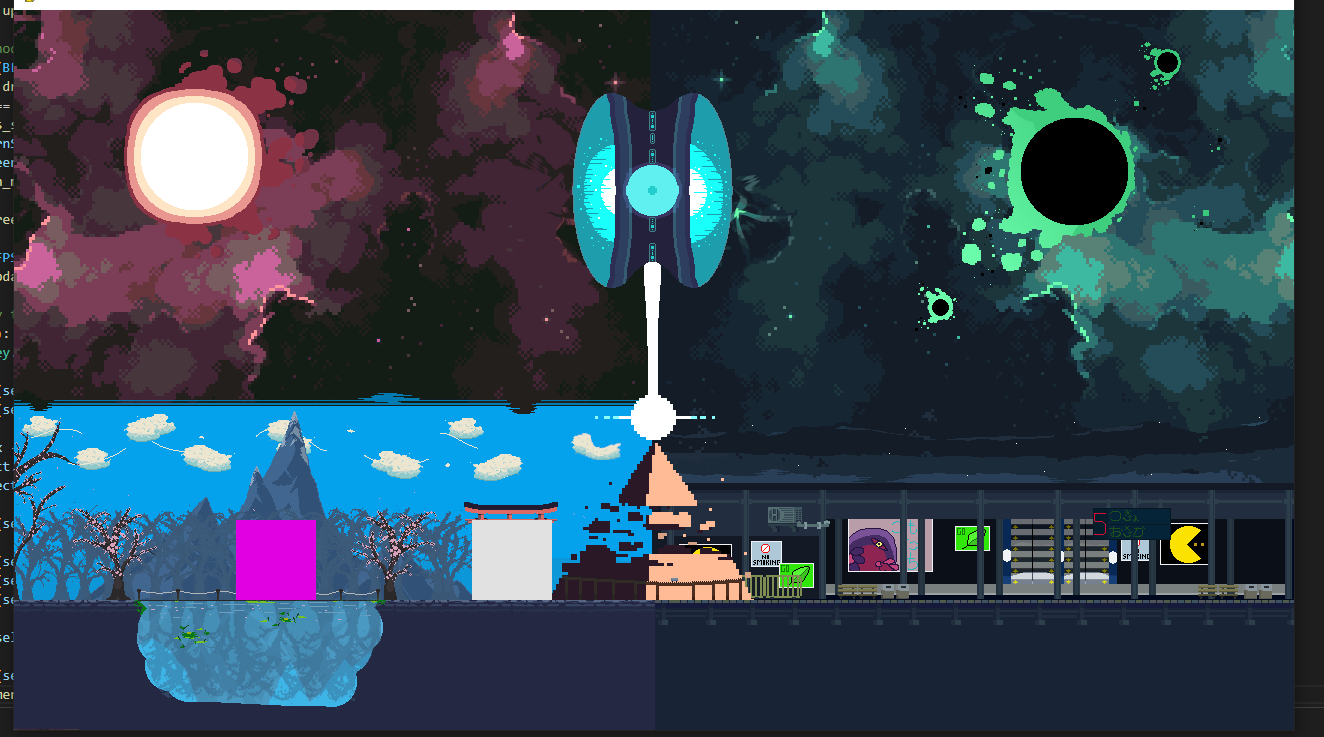
**Aim of Session:** For today's session, I would like to add the attack functionality. I am going to be firstly creating a red box in front of the white box which will indicate that the player is attacking. The red box is a placeholder, which will turn invisible. The red box will be just an indication for me to know that the attack key is working.

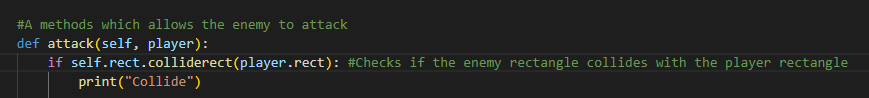
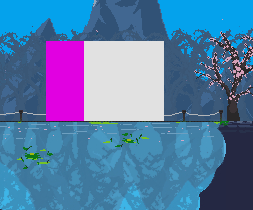
The image below is the indication that the player is attacking. The red box is added to the players centre by adding self.rect.centerx which draws the red hit box at the centre of the player. I coded in the red box to help me visualise that the player is attacking. This will help me implement the attack animation more easily as I know that the player is attacking.

