**Zain Mobarik**
**CS Coursework**

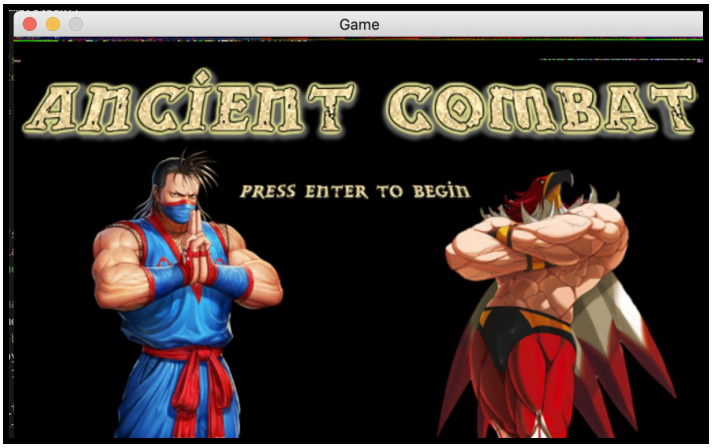
# AQA A-LEVEL CS COURSEWORK:
## ANCIENT COMBAT

Dr. Challoner's Grammar School

Candidate Number - 7233

Teacher - Mr Keen

Centre Number - 55205




# Contents

# Introduction

## PREFACE

I had spent a long time looking for a project that would click with me and be on a level I could comprehend. I had different projects in mind but most of them weren't as exciting for me. For example, I looked into making a school facility hiring system and an employee timetable database as well as various other projects. But this project was more to my liking because I was intrigued by the idea of making something my fellow peers and I could actually use for ourselves. Furthermore, this project was just the right difficulty for me - I could understand all the code yet, it was still a challenge to reach the end. This is why I chose to make this Street Fighter type game called, “Ancient Combat”.

## PURPOSE

Since my early gaming days on my Nintendo DS and PlayStation 2, I have always enjoyed tough, fighting games with super complicated special moves and awesome attacking combinations. When it comes to creating a game, it’s almost like a piece of art - you put a bit of yourself into it. It’s this visualisation of my ideas - the fact that the thoughts in my head can become a real, working game is what enthralled me to just go for it and really make this my game. Therefore, the purpose of this project is to sharpen my skills and use them to make my plan come to life in a game that represents me and is a part of me which others can enjoy.

## KEY

(A1) = Any errors or issues I had in my code. I link this using the same key under the “Test Table” section

(B1) = Any changes I made to my code due to a design change. This could either be from feedback from an end-user or from my own research. I link thus using the same key under the “Analysis” section

**Bold Words** = Any word or phrase in bold is a direct quote from my code. I purposefully do not quote my code using speech marks (“”) because it might confuse the reader that the part of my code that I quoted is a string value not

**[1]** = An indication of the image number of my screenshots. This is so I can easily refer to a screenshot when I am talking about it in my implementation.

Blue Text = Any attributes on my class diagram that I added as I was coding the project that weren't there in my original plan.

## SUCCESS CRITERIA

1. Setting up the game
    1.1. Setting up the game window
        1.1.1. Mouse not to be used except for the ability to press the red “x” button to close the game.
        1.1.2. Reasonable window size.
        1.1.3. Press ESC anywhere to quit the program
        1.1.4. BACKSPACE to go to the previous screen.
    1.2. Planning out the various game screens and their order
        1.2.1. Splash Screen
            1.2.1.1. First screen of the game.
            1.2.1.2. Background music
            1.2.1.3. Main characters and name of the game clearly presented
            1.2.1.4. Indicator on how to start the game.
        1.2.2. Menu Page
            1.2.2.1. Clear options to navigate throughout the game. “Start Game”, “Instructions”, “View Highscores”, “Settings” and “Exit”.
            1.2.2.2. If a user uses W (up), S (down) keys to go on a button, the button should illuminate to indicate that that is the option the user is currently on.
            1.2.2.3. RETURN key on an illuminated button should select it.
            1.2.2.4. Different background music.
        1.2.3. Settings Page
            1.2.3.1. If RETURN is pressed on the **Settings** button, display the **Settings** page.
            1.2.3.2. Display all options the user can change. Sound effects on/off, controller settings (WASD to arrow keys), game speed (slow, medium or fast), screen annotation on/off
        1.2.4. High Score Page
            1.2.4.1. If RETURN is pressed on the **View High Scores** button, display the **High Scores** page.
            1.2.4.2. Display top 5 scores highest to lowest.
            1.2.4.3. Rank, Score, Name (3 letters), Date (e.g 13/09/2020) subtitles.
        1.2.5. Instructions Page
            1.2.5.1. If RETURN is pressed on the **Instructions** button, display the **Instructions** page
            1.2.5.2. Navigation keys and game moves displayed as well as in-game controls.
2. Character Selection
    2.1. If RETURN is pressed on the **Start Game** option, display the Character **Select** page.
    2.2. Main characters to choose from must be clearly shown.
    2.3. Move an arrow up and down (using W and S) to select a character.
    2.4. Different background music.
3. Game Logistics
    3.1. 360 second timer. The user has 360 seconds to defeat 21 enemies to move on to the next level.[1]
    3.2. Every 15 seconds a new enemy appears.
    3.3. 100 health points to begin with.
    3.4. Enemies have 20 health points to begin with.
    3.5. Enemies must always follow you.
    3.6. All users' scores will be recorded and saved to a database.
    3.7. Different sound effects for each character.

1. The user has 360 seconds to defeat 21 enemies to move on to the next level. 21 kills to move to the next level. Every 15 seconds a character appears. Killing 21 characters with one character appearing every 15 seconds, requires 315 seconds. 360 - 315 gives the player 45 extra seconds to try to get 21 kills.

4. In-game Screen Layout
   - 4.1. Once the character is chosen, display the **level 1 background image**.
   - 4.2. Display chosen character.
   - 4.3. Timer on the top middle of the screen, health bar in the top left, kill count in the top right. Special move bar underneath health bar. Level indicator under the timer.
   - 4.4. Enemy's health and special move bars on top of their heads.
   - 4.5. Bins and boxes scattered throughout the game to be used as a projectile or to destroy.
5. Progressing in the game
   - 5.1. Moves
     - 5.1.1. Standing
     - 5.1.2. Left and right movement
     - 5.1.3. Punching
     - 5.1.4. Kicking
     - 5.1.5. Jumping
     - 5.1.6. Blocking
     - 5.1.7. Special Move
   - 5.2. Interactions within the game
     - 5.2.1. Landing a punch or kick on the enemy loses them 5 health points. Special move on the enemy deducts 10 health points. Throwing an obstacle at the enemy deducts 5 health points.
     - 5.2.2. If an enemy lands a punch or kick, you lose 2 health points. Special move loses you 6 health points.
     - 5.2.3. Enemies cannot pick up obstacles.
     - 5.2.4. If anyone blocks, no damage is deducted.
     - 5.2.5. Every 10 health points decrease, the health bar will be updated to display the new health.
   - 5.3. Special Move
     - 5.3.1. Special move bar with 5 segments. Landing a punch, a kick, blocking an enemy's attack must fill the special move bar by a segment.
     - 5.3.2. Special move uses up all 5 segments.
     - 5.3.3. Enemies' special move bar has 4 segments. Landing a punch or a kick must fill the special move bar by a segment.
   - 5.4. Progressing onto the next level
     - 5.4.1. If the user is successful in defeating the 21 enemies in the time given, then the next level will begin. 5 levels in total.
     - 5.4.2. The next wave of enemies will have 5 more health than the previous level and deduct 2 more damage in all attacks. The time allowed for that level will increase by 15 seconds.
     - 5.4.3. If you complete all 5 levels, there will be an end credit and you will return to the **Menu** page.
   - 5.5. Dying in the game
     - 5.5.1. If the user is unsuccessful, then a **GAME OVER** message will display and return the user back to the Menu screen.

# Analysis

## FEEDBACK FROM END-USERS

### SYSTEM SETUP

To begin with, I had to make a decision regarding what software I would use to code my project. I had “Atom” and “Visual Studio Code” downloaded as two potential text editors that I could use. I went for Visual Studio Code as I felt the design of the software was more intuitive and I felt as though it had a more interactive and accessible user interface due to the colour coding system in place.