The image is the indication that the player is attacking the enemy. The enemy should be attacked and lose health when the red box, which serves as the player's attack indicator, collides with the enemy hitbox.

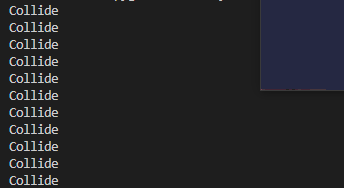


A problem that I ran into was the left click not registering when attacking. When attacking for the first time, it requires several clicks in order to register the second click for the red box to appear.

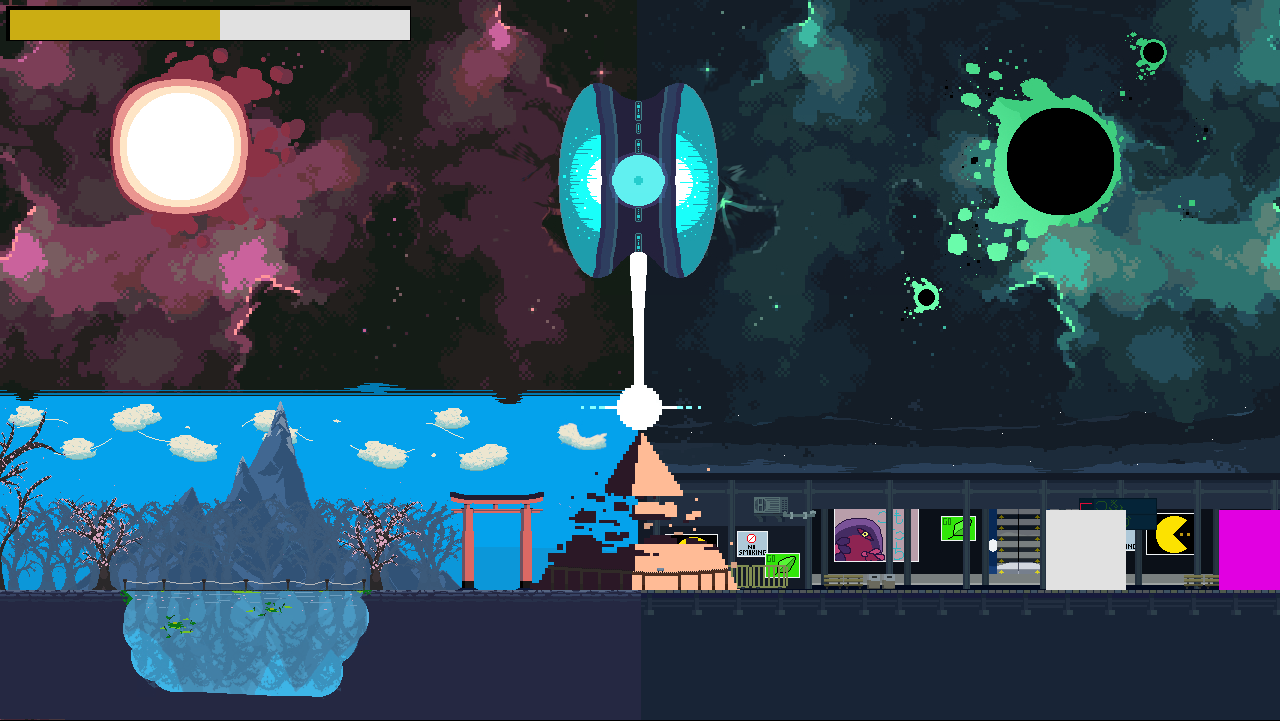
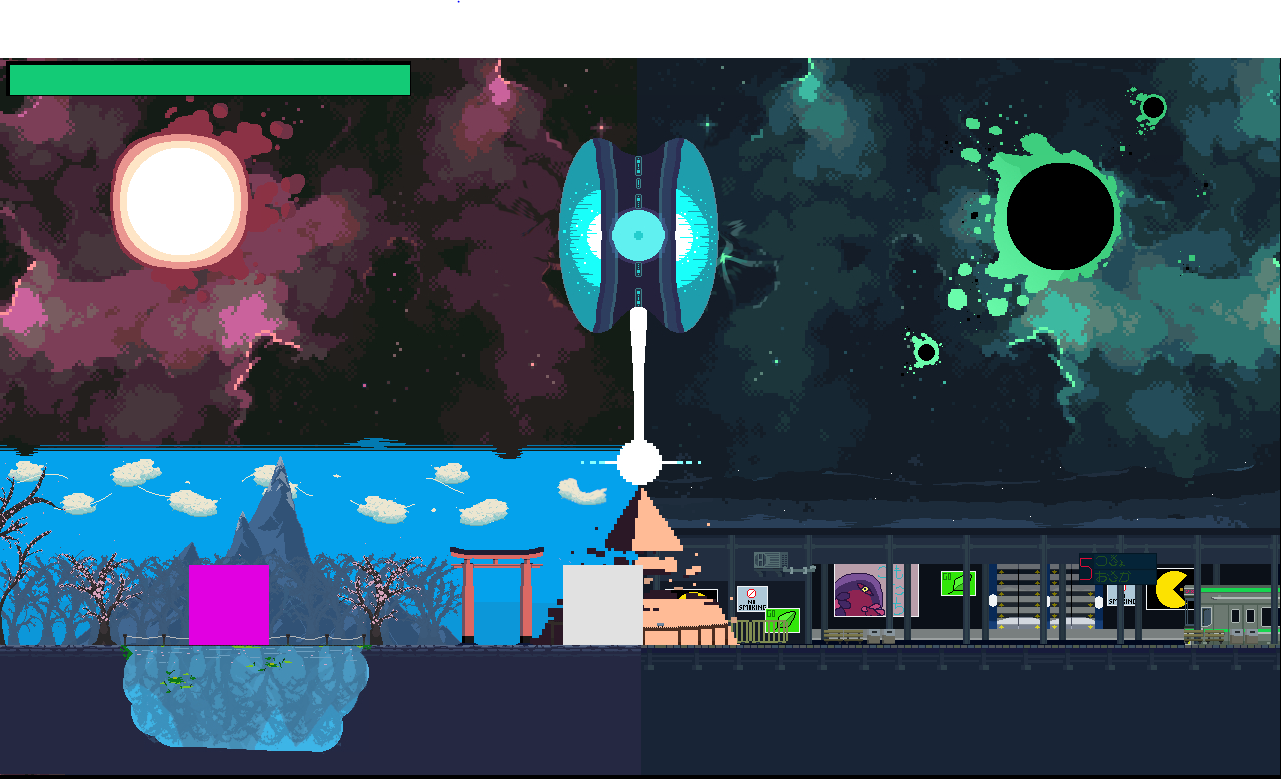
To see that the player is attacking the enemy, I have created an enemy class, where a purple box will be the placeholder for the enemy sprite.