### IN-GAME TECHNICALITIES

My target audience includes anyone who likes playing video games. So, after an initial consultation and an explanation of my game with an end-user, I received very valuable feedback. For example, I was asked whether there would be the option to jump and kick at the same time; something which I had not yet considered. Furthermore, the user asked me about the possibility of diagonal jumps; again something which I hadn’t included in my first draft of the game. These are two key abilities I have decided to add to my project after concluding my first consultation with an end-user.

In a subsequent conversation with a member of my target audience, they mentioned the use and value of adding annotation of the entire screen when a new game starts. This could be very helpful for new players who may not know what all the bars and numbers on the screen represent. This is also something I have added to my plans.

Pertaining to high scores, I originally planned to let players choose a 5 letter name to save their score. However, I couldn’t think of a way around two people having the same name. After some feedback with an end-user, I have concluded that it would be better to have 3 letter long names. Furthermore, if two users have the same name, then the higher score will override the previous score saved to that name, otherwise, the previous score will remain. This is very similar to how arcade high scores work and will consequently save a lot of memory. Moreover, it was suggested that it would be of use to incorporate a date and time system for the high scores so that players know when the event took place. These are some new ideas that I will append to my project.

In addition, one of my end users is left-handed and brought to my attention that using the WASD keys for movement is quite uncomfortable and that most left-handed users prefer the arrow keys. Therefore, I have instated an option in the setting screen to change the keys that correspond to the movement of your character.

Another piece of feedback I received was to add the ability to change the speed of the game on the settings screen. It seems that some more advanced players prefer a faster game speed, with the characters and computer moving at a much quicker pace. Similarly, those playing the game for the first time or those not used to this style of video games would prefer a slower setting for the movement for all the sprites. This is a very intuitive idea and definitely something I will incorporate into my project.

(B3) After getting some end-users to test my game to the point where I had coded it, one user suggested to include a rank column in my “Highscores” page as right now I only had “Name, Score, Date”. This seemed like an obvious addition to include in my “Highscores”’ page.

I had an interesting discussion with an end-user regarding scores of players even if they die partway through the game. We came to the conclusion that it would be quite important to record unsuccessful attempts as otherwise (if no one is successful for a while), there will be no scores on the leaderboard - which would be quite strange for a game. Therefore, this is something I shall investigate further and implement into my project.

(B4) Originally, I had it so that all the punching and kicking buttons were in the middle of the keyboard. This would then mean that no matter if you are using the WASD or arrow keys for movement, all the other buttons will be the same distance on either option so it's fair to both left and right-handed players. However, after some end-user feedback, it was suggested that this is not the right way to go about and that I should just create two entirely separate key layouts for right and left-handed users. Therefore, I changed my project so that if you use the WASD keys then, all the other moves will be on the right-hand side of the keyboard to give your hands enough space and similarly if you use the arrow keys then, all the other moves will be on the left-hand side of the keyboard.

(B5) A final run through of the keyboard layout made me make a crucial change. 5/5 users agreed that instead of having the spacebar for jump and a random button for block, it makes a lot more sense to have block as **S** or the **Down Arrow** key and jump as **W** or the **Up Arrow** key. This seems very logical as it just reduces the amount of hand movement required to play the game.

(B6) Regarding special moves for the characters, I decided to reduce the number of special moves for the characters from 2 to 1. This is because in most games there only seems to be one special move per character and it makes more sense.

## SPRITE SELECTION

I have chosen a total of 12 sprites for my project; 5 level backgrounds, 5 opponents and 2 main characters. I found a website called “The Spriters Resource” that had all the sprites I was looking for. This took place on the 22nd of June. I also found out that all the images from this site were allowed to be used for educational purposes or to be used in games for pleasure, but not in games made for profit. Thus, with the use of sprites from this website, I will not be monetising my project. [NOTE: level background is not from the website but from google]

After showing the original sprites I had chosen for the characters and opponents, I received a lot of valuable feedback. One end-user pointed out the similarity between one of the two characters that you can choose to fight with and one of the opponents. Furthermore, another end user proposed to me that I should keep the style of all my sprites the same. For example, in my first draft; one sprite, in particular, was much more animated than the others and stuck out from the rest in an unaesthetic way. Therefore, I have changed the sprite for one of the main characters and the sprite for the animated opponent. This will provide a much more professional and consistent user experience.

However, I still felt as though my characters were slightly odd together. Consequently, I found a group of characters from a game called “King of Fighters” and will be using them because an end-user suggested that this would work out much better as all the characters will be the same style and level of animation. After viewing the level background image I chose for my project in the window I coded, I was quite unsatisfied with the quality and resolution of the background. Consequently, I spent some more time researching various backdrops that would fit the dimensions of my game window and be of sufficient quality. In the end, I chose this backdrop (insert image). However, according to my original objectives, I decided on having a 3D platform with a moving backdrop, therefore, when I edited this backdrop in Powerpoint to see what three of them would look like side by side, I found the repetition of the big building quite unaesthetic. I then proceeded to review the image I had chosen with an end-user and their feedback suggested that the repetition of the big building wasn't a big deal and that they would not mind it in the game.

## GAME WINDOW

I went for a 720-pixel resolution for my backdrop. This is a better resolution in terms of quality than a 480-pixel background and most laptops are compatible with 720 pixels, so there is a very low chance of crashing. Furthermore, after consultation with an end-user, it was suggested that a bigger size would enhance their in-game experience and allow the detail of the game to shine through as well as the playability of the project.

(B1) Furthermore, regarding the size of the window, I decided to decrease the height and width of the game window from 1280 by 720 pixels to 600 by 380 pixels to make sure that all devices can run my project without any lag. Consequently, I tested to see whether I had to manipulate the sizes of any of my characters to fit the new screen dimensions and after some feedback from an end-user, the characters seem to fit in the window quite well.

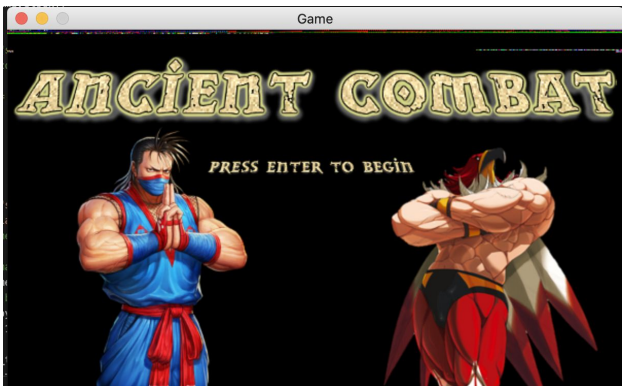
## STYLISTIC FEEDBACK

For the different pages, I decided to use a website called “Text Craft” to find different fonts that would suit the style of my game. I chose the “STONECRAFT” font for headers in my Menu pages and I chose “ArcadePix” for smaller bodies of text.

(B2) Moreover, in relation to the general look and feel of the game, I analysed various fighting games online and inspected their menu screens and loading pages. This gave me an idea of what my title screen and general layout should look like. It became apparent that one key feature of these games is the display of the main character(s) in an imposing stance in most, if not all, of the introductory screens. Furthermore, I noticed a “top to bottom” list of all the options available for the user. These ideas are shown in the images below:



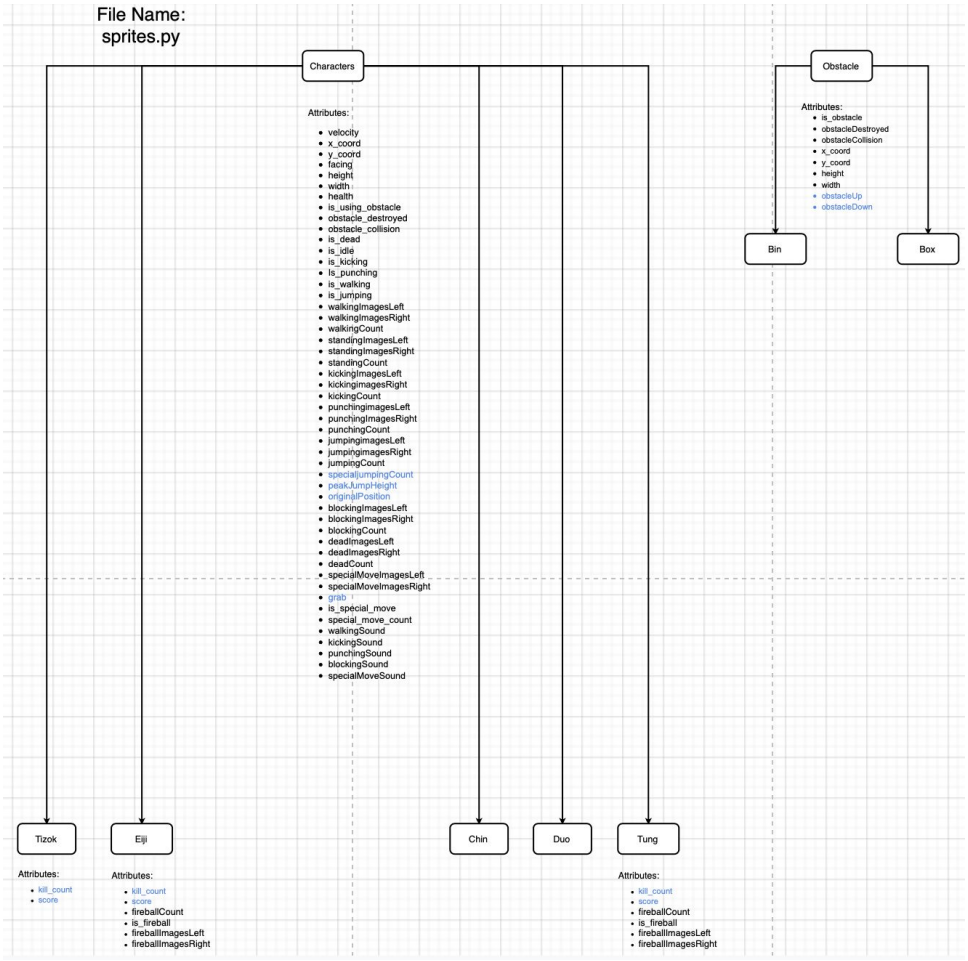
I went on to implement these ideas here:

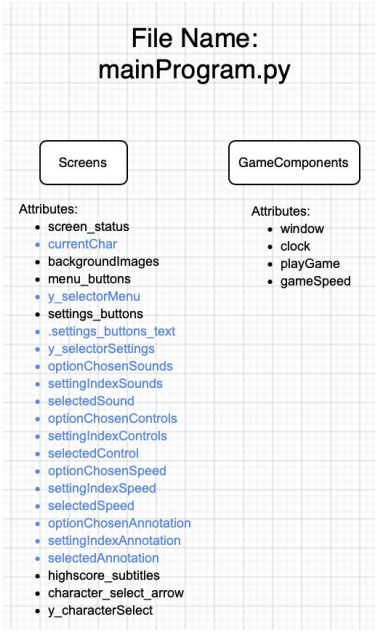




A user reported that the feel and authenticity of my game was true to that of similar fighting games he had played before.

## FLOWCHARTS AND CLASS DIAGRAMS



# Implementation

**Code 1**

```
1   import pygame
2
3   pygame.init()
4   window = pygame.display.set_mode((1280, 720))
5   pygame.display.set_caption("Game")
6
7   background_image = pygame.image.load("level_1_test.jpg")
8
9   loop = True
10
11  while loop:
12      window.blit(background_image, (510, 650))
13      pygame.display.update()
14
15  pygame.quit()
16
```

**[1]**

Python has a variety of modules and libraries that can be imported to allow users to enhance their code using in-built functions. Pygame is a good example of this that allows users to code games in Python.

**.init()** is a method that initialises the pygame instance from the library to allow the class objects to be constructed to build your game.

I applied the **.set_mode** method which is encapsulated inside the **display** class on the instance of pygame that I initialised earlier and passed in the argument width and height as a tuple giving the dimensions in pixels. I then assigned this to the instance object **window**.

Furthermore, I then applied another method **.set_cpation** which passed in the title as an argument that will appear on the top of the tab as **"Game".** The **.setcaption** method is also encapsulated inside the **display** class and applied to pygame.

Then I applied the method **.load** which allowed me to pass in and load the background image from my laptop. The **.load** method is encapsulated inside the **display** class and consequently applied to pygame. This is then assigned to the instance object **background_image**.

Firstly, I set the boolean value to **True** and assigned it to the variable **loop**. On line 11, I created a while loop that will execute the instructions indented inside until the condition (the variable **loop**) is set to false. Inside the while loop, I have applied the **.blit** method to the instance object **window** and passed in two arguments; the image that I want to be displayed and the dimensions of the window. Finally, on line 13, I called the **.update** method which is encapsulated inside the **display** class and applied it to pygame. This constantly updates the **blit** method so that the window is always being displayed.

**pygame.quit()** allows the user to press the close tab button to exit the programme.
(A1) Lesson 1 issues: Screen only popped up for a very short time then disappeared.

## Code 1: Issue Solved (A2)

```
1   import pygame
2
3   pygame.init()
4   window = pygame.display.set_mode((1280, 720))
5   pygame.display.set_caption("Game")
6
7   background_image = pygame.image.load("images/Levels/level_1_test.jpg")
8
9   loop = True
10
11  while loop:
12
13      window.blit(background_image, (0,0))
14      events = pygame.event.get()
15      for event in events:
16          if event.type == pygame.QUIT:
17              loop = False
18
19      pygame.display.update()
20
21  pygame.quit()
```

[2]

The previous code threw an error of the window appearing then disappearing instantly when the program ran. I attempted to fix this using the code from line 14 to line 17.

On line 14, I call the **.get()** method (which took no arguments) on the event class and apply it to the instance of pygame that I created. This gets all the different events that occur in pygame and I have assigned (stored) this information to the variable **events**.

On line 15, I have iterated through every **event** in all the different events in pygame (which I have stored in the variable **events** as mentioned before). It is important to note that the word **event** which I have used in the iteration is a temporary variable whose scope lies only within the **for loop**.

On line 16, I have used a conditional statement. Once the computer starts to iterate through all the different events in pygame and momentarily storing each one in the temporary variable **event**, it will check whether the event is of the type **QUIT**. If the event is of the type **QUIT**, then and only then will line 16 be executed. Otherwise, the computer will proceed to the next iteration of the **for loop**.

On line 16, if the condition in line 15 is met, then I assign the boolean value **False** to the global variable **loop**. This essentially breaks the entire **while loop** that I constructed on line 11. This is because the **while loop** was conditioned to run so long as the variable **loop** was assigned the boolean value **True**, however, on line 16, the opposite occurs. Thus, the programme exits the **while loop**.

This then fixed the error of the screen only showing for a very short time. This is due to the fact that we are now saying that for any event that occurs under the instance **pygame**, only exit or stop the programme if the event is of the command **QUIT**. Now, the screen is constantly being displayed until someone presses the exit button.

## Code 2: Implementation of ESC button to exit game (A3)

```
1   import pygame
2
3   pygame.init()
4   window = pygame.display.set_mode((1280, 720))
5   pygame.display.set_caption("Game")
6
7   background_image = pygame.image.load("images/Levels/level_1.png")
8
9   loop = True
10
11  while loop:
12      window.blit(background_image, (0,0))
13      events = pygame.event.get()
14      for event in events:
15          if event.type == pygame.QUIT:
16              loop = False
17
18      else:
19          keys = pygame.key.get_pressed()
20          if keys[pygame.K_ESCAPE]:
21              loop = False
22
23      pygame.display.update()
24
25  pygame.quit()
26
```

**[3]**

Now that I was able to run and exit my programme with ease, I wanted to customise how the user would be able to execute the process of exiting the game. I decided to use the **Escape** button to be used to exit the game (please refer to success criteria 1.1.3). This is due to the ease, intuition and commonality of using this button for quitting games and programmes. This was implemented using the code from line 18 to 21.

On line 18, if the user does not perform an event that has the type **QUIT**, then the programme will check to see if the user has used another method to trigger the exiting of the game. This is the route that the **else** statement provides to allow the user to quit the game using the **escape** button.

On line 19, I called the **.get_pressed()** method (which took no arguments) on the **key** class in the instance of pygame. This allowed me to access all the keys that could be used in making a game with pygame. I then assigned this information to the variable **keys**. It is important to note that the information inside the variable **keys** is stored as a dictionary.

On line 20, I access the data structure (which is a dictionary/hash map) by giving the name of the dictionary and the key that I want to find the value of. I then use an **if** statement to say that if the key pressed is **K_ESCAPE** then continue. **K_ESCAPE** is just the key in the dictionary for the **Escape** button on your keyboard. Therefore, if this is **True** and the **Escape** button was pressed, then the programme will continue to line 21.

On line 21, I do exactly what I did on line 16 and assign the boolean value **False** to the global variable **loop** as a way of breaking out of the **while loop** I created on line 11. Now, my game can be exited using the **Escape** button on the keyboard or by clicking the close button on the top of the tab.