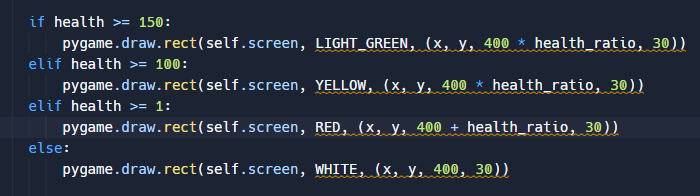
I wrote a print statement for every time the player attack rectangle collided with the enemy rectangle. This tell me that the player is attacking the enemy.

The white box being on top of the purple box showing that the player and the enemy is colliding.

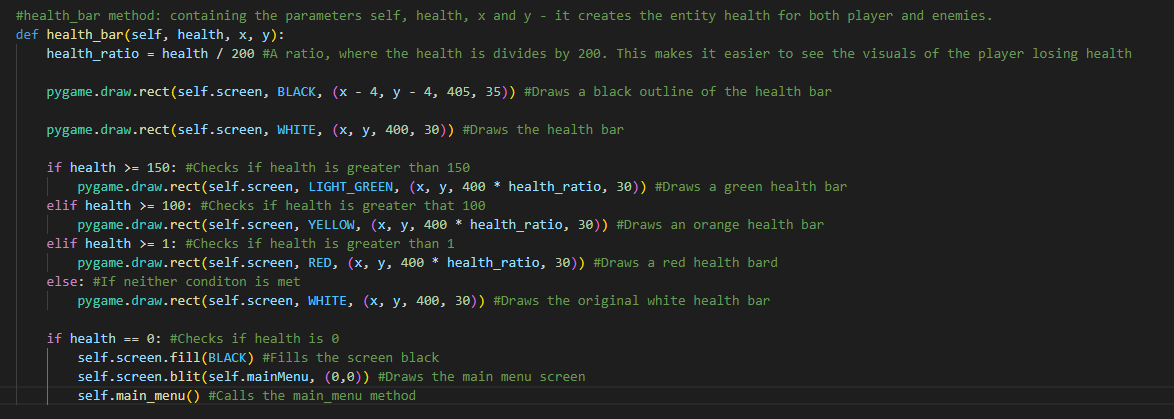


Character health bar has been added for the player. The oval yellow outlines show the health of the player at full health. When the player loses health by an enemy attack, the players will lose a set amount of health. Once the health is below 150, then the health bar will change colour to orange. Once health is below 50, then the health will be red. Having these changes in colour make the game more usability friendly, as the player will know when they are in danger and when to think strategically.



An error that occurred with the health bar was, when the health dropped below 50, the red bar would fill in the entire the health bar, rather than slowly reducing. The problem was wrong mistyping + and \* for the red health bar. This was difficult to spot, and took me 20 minutes to find since the + and \* were similar.

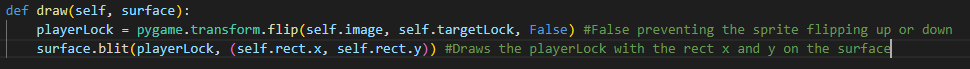
Fortunately, I found the problem and solved the issue by replacing the + to \*.



**Session Aim:** During this session, I plan to code the sprites for the game. I will be implementing the character sprites and add character animations for when they move left, right, jump or die. I will also be doing the same for enemy (the purple box placeholder).

The yellow circles will represent the sprites which have replaced the placeholder boxes.

Here is the player sprite added to the game. I have removed the white box (the placeholder) and replaced it with the character sprite.

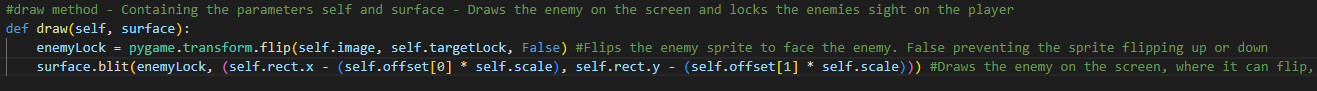


I have replaced the purple box (placeholder) with the enemy sprite.

The yellow box is the target lock. This is defined at the initialise method. This has been changed to True, as I do not want the player to focus on the enemy. I want the player to face the direction depending on the keys pressed.





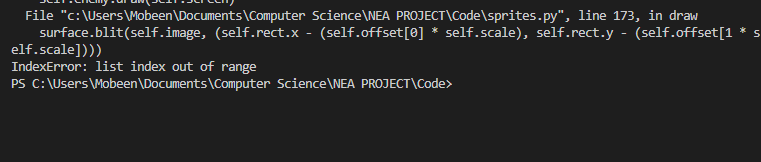


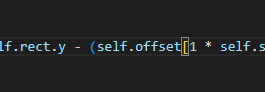
These methods draws the enemy onto the screen.

The enemyLock is a function where the enemy image can flip. With this transformation, it allows the enemy to always lock onto and face the direction of the player.

I also use the blit function to draw the enemy where I set the offset and the scale. This is because the enemy scale was set to 3. This meant that the enemy sprite was off the screen, so setting the offset multiplied by the scale allowed the enemy to be on the screen and on the surface.

**Justification: Types of errors and solutions:**

**Error [IndexError]:** When changing the offset of the enemy sprite, I closed the square brackets for the position of the offset at the wrong spot. Shown by the yellow circle



The square bracket was at the wrong position. It was supposed to be closed after the 1 index.

**Fix:** All I had to do was to close of the bracket after calling index 1 for self.offset.



**Error [AttributeError]:** When trying to flip the enemy to always face the player, this error occurred. This is because I forgot to define targetLock.

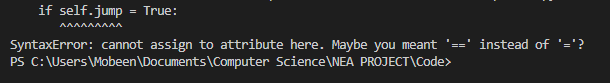
**Fix:** I defined self.targetLock within the enemy class \_\_inti\_\_ function and set it to False.



The function (sprite\_animation) is going to loop through the animations by utilising the sprite sheet. The highlighted area is an enumerate where the value of y is going to be the position/index while the animation is the sprite animation within the sprite sheet. The enumerate function allows an easy way to add a counter to a list.



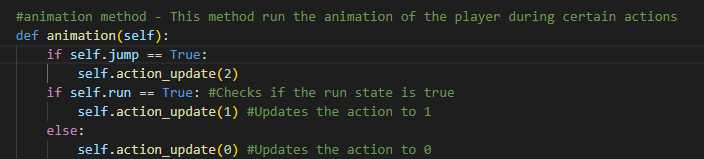
**Error [SyntaxError]:** A syntax error that I made was forgetting to add another ‘=’ sign

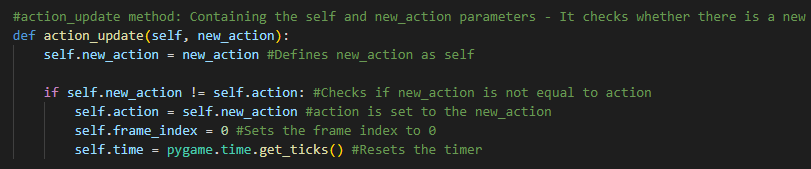


**Fix:**

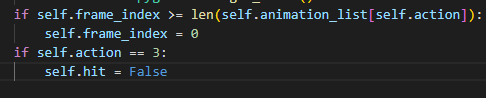


With this function, I am going to be going through the actions of the player. This animates the player when doing a certain movement. This also will be done for the enemy

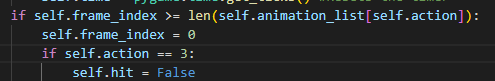


 The animation method ensures that the entity is acting as intended. If the player is doing a certain action, then the self.action\_update method is called. The action\_update method updates the animation of the code entity. When the current animation is running and a new action is played, then the new action done.

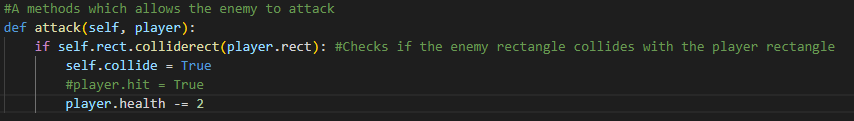
**Logic Error:** The problem was that I forgot to indent the action within the frame index.

The reason for the problem was because when running the action, the frame index has to always set to 0. This ensures that the correct animations are running when doing a certain action.

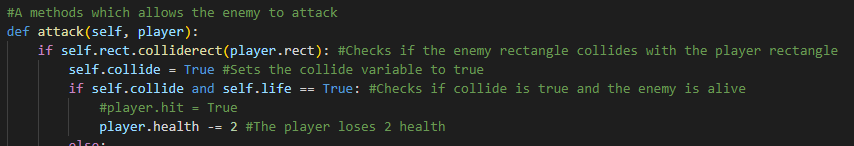
**Fix:** What I want is the code to check if the frame index is set to 0 and then check the action being done. This allows me to manipulate the animations depending on the context.



The enemy sprite would still be loaded into the screen after the enemies died, causing the player to continuously take damage whenever they collided with the enemy rect. Therefore, I had to make changes to the attack function for the enemy.

**Original (Before Changes):**

Within the attack function I added a ned condition. The original program checked whether the enemy was colliding with the player. With the new condition, it checked whether the enemy was alive. If the enemy was alive, then the player would take damage. However, if the enemy was not alive, then the player would take no damage

**New (After Changed):**

### Result of testing plan

|  |  |  |  |
| --- | --- | --- | --- |
| Test Description | Expected Results | Type | Actual Test (Success/Fail) |
| Movement | The player should be able to move left, right and jump by pressing the right keys | Normal | Success |
| Pressing the ‘a’ key | The player sprite should move left | Normal | Success |
| Pressing the ‘d’ key | The player sprite should be able to move right | Normal | Success |
| Pressing the ‘w’ or ‘SPACE’ key | The player sprite should jump | Normal | Success |
| Attack | It will create an invisible attack rectangle. When the correct key is pressed, then the attack will register. | Normal | Success |
| Load Sprites | I would like to replace the white box with the correct sprite visuals. With the implementation of the sprite, I can animate the players | Normal | Success |
| Camera | The camera should be focused on the player, so when the player moves, the camera should always be centred to the player |  |  |

### Stakeholder Feedbacks

## Iteration 3: Pause and Quit menus and Enemy AI

### Aim

The aim of this iteration is to allows the player to pause and quit the game. The player should be able to pause the game, where they will have a choice to return to the game or quit. Additionally, I would like to implement enemy AI, where the enemies have a mind of their own. This will liven the game up as the player would be immersed into the game.