## Code 3: Implementation of frame rate to begin preparation for animation

```
1   import pygame
2
3   pygame.init()
4   window = pygame.display.set_mode((1280, 720))
5   pygame.display.set_caption("Game")
6
7   background_image = pygame.image.load("images/Levels/level_1.png")
8   clock = pygame.time.Clock()
9
10  loop = True
11
12  while loop:
13      clock.tick(27)
14      window.blit(background_image, (0,0))
15      events = pygame.event.get()
16      for event in events:
17          if event.type == pygame.QUIT:
18              loop = False
19
20      else:
21          keys = pygame.key.get_pressed()
22          if keys[pygame.K_ESCAPE]:
23              loop = False
24
25      pygame.display.update()
```

**[4]**

After sorting out the basic functionality of the display window, I shifted my focus towards the animation aspect of my project. I incorporated lines 8 and 13 into my program.

On line 8, I called the **.Clock()** (with no arguments) method on the **time** class on the instance of pygame. This simply creates an object to allow the computer to track the time. I then assigned this information to the global variable **clock**.

On line 13, I applied the **.tick()** method to the variable **clock**. I passed in an argument using the number 27. This ensures that the programme will never run at more than 27 frames per second. This seems like a reasonable frame rate for my animations to run smoothly in my project.

## Code 4: Introducing classes and encapsulation in my code

```
1   import pygame
2
3   def createPygameWindow():
4       global pygame
5       pygame.init()
6
7       pygame.display.set_caption("Game")
8
9   class GameComponents():
10      def __init__(self):
11          self.playGame = True
12          self.window = pygame.display.set_mode((1280, 720))
13          self.background_image = pygame.image.load("images/Levels/level_1.png")
14          self.clock = pygame.time.Clock()
15
16  createPygameWindow()
17  startGame = GameComponents()
18
19  while startGame.playGame:
20      startGame.clock.tick(27)
21      startGame.window.blit(startGame.background_image, (0,0))
22      events = pygame.event.get()
23      for event in events:
24          if event.type == pygame.QUIT:
25              startGame.playGame = False
26
27      else:
28          keys = pygame.key.get_pressed()
29          if keys[pygame.K_ESCAPE]:
30              startGame.playGame = False
31
32      pygame.display.update()
33
34  pygame.quit()
35
```

**[5]**

My next task was to clean up my code and encapsulate it into different areas to make it easier to read and understand.

I defined the function **createPygameWindow()** on line 3 to include all my code for aspects of my game related to the basic structure of the window.

For example, I defined on line 4 to make **pygame** a global variable. This was necessary as earlier on, I ran across some problems as the programme gave an error saying that **pygame** had not been defined correctly. (A4)

```
1
2
3   def createPygameWindow():
4       import pygame
5       pygame.init()
6       window = pygame.display.set_mode((1280, 720))
7       pygame.display.set_caption("Game")
8       clock = pygame.time.Clock()
9
10  class GameComponents():
11      def __init__(self):
12          self.playGame = True
13          self.background_image = pygame.image.load("images/Levels/level_1.p
```

```
Exception has occurred: NameError
name 'pygame' is not defined
  File "/Users/zainm/Desktop/coursework/game/mainProgram.py", line 13, in
__init__
    self.background_image = pygame.image.load("images/Levels/level_1.png")
  File "/Users/zainm/Desktop/coursework/game/mainProgram.py", line 17, in
<module>
    startGame = GameComponents()
```

**[6]**

To further my endeavour in encapsulating my code, I defined my first Class Object called **GameComponents**. This is to simply include the basic aspects of my project that are required for it to function properly.

For example, I have included the window size, I will be including all the different images for the backgrounds and levels under this class as well as various other lines of code that are necessary for the foundation of my project.

Some other minor changes I made were on lines 19 and 20. In image 4, line 12, I use ***while loop*** where the variable **loop** is assigned to the boolean value **True**. However, in image 5, line 19, I used the code ***while startGame.playGame***. This is because I encapsulated **playGame** as an attribute of the class **gameComponents**. This is so this attribute becomes locally defined within the class and so when I want to access this attribute outside the scope of the class (globally), I have to use the object instance of the class to do so. In other words, I have to use **startGame.playGame** to access it.

## Code 5: Getting the first few screens of my game to run.

```
3  class GameComponents():
4      def __init__(self):
5          pygame.init()
6          pygame.display.set_caption("Game")
7          self.playGame = True
8          self.window = pygame.display.set_mode((600, 380))
9          self.splash_image = pygame.image.load("images/Backgrounds/Splash Screen/splash_screen.png").convert_alpha()
10         self.menu_image = pygame.image.load("images/Backgrounds/Menu/menu_final.png").convert_alpha()
11         self.button_names = ["instructions_up", "instructions_down", "start_up", "start_down", "highscore_up", "highscore_down
12         self.buttons = []
13         self.screen_status = "level 1"
14
15         for i in range(len(self.button_names)):
16             self.buttons.append(pygame.image.load("images/Backgrounds/Menu/{}.png".format(self.button_names[i])).convert_alpha
17         self.char = pygame.image.load("images/Characters/tizoc_1/Standing/Tizoc_0L.png")
18         self.char2 = pygame.image.load("images/Opponents/Duo/duo_0L.png")
19         self.level_1bg_image = pygame.image.load("images/Backgrounds/Levels/level_1.png").convert_alpha()
20         self.clock = pygame.time.Clock()
```

**[7]**

Next, I proceeded to load all the images that I was going to use for my Splash, Menu and Level 1 screens. For example, you can see that I load the splash screen on line 9, the menu on line 10, the buttons for the menu on lines 10-16, the characters on lines 17-18, and the background image on line 19.

I applied the method **.load** which allowed me to pass in and load any image I specified from my laptop. The **.load** method is encapsulated inside the **display** class and consequently applied to pygame. This is then assigned to the instance object **background_image** or **splash_image** etc.

I attempted to make my code more efficient when loading the images for the various buttons used in the Menu. Instead of saving all my buttons individually to different variables and loading them separately, I thought it would be more effective to make a list of all the button names (line 11), then iterate through a range of the number of items in this list (line 15), then at each iteration, write the code needed to load an image from the **Menu** folder on my laptop and append the name of the button that I want to load using **.format** and using **self.button_names[i]** to append the name of the button at that iteration. Finally, I needed to append this whole line into a new list called **self.buttons**. This was a much more effective and efficient method.

I also created the attribute **screen_status** for my **GameComponents** class (line 13) to be used to change what screen the user is on. This will be further explained later on.

One final thing to bring to attention is that I change the window size from 1280 by 720 pixels (image 6, line 6) to 600 by 380 pixels (image 7, line 8). Please refer to (B1) in my **Analysis** section for further explanation.

**Code 5: Getting the first few screens of my game to run (2)**

```
54
55      else:
56          keys = pygame.key.get_pressed()
57          if keys[pygame.K_ESCAPE]:
58              game.playGame = False
59          if game.screen_status == "splash" and keys[pygame.K_SPACE]:
60              game.clock.tick(2)
61              game.screen_status = "menu"
62          elif game.screen_status == "menu":
63              game.clock.tick(2)
64              if keys[pygame.K_s]:
65                  self.yDetect += 50
66                  if self.yDetect == 115 and keys[pygame.K_RETURN]:
67                      game.screen_status = "level 1"
68      pygame.display.update()
```

**[8]**

After loading all the images I needed for my Splash, Menu and Level 1 screens, I needed to find a way to go from one screen to another. According to my success criteria (1.2.1.4), there must be a clear indicator on the splash screen "on how to start the game". I have written on my splash screen, "Press Space to begin". Therefore, I must code my programme so that when the user presses the spacebar, he goes from the splash screen to the Menu screen.

Using the aforementioned **screen_status** variable, on line 59, I use the conditional **if** statement to say that if the **screen_status** equals "splash" and the spacebar is pressed, then follow the code indented under the if statement.

Then on line 61, it says that if **screen_status** was indeed equivalent to "splash" and the spacebar was pressed, then make **screen_status** equal to "menu" to signify that the user has now moved on to the **Menu** screen.

```
43      if game.screen_status == "splash":
44          game.splashDisplay()
45      elif game.screen_status == "menu":
46          game.menuDisplay()
47      elif game.screen_status == "level 1":
48          game.levelDisplay()
```

**[9]**

At the beginning when **screen_status** equalled "splash", I called **game.splashDisplay** to display the splash screen. However, when **screen_status** changed to "menu" using the code on line 61, this meant that the first if statement on line 43 would be false and so the programme would test the next if statement which is on line 45 as it is an **elif** (else if). This would result in the statement being True as **screen_status** does now equal "menu" so the **menuDisplay()** method will be called displaying the menu screen. There would then be no need to test the final **elif** statement as the condition has been met on line 45.