1. Enemies should spawn at random positions; this makes it unique and helps the player not be bombarded with enemies. In addition, it reduces the likelihood of entity cramming, where there are too many entities in an area, causing lag. I can prevent this by limiting the number of spawns. I am going to do this by having a new enemy spawn once another enemy dies
2. When health enemy health reaches 0, they should die and a new one should spawn
3. When the player dies, it should display the game over and return to the main menu
4. Allow the user to pause displaying the ‘resume’ and ‘quit’ options

### Key variables

|  |  |  |
| --- | --- | --- |
| **Name** | **Data Type** | **Justification** |
| pause\_screen | function | The pause screen function that when the Escape key is pressed, then the game should pause |

### Pseudocode

### Test plan table

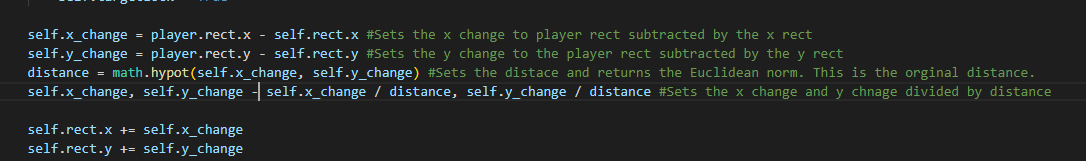
|  |  |
| --- | --- |
| Test Description | Expected Results |
| Player presses the ‘ESC’ key | Pauses the game |
| Display the pause menu | The pause screen should be shown when pressing ‘ESC’ key where it would display return and quit button |
| Clicking the return button on the pause menu | It should return the player back to the game |
| Click the quit button on the pause menu | It should quit the game |
| Display the game over screen | When the player’s life is 0, then it should display the game over screen to tell the user has lost/died. |
| Clicking the return to menu button on the game over screen | When the player dies, then there would be an option for the player to return back to the main menu on the game over screen. |

### Development Diary

**Session Aim:** I would like to implement the pause menu. When the player presses the ‘ECS’ key on their keyboard, the pause menu should pop up. This should pause the game until the player presses the resume button. There will be another option for the player to quit the game. When the quit button is pressed, then the game will close.

**Creating Enemy AI:**

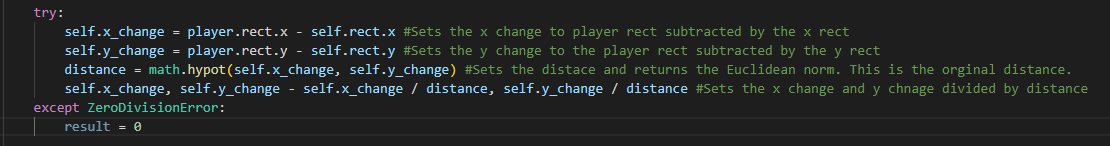
When adding a pathfinding algorithm for the enemy to follow the player, I found it hard to implement an algorithm. For that reason, I used this Stack overflow[[7]](#footnote-7) to help me. I have understood that a lot of maths is required for the enemy entity to follow the player. Therefore, I found that math.hypot is essential. This is the distance which returns the Euclidean norm. This is the original distance of the player which moves the enemy towards the player.



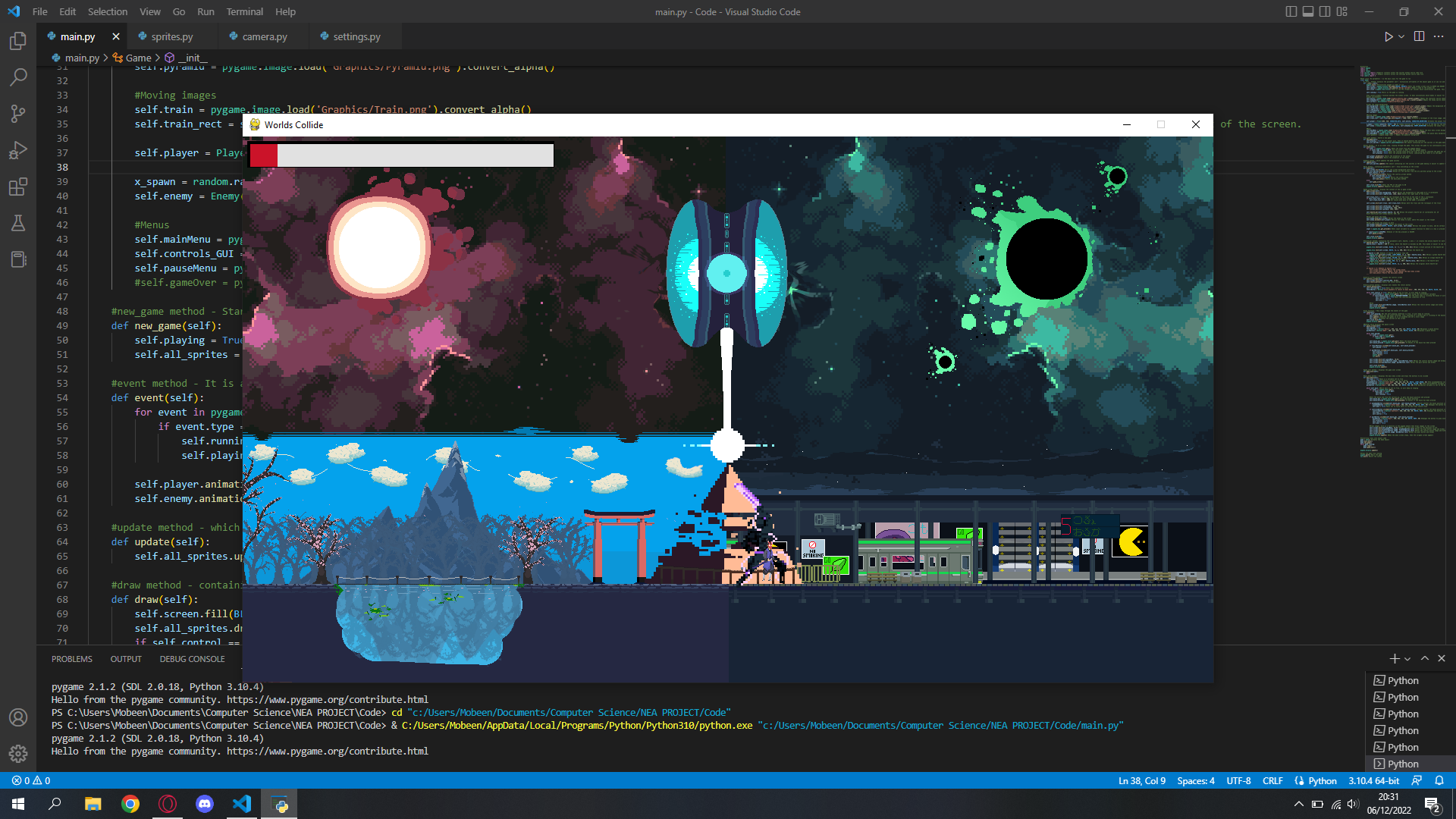
**Error [ZeroDivisionError]:** However, when running this code, I came up with this error. This is telling me the value gotten from the x and y change divided by distance, that value is divided by 0. This causes a problem as it goes through infinity numbers when divided by 0.



**Fix:** By adding try and expect function, it makes it so, when the ZeroDivisionError occurs the program will expect for a ZeroDivisionError to occur.