**Code 5: Getting the first few screens of my game to run (3)**

```
54
55      else:
56          keys = pygame.key.get_pressed()
57          if keys[pygame.K_ESCAPE]:
58              game.playGame = False
59          if game.screen_status == "splash" and keys[pygame.K_SPACE]:
60              game.clock.tick(2)
61              game.screen_status = "menu"
62          elif game.screen_status == "menu":
63              if keys[pygame.K_s]:
64                  game.clock.tick(13)
65                  game.yDetect += 50
66                  if game.yDetect >= 315:
67                      game.yDetect = 315
68
69              elif keys[pygame.K_w]:
70                  game.clock.tick(13)
71                  game.yDetect -= 50
72                  if game.yDetect <= 115:
73                      game.yDetect = 115
74
75              if game.yDetect == 115 and keys[pygame.K_RETURN]:
76                  game.screen_status = "level 1"
77
78      pygame.display.update()
79
```

**[10]**

Now that the user can successfully go from the **Splash Screen** to the **Menu Screen** using the spacebar, my next task was to code the workings of the buttons on the **Menu Screen**.

```
3   class GameComponents():
4       def __init__(self):
5           pygame.init()
6           pygame.display.set_caption("Game")
7           self.playGame = True
8           self.window = pygame.display.set_mode((600, 380))
9           self.splash_image = pygame.image.load("images/Back
10          self.menu_image = pygame.image.load("images/Backgr
11          self.button_names = ["start_down", "instructions_d
12          self.buttons = []
13          self.screen_status = "splash"
14
15          self.menuChoice = ""
16          self.yDetect = 115
```

**[11]**

As one of the **buttons** on my Menu page, I used an image called **glow.png** which is an illuminating, glowing box. I want to use this box to be in the foreground of whatever option the user is currently on. So if the user is on the **instructions** button on the **Menu Screen**, then there will be this glow behind it. Thus, I will successfully meet success criteria number 1.2.2.2: If a user uses W (up), S (down) keys to go on a button, the button should illuminate to indicate that that is the option the user is currently on.

By trial and error and placing the different buttons on the screen and seeing where they looked the best as well as analysing various menu screens on other fighting games, I decided to start the glow at y position 115 as shown on line 16 using the attribute **yDetect**. (B2)

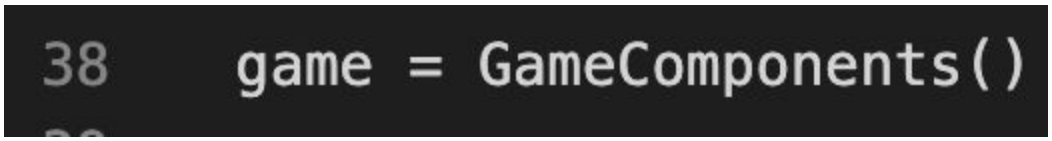
The x position of the glow will remain the same as all the buttons are in a vertical line about the centre of the page. As the user has pressed the spacebar, **screen_status** is now “menu”, therefore the **ELIF** statement on line 62 is True.

Indented inside this **ELIF** statement, I use another **IF** statement to say that if the **S** key is pressed then execute some more code. According to my success criteria (1.2.2.2), if the user presses the **S** key, the game should go DOWN. Therefore, I need the illuminating glow to go down to the next button to signify to the user that he is on the option below now. Therefore, using trial and error, I found that I should add 50 to **yDetect** (line 65).

On lines 66 and 67, I am saying that if **yDetect** ever gets bigger than 315, then make it equal to 315. In other words, if the user is on the last button on the **Menu Screen**, and he presses the **S** key, the illumination should not go down the page by 50 as there is no button there, so it should stay at 315 which is the y-position for the last button.

Similarly, on lines 69-73, I have done the same thing but just if the user presses the **W** button which, according to the success criteria, should be used for going UP.

I had two minor problems that occurred in this phase of my code. The first one can be seen in image **8** on lines 65 and 66. I used **self.yDetect** instead of using **game.yDetect**. (A5). I instantiated the **GameComponents()** class under the variable name **game**. **yDetect** is an attribute of the class **GameComponents()**. Therefore, when I am accessing this attribute outside the scope of the class (globally), I have to use the object instance of the class to access it, in other words, I have to use **game.yDetect** and not **self.yDetect**.




**[12]**

The second problem I had was that when I pressed the spacebar to go from the **Splash Screen** to the **Menu Screen**, it would go straight from the **Splash Screen** to the **Level 1 Screen**. This is because the frame rate was originally at 27 (image 5, line 20) and so I slowed it down to 2 frames per second (image 10, line 60) for when the user pressed the spacebar and similarly for the buttons, so that the game doesn't skip a button when the user presses **S** or **W**, I slowed the frame rate to 13 (image 10, line 64). This slower frame rate ensures that Python doesn't detect the spacebar, **S** or **W** buttons multiple times when the user has only pressed it once. (A6)

## Code 6: Condensing and making code more efficient (3)

```
1   import pygame
2
3   class GameComponents():
4       def __init__(self):
5           pygame.init()
6           pygame.display.set_caption("Game")
7           self.playGame = True
8           self.window = pygame.display.set_mode((600, 380))
9           self.splash_image = pygame.image.load("images/Backgrounds/Splash Screen/splash_screen.png").convert_alpha()
10          self.menu_image = pygame.image.load("images/Backgrounds/Menu/menu_final.png").convert_alpha()
11          self.button_names = ["start_down", "instructions_down", "highscore_down", "settings_down", "exit_down", "glow"]
12          self.buttons = []
13          self.screen_status = "splash"
14
15          self.menuChoice = ""
16          self.yDetect = 115
17
18          for i in range(len(self.button_names)):
19              self.buttons.append(pygame.image.load("images/Backgrounds/Menu/{}.png".format(self.button_names[i])).convert_alpha())
20          self.instructions_image = pygame.image.load("images/Backgrounds/Instructions/instructions.png").convert_alpha()
21
22          self.highscore_image = pygame.image.load("images/Backgrounds/Highscores/highscores.png").convert_alpha()
23          self.highscore_name = pygame.image.load("images/Backgrounds/Highscores/name.png").convert_alpha()
24          self.highscore_score = pygame.image.load("images/Backgrounds/Highscores/score.png").convert_alpha()
25          self.highscore_date = pygame.image.load("images/Backgrounds/Highscores/date.png").convert_alpha()
26
27          self.settings_image = pygame.image.load("images/Backgrounds/Settings/settings.png").convert_alpha()
28          self.settings_sound = pygame.image.load("images/Backgrounds/Settings/sound_on.png").convert_alpha()
29          self.settings_controls = pygame.image.load("images/Backgrounds/Settings/controls_wasd.png").convert_alpha()
30          self.settings_speed = pygame.image.load("images/Backgrounds/Settings/speed_medium.png").convert_alpha()
31          self.settings_annotations = pygame.image.load("images/Backgrounds/Settings/screen_annotation_on.png").convert_alpha()
32          self.settings_glow = pygame.image.load("images/Backgrounds/Settings/glow.png").convert_alpha()
```