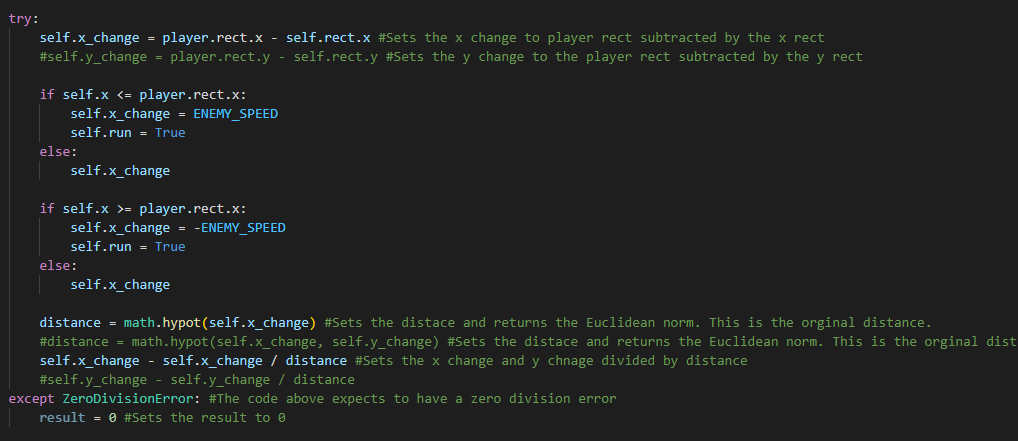


However, when implementing the code into my game, the enemy sprite would quickly stay on top of the player. This made it so the player would lose health continuously since the enemy and player hitbox a colliding. Therefore, I had to change the code.

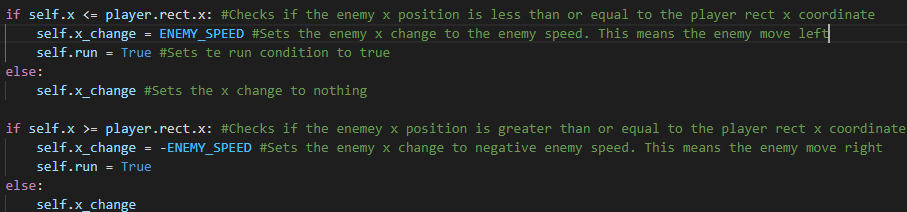


Firstly, I commented out the unnecessary parts of the original program. This is because whenever the player jumped, the enemy would jump. This was unnecessary because adding multiple enemies would make it hard to see what is happening within the game. Which is why only the player is able jump.

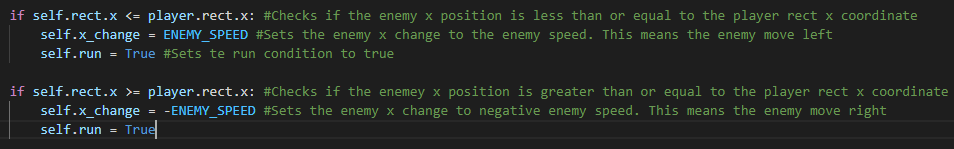
The highlighted yellow boxes show the commented-out regions of the game.



After this, I decided to a block of code into the try/expect function.

With this code, the enemy moves toward the player. However, this has caused some problems. The first problem is when the player collides with the enemy, the animation of the enemy attacking should be running however, the run animation state takes higher priority. Therefore, I would need to change the code. Another problem is that the pathfinding to the player was not really complete. The enemy AI would trace the players x change. Even though this would make sense, for some reason the enemy was not doing the function as intended.

For this pathfinding of the enemy AI, I found a fix:



I also had to change the enemy speed, as it was catching up to the player way to fast, causing the player to lose too much health

### Result of testing plan

### Stakeholder Feedbacks

1. <https://en.wikipedia.org/wiki/Puberty> (26/09/2022) [↑](#footnote-ref-1)
2. https://www.reddit.com/r/truegaming/comments/9a5sy0/why\_is\_combat\_so\_prevalent\_in\_games/ (23/09/2022) [↑](#footnote-ref-2)
3. <https://en.wikipedia.org/wiki/COVID-19_pandemic> (26/09/2022) [↑](#footnote-ref-3)
4. <https://www.cbc.ca/news/canada/saskatoon/u-of-s-research-finds-video-games-can-relieve-stress-improve-mental-health-1.5563824> (26/09/2022) [↑](#footnote-ref-4)
5. <https://fortnite.fandom.com/wiki/Fortnite_Wiki> (26/09/2022) [↑](#footnote-ref-5)
6. <https://www.pygame.org/docs/> [↑](#footnote-ref-6)
7. <https://stackoverflow.com/questions/20044791/how-to-make-an-enemy-follow-the-player-in-pygame> [↑](#footnote-ref-7)