**[13]**

Initially, my code was very jumbled and hard to read so I decided to take a stop here before I moved on to work on the efficiency, presentation and condensation of my code. My first objective was to sort out all the images I was going to use for my game. In the screenshot above, you can see that I sorted the images out in relation to the screen they were for. For example, I had the high score page images from lines 22-25 and the settings page images from lines 27-32. However, I thought it would make more sense to separate the background images of each screen from the buttons, subtitles and all the other images.

```
17      #loading background images:
18      self.splash_bgimage = pygame.image.load("images/Backgrounds/Splash Screen/splash_screen.png").convert_alpha()
19      self.menu_bgimage = pygame.image.load("images/Backgrounds/Menu/menu_final.png").convert_alpha()
20      self.instructions_bgimage = pygame.image.load("images/Backgrounds/Instructions/instructions.png").convert_alpha()
21      self.highscore_bgimage = pygame.image.load("images/Backgrounds/Highscores/highscores.png").convert_alpha()
22      self.settings_bgimage = pygame.image.load("images/Backgrounds/Settings/settings.png").convert_alpha()
23      self.level_1bg_image = pygame.image.load("images/Backgrounds/Levels/level_1.png").convert_alpha()
24
25      #array to store all the loaded buttons for the menu
26      self.menu_button_names = ["start_down", "instructions_down", "highscore_down", "settings_down", "exit_down", "glow"]
27      self.menu_buttons = []
28      for i in range(len(self.menu_button_names)):
29          self.menu_buttons.append(pygame.image.load("images/Backgrounds/Menu/{}.png".format(self.menu_button_names[i])).convert_alpha())
30
31      self.highscore_button_names = ["name", "score", "date"]
32      self.highscore_buttons = []
33      for i in range(len(self.highscore_button_names)):
34          self.highscore_buttons.append(pygame.image.load("images/Backgrounds/Highscores/{}.png".format(self.highscore_button_names[i])).convert_alpha())
35
36      self.settings_button_names = ["sound_on", "controls_wasd", "speed_medium", "screen_annotation_on", "glow"]
37      self.settings_buttons = []
38      for i in range(len(self.settings_button_names)):
39          self.settings_buttons.append(pygame.image.load("images/Backgrounds/Settings/{}.png".format(self.settings_button_names[i])).convert_alpha())
40
```

**[14]**

This is shown on lines 17-23 where I now had my background images and lines 25-39 where I had all my buttons and other images. Furthermore, I worked on the efficiency of my code. In the previous screenshot, you can see that I individually loaded each image for the Settings and High score screens. However, I decided to use a similar method to Code 5 (1) Bullet point 4 where I condensed my code for the Menu images. Instead of saving all my buttons individually to different variables and loading them separately, I thought it would be more effective to make a list of all the button names for each respective screen (lines 31 and 36), then iterate through a range of the number of items in each list (lines 33 and 38), then at each iteration, write the code needed to load an image from the **Highscores or Settings** folder on my laptop and append the name of the button that I want to load using **.format** and using **self.button_names[i]** to append the name of the button at that iteration. Finally, I needed to append this whole line into a new list called **self.buttons**. This was a much more effective and efficient method to load all the images for the Highscores and Settings screen.

```
17      #loading background images:
18
19      self.bgimage_names = ["splash_screen", "menu", "instructions", "highscores", "settings", "level_1"]
20      self.bgimages = []
21      for i in range(len(self.bgimage_names)):
22          self.bgimages.append(pygame.image.load("images/Screens/Backgrounds/{}.png".format(self.bgimage_names[i])).convert_alpha())
23
```

**[15]**

My next task was to not individually load all the background images. So using the same method as above, I iterated and appended to create a much more efficient method to load all my background images.

```
17    #loading background images:
18
19    self.bgimages = [pygame.image.load("images/Screens/Backgrounds/{}.png".format(i)).convert_alpha() for i in ["splash_screen",
20
21    #array to store all the loaded buttons for the menu
22
23    self.menu_buttons = [pygame.image.load("images/Screens/Menu/{}.png".format(i)).convert_alpha() for i in ["start_down", "instr
24
25    self.highscore_buttons = [pygame.image.load("images/Screens/Highscores/{}.png".format(i)).convert_alpha() for i in ["name", "
26
27    self.settings_buttons = [pygame.image.load("images/Screens/Settings/{}.png".format(i)).convert_alpha() for i in ["sound_on",
28
```

**[16]**

One point I later realised was that I can simply use list comprehension in all 4 of these cases (for the background, menu, high score and settings images). Therefore, on line 19 for example, I created the attribute **self.bgimages** and used list comprehension to do the same job as the code on lines 19-22 in the screenshot before. This actually turned out to be more efficient as I was no longer creating two attributes (as you can see on lines 19 and 20 in screenshot 15) but only one (screenshot 16 line 19) as I just passed in the list of all the names of the images instead of creating a separate attribute for it.

## Code 6: Condensing and making code more efficient

```
39      def splashDisplay(self):
40          self.window.blit(self.splash_image, (0,20))
41      def menuDisplay(self):
42          self.window.blit(self.menu_image, (-25,-30))
43          self.window.blit(self.buttons[0], (205,120))
44          self.window.blit(self.buttons[5], (163,self.yDetect))
45          self.window.blit(self.buttons[1], (200,165))
46          self.window.blit(self.buttons[2], (183,215))
47          self.window.blit(self.buttons[3], (240,265))
48          self.window.blit(self.buttons[4], (273,315))
49      def instructionsDisplay(self):
50          self.window.blit(self.instructions_image, (0,10))
51      def highscoresDisplay(self):
52          self.window.blit(self.highscore_image, (0,10))
53          self.window.blit(self.highscore_name, (170,110))
54          self.window.blit(self.highscore_score, (270,110))
55          self.window.blit(self.highscore_date, (370,108))
56      def settingsDisplay(self):
57          self.window.blit(self.settings_image, (0,10))
58          self.window.blit(self.settings_sound, (182,120))
59          self.window.blit(self.settings_glow, (118,self.yDetect))
60          self.window.blit(self.settings_controls, (190,165))
61          self.window.blit(self.settings_speed, (165,215))
62          self.window.blit(self.settings_annotations, (135,265))
63      def levelDisplay(self):
64          self.window.blit(self.level_1bg_image, (0,0))
```

[17]

Lines 39 to 64 are dedicated to displaying the images and buttons of the various screens. In each case, I have applied the **.blit** method to the instance object **window** and passed in two arguments; the image that I want to be displayed and the position of the window.

I thought about a way to condense this piece of code, however I couldn't find enough areas of commonality to be able to use a shorter, more efficient method. This is due to the fact that the x-coordinates of the images in each method have no correlation between them.

However, this is not the case for the **highdcoresDisplay** method. On the High Score page, the subtitles are all on one row. This means that the c-coordinates increment by a constant amount and their y-coordinates are already the same. Therefore, I decided to make a more efficient piece of code for the **highdcoresDisplay** method.

```
46      def highscoresDisplay(self):
47          game.window.blit(self.bgimages[3], (0,10))
48          x = 30
49          for i in self.highscore_subtitles:
50              x += 100
51              game.window.blit(i, (x,110))
```

[18]

After trial and error and determining at what x-coordinate my subtitles of “rank”, “name”, “score” and “date” would look good on the page, I found that it should be at 130, 230, 330 and 430 respectively. Thus, on line 48 I assign the integer value 30 to the local variable **x**. Then on line 49, I iterate through an attribute of this class called **highscore_subtitles** which has the images of all the subtitles stored. Then, as the x-coordinate of the first image I want is 130, the value of x is updated to 130 and the first image at coordinates (130, 110) is displayed. A similar process will happen for the second, third and fourth iterations.

(B3) The idea behind changing the code into this format was to increase efficiency as well as to create an easier way to add an extra subtitle if needed. This was actually the case. Initially I just had the “name”, “score” and “date” headers on my high Score page. Then after reviewing my code with an end-user, it was brought to my attention that I needed a “rank” header as well. This is essential to a High Score page. Consequently, with this new code, it was very easy to simply add the new subtitle image for “rank” into the attribute **highscore_subtitles** and then it would simply display it.

## Code 6: Condensing and making code more efficient (2)

```
67  def callthis(y_coord, end_point, screen_name):
68      if keys[pygame.K_s]:
69          game.clock.tick(5)
70          game.yDetect += y_coord
71          if game.yDetect >= end_point:
72              game.yDetect = end_point
73
74      elif keys[pygame.K_w]:
75          game.clock.tick(5)
76          game.yDetect -= y_coord
77          if game.yDetect <= 115:
78              game.yDetect = 115
79
80      if screen_name == "menu":
81          if game.yDetect == 115 and keys[pygame.K_RETURN]:
82              game.screen_status = "level 1"
83          elif game.yDetect == 165 and keys[pygame.K_RETURN]:
84              game.screen_status = "instructions"
85          elif game.yDetect == 215 and keys[pygame.K_RETURN]:
86              game.screen_status = "highscores"
87          elif game.yDetect == 265 and keys[pygame.K_RETURN]:
88              game.screen_status = "settings"
89          elif game.yDetect == end_point and keys[pygame.K_RETURN]:
90              game.playGame = False
```

**[19]**

```
122
123     else:
124         keys = pygame.key.get_pressed()
125         if keys[pygame.K_ESCAPE]:
126             game.playGame = False
127         if game.screen_status == "splash" and keys[pygame.K_RETURN]:
128             game.clock.tick(5)
129             game.screen_status = "menu"
130         elif game.screen_status == "menu":
131             callthis(50, 315, "menu")
132         elif game.screen_status == "settings":
133             callthis(50, 265, "settings")
134
```

**[20]**

Firstly, I need to explain what the attribute **yDetect** encompasses. If you look in image **[17]**, the methods **settingsDisplay** and **menuDisplay** have this attribute **yDetect**. These two screens are the only screens in the game (so far) with buttons or options for the user to choose. These buttons are displayed vertically on the screen with the same amount of space between each button. According to my success criteria number 1.2.2.2, "If a user uses W (up), S (down) keys to go on a button, the button should illuminate to indicate that that is the option the user is currently on**.**" Therefore, for both my Settings and Menu page, I created an additional glow image that would surround the button chosen by the user as an indicator. If the user pressed **S** (down), then this glow should go down to the next button and **W** (up) should make the glow go up a button.

On line 67, I create a function for my illuminating indicator (the name **callthis** was just temporary and is changed later on). Note how I have not passed in self as it is not a class method but just a function. On line 68, I use an **IF** statement to see if the user has pressed the **S** button. If this is true, then the code will continue to line 69. On line 69, I slow the frame rate of the game due to a similar problem that occurred earlier. Please refer to (A6) earlier in the Implementation section under "Code 5: Getting the first few screens of my game to run (3)"

On line 70, I take **y_coord** away from **yDetect** as I want to decrease the y-coordinate of my illumination to simulate the user going down a button. I initially have my attribute **yDetect** assigned to the integer value 115 (see image **[13]** line 16). I have chosen this number through seeing at what y-coordinate will the first button look aesthetically pleasing at. Thus, I concluded that the first button on the Settings and Menu page will be at y-coordinate 115 and so the first illumination will be at 115. Then when the user presses **S**, the illumination will go down to the next button ny decreasing in y-position by a fixed value, **y_coord**. If you look in image **[20]** lines 130-133, where I call this function, I have set **y_coord** for both the Settings and Menu page to be 50 as each button on these pages are 50 y-coordinates apart from each other.

Then lines 71 and 72 say that if **yDetect** becomes bigger than the **end_point**, then make **yDetect** equal **end_point.** As you can see again in image **[20]** lines 130-133 where I call this function, I have set the **end_point** for the Menu page as 315 and 265 for the Settings page. This is simply because the Menu page has one more button than the Settings page so the **end_point** will be 50 y-coordinates further down. Lines 71 and 72 are implemented to ensure that if the user presses the **S** (down) button when he is already on the last button, the illumination should not go down another 50 y-coordinates but should stay on the last button.

Then from lines 74-78 I have repeated the same process but for going up a button by pressing **W**.

```
69      if screen_name == "menu":
70          select_screens = {115: "level 1", 165: "instructions", 215: "highscores", 265: "settings", 315: "exit"}
71          if keys[pygame.K_RETURN]:
72              game.screen_status = select_screens[game.y_selector]
73          if game.screen_status == "exit":
74              game.playGame = False
```

**[21]**

My next task was to condense the extremely inefficient code on Lines 80-90 (image **[19]**). The main purpose of these lines of code is to basically direct the user to the next screen. For example, If they press return at Y coordinate 115 then take them to the first level. It seemed quite intuitive to me to use a dictionary. Hence, I create a dictionary on line 70 called **select_screens** and set the keys at each index as the Y coordinate of the various buttons and the values are set as the name of the screen at the respective Y coordinates. So, at index 0 of the dictionary the key is **115** and the value is **level 1** which is in accordance with what I had coded in the non-condensed version.

Then on line 71, I introduce a simple **IF** statement asking whether the user has pressed the **RETURN** button on their keyboard. If so, then line 72 is executed.

**screen_status** is an attribute of the class instantiated under the name **game** that I use countless times in my code. If I set it to a certain string such as the ones set as the values in the dictionary on line 70, then the program will blit the respective screen. So changing this attribute is what will change what the user will see (see lines 127-133 image **[20]** to see **screen_status** in action and using it to blit the respective screens). As you can see on line 72, I set **screen_status** to call the dictionary using the key **game.y_selector**. **y_selector** is also an attribute of the **game** class and is simply set to the Y coordinate of the aforementioned in-game glow. Essentially this **IF** block is saying that if the user presses **RETURN** then key search the dictionary at whatever Y coordinate the user is at and find the corresponding value. Then assign this value to the attribute **screen_status** which will then blit the new screen.

The **IF** block on lines 73-74 says that if the value returned from the dictionary is **exit** then the class attribute **playGame** will be set to boolean **False** causing the game to quit.

## Code 7: Working on the Menu page buttons

```
67    def buttonSelectMenu(self, y_coord, end_point):
68        if keys[pygame.K_s]:
69            game.clock.tick(5)
70            self.y_selectorMenu += y_coord
71            if self.y_selectorMenu >= end_point:
72                self.y_selectorMenu = end_point
73        elif keys[pygame.K_w]:
74            game.clock.tick(5)
75            self.y_selectorMenu -= y_coord
76            if self.y_selectorMenu <= 115:
77                self.y_selectorMenu = 115
78        if keys[pygame.K_RETURN]:
79            self.screen_status = {115: "level 1", 165: "instructions", 215: "highscores", 265: "settings", 315: "exit"}[self.y_selectorMenu]
80        if self.screen_status == "exit":
81            game.playGame = False
```

**[22] ^^**

```
156   while game.playGame:
157       game.clock.tick(27)
158
159       if screen.screen_status == "splash":
160           screen.splashDisplay()
161       elif screen.screen_status == "menu":
162           screen.menuDisplay()
163       elif screen.screen_status == "instructions":
164           screen.instructionsDisplay()
165       elif screen.screen_status == "highscores":
166           screen.highscoresDisplay()
167       elif screen.screen_status == "settings":
168           screen.settingsDisplay()
169       elif screen.screen_status == "level 1":
170           print(screen.selectedSound)
171           print(screen.selectedControl)
172           print(screen.selectedSpeed)
173           print(screen.selectedAnnotation)
174           screen.levelDisplay()
```

**[23]**

```
177       else:
178           keys = pygame.key.get_pressed()
179           if keys[pygame.K_ESCAPE]:
180               game.playGame = False
181           if screen.screen_status == "splash" and keys[pygame.K_RETURN]:
182               game.clock.tick(5)
183               screen.screen_status = "menu"
184           elif screen.screen_status == "menu":
185               screen.buttonSelectMenu(50, 315)
186           elif screen.screen_status == "settings":
187               screen.buttonSelectSettings(50, 265)
```

**[24]**


**[25]**

On line 67, I created the class method **buttonSelectMenu**. I have previously explained the workings of lines 68-77 under “Code 6: Condensing and making code more efficient” under images **[19]** and **[20]**. The only difference is that in this new version, I have renamed the attribute **y_detect** to **y_selectorMenu** so that in the future I can also create a **y_selectorSettings** for the y-coordinates of the buttons of the respective screens.

Liines 78-81, I actually implement the action of a button being chosen on the **Menu** screen. Line 78 starts with an **if** statement saying that if the **RETURN** key on the keyboard is pressed then to follow the indented code. This fulfils success criteria number 1.2.2.3 which says, pressing the “RETURN key on an illuminated button should select it.”

On line 79, I bring up this important **screen_status** attribute. As you can see in image **[22]** lines 159-169, depending on what string name the attribute **screen_status** is assigned to, the respective method will be called to display that screen. So on line 79, I assign a dictionary to the **screen_status** with the keys as the y-coordinates of the buttons shown in image **[23]** and the values of the dictionary as string names that will be assigned to **screen_status** and in turn, displaying the chosen screen.

This works because I index the dictionary with the class attribute **y_selectorMenu** which starts at 115 and has the ability to assume all the y-coordinates of each of the buttons on the **Menu** screen. (This process is explained further in “Code 6: Condensing and making code more efficient” under images **[19]** and **[20]**” Therefore if say the integer **115** is indexed into the dictionary, it will return the value **“level 1”**. Consequently, the attribute **screen_status** will have the string **“level 1”** assigned to it and so the **levelDisplay** method will be called displaying the first level of the game. This is how the **Menu** screen option select feature works.

Finally, if **y_selectorMenu** equals 315, then 315 will be indexed into the dictionary and the value **“exit”** will be returned and assigned to **screen_status**. Then if this is the case, as laid out on line 80, the **game** class attribute **playGame** will be assigned the boolean value **False**, which in turn quits the program.

I call the **buttonSelectMenu** method in image **[23]** line 184-185. The workings of this have also been previously explained in “Code 6: Condensing and making code more efficient” under images **[19]** and **[20]**”

## Code 7: Working on the button change for the Settings page

**[26]**

```
75      if screen_name == "settings":
76          if keys[pygame.K_RETURN]:
77              game.screen_status = {115: "settings", 165: "settings", 215: "settings", 265: "settings"}[game.y_selector]
```

Once i had condensed the menu selection code using a dictionary, coding the first steps for the settings screen for the button selection followed a similar route. I thought that it would also make sense to use a dictionary. However there was a key difference with the settings screen in that if a user presses **RETURN** at a given Y coordinate, they should not be taken to another screen but instead their option should change. For example if the user presses **RETURN** at Y coordinate 115, it should change the **Sound** option from **ON** to **OFF**.

Thus, on line 77, for every key in the dictionary, the value is **settings**. This means that no matter what **y_selector** is, **screen_status** will always be set to **settings** so only the **settings** screen will ever blit and so the screen will never change. Now the only aspect of the **settings** screen left to code is the changing of the buttons.

**[27]**


I then went on to try this code, where I had an array as the value which I attempted to index with a 0 then when the user presses **RETURN** at a given Y coordinate, it would index the array for that key with a 1 instead. This seemed to work in theory, however, one button on the **settings** screen has 3 options and all the rest only have 2. This meant that if the user presses the **RETURN** button 3 times on a button that only has 2 options, an **index out of range** error will be thrown (A7).

```
37          self.optionChosenSounds = 0
38          self.settingIndexSounds = 0
39          self.selectedSound = self.settings_buttons_text[self.optionChosenSounds]
40
41
42          self.optionChosenControls = 2
43          self.settingIndexControls = 0
44          self.selectedControl = self.settings_buttons_text[self.optionChosenControls]
45
46          self.optionChosenSpeed = 5
47          self.settingIndexSpeed = 0
48          self.selectedSpeed = self.settings_buttons_text[self.optionChosenSpeed]
49
50          self.optionChosenAnnotation = 7
51          self.settingIndexAnnotation = 0
52          self.selectedAnnotation = self.settings_buttons_text[self.optionChosenAnnotation]
```

**[28]**

```
101     def buttonSelectSettings(self, y_coord, end_point):
102         if keys[pygame.K_s]:
103             game.clock.tick(4)
104             self.y_selectorSettings += y_coord
105             if self.y_selectorSettings >= end_point:
106                 self.y_selectorSettings = end_point
107 
108         elif keys[pygame.K_w]:
109             game.clock.tick(4)
110             self.y_selectorSettings -= y_coord
111             if self.y_selectorSettings <= 115:
112                 self.y_selectorSettings = 115
113 
114         elif keys[pygame.K_SPACE] and self.screen_status == "settings":
115             self.settingCoords = {115: [0,1], 165: [2,3], 215: [4,5,6], 265: [7,8]}
116             game.clock.tick(4)
117 
118             if self.y_selectorSettings == 115:
119                 self.settingIndexSounds +=1
120                 if self.settingIndexSounds >1:
121                     self.settingIndexSounds = 0
122                 self.optionChosenSounds = self.settingCoords[115][self.settingIndexSounds]
123                 self.selectedSound = self.settings_buttons_text[self.optionChosenSounds]
124 
125             if self.y_selectorSettings == 165:
126                 self.settingIndexControls +=1
127                 if self.settingIndexControls >1:
128                     self.settingIndexControls = 0
129                 self.optionChosenControls= self.settingCoords[165][self.settingIndexControls]
130                 self.selectedControl = self.settings_buttons_text[self.optionChosenControls]
131 
132             if self.y_selectorSettings == 215:
133                 self.settingIndexSpeed +=1
134                 if self.settingIndexSpeed >2:
135                     self.settingIndexSpeed = 0
136                 self.optionChosenSpeed = self.settingCoords[215][self.settingIndexSpeed]
137                 self.selectedSpeed = self.settings_buttons_text[self.optionChosenSpeed]
138 
139             if self.y_selectorSettings == 265:
140                 self.settingIndexAnnotation +=1
141                 if self.settingIndexAnnotation >1:
142                     self.settingIndexAnnotation = 0
143                 self.optionChosenAnnotation= self.settingCoords[265][self.settingIndexAnnotation]
144                 self.selectedAnnotation = self.settings_buttons_text[self.optionChosenAnnotation]
```

**[29]**

**To do ^^**

## Code 8: Animating Tizoc's movement

```
69      def standingAnimation(self, game):
70          if self.standingCount == 35:
71              self.standingCount = 0
72
73          if self.facing == "right":
74              game.window.blit(self.standingImagesRight[self.standingCount//5], (self.x_coord, self.y_coord))
75              self.standingCount += 1
76
77          if self.facing == "left":
78              game.window.blit(self.standingImagesLeft[self.standingCount//5], (self.x_coord, self.y_coord))
79              self.standingCount += 1
80
81
82      def walkingAnimation(self, game):
83          if self.walkingCount == 40:
84              self.walkingCount = 0
85              self.is_idle = True
86              self.is_walking = False
87
88          if self.facing == "right":
89              game.window.blit(self.walkingImagesRight[self.walkingCount//5], (self.x_coord, self.y_coord))
90              self.walkingCount += 1
91
92          if self.facing == "left":
93              game.window.blit(self.walkingImagesLeft[self.walkingCount//5], (self.x_coord, self.y_coord))
94              self.walkingCount += 1
```

**[30]**

On line 69 I create the class method **standingAnimation** and pass in the class **game** by value. I construct a similar code on line 82 with the class method **walkingAnimation**. I had 7 images for my standing animation and wanted the character to move at a steady and slow pace. Therefore, I calculated that each image would be displayed for 5 counts. Hence on line 70, I have the conditional statement asking whether the count has reached 35 (7 images X 5). If it has reached 35, i.e each image has been displayed for 5 counts, then the attribute **standingCount** is reset to 0 and all the images are displayed for a second time in a cycle indefinitely. On line 73, I ask whether the attribute facing is **"right"** and if so, then all the right facing standing images are displayed else all the left ones. Finally, in each case, if **facing** is **"right"** or **"left"**, the attribute **standingCount** goes up by one. And as you can see, every time the count goes up by one, the algorithm asks whether it is equal to 35 and will repeat this process till the count is equal to 35 and then reset it back to 0. An identical process is being carried out in the class method below. Hence, there is significant repetition in my code, meaning that it is quite inefficient. This is how I improved and optimised my code:

(A8) I had a problem where when the user stops pressing the keyboard button **d** (right movement) or **a** (left movement), then the character would display the entirety of its last cycle of walking images before it starts to display its standing images. What should be the case is that as soon as the player stops pressing **d** or **a**, i.e wanting the character to stop walking, the walking images should stop being displayed immediately and the standing images should proceed to display. In order to fix this problem, I implemented this code:

```
71      def standingAnimation(self, game, xVal):
72          if self.standingCount == len(self.standingImagesRight)*3:
73              self.standingCount = 0
74              self.is_idle = True
75              self.is_walking = False
76
77          if self.facing == "right":
78              self.x_coord += xVal
79              game.window.blit(self.standingImagesRight[self.standingCount//3], (self.x_coord, self.y_coord))
80              self.standingCount += 1
81
82          if self.facing == "left":
83              self.x_coord -= xVal
84              game.window.blit(self.standingImagesLeft[self.standingCount//3], (self.x_coord, self.y_coord))
85              self.standingCount += 1
86
87
88      def walkingAnimation(self, game, xVal):
89          if self.walkingCount == len(self.walkingImagesRight)*3:
90              self.walkingCount = 0
91              self.is_idle = True
92              self.is_walking = False
93
94          if self.facing == "right":
95              self.x_coord += xVal
96              game.window.blit(self.walkingImagesRight[self.walkingCount//3], (self.x_coord, self.y_coord))
97              self.walkingCount += 1
98
99          if self.facing == "left":
100             self.x_coord -= xVal
101             game.window.blit(self.walkingImagesLeft[self.walkingCount//3], (self.x_coord, self.y_coord))
102             self.walkingCount += 1
```

**[31]**

I added the code on lines 95 and 100 using the new parameter **xVal**. I am fundamentally saying that the **x_coord** or the x coordinate of the character should continue to change even after the user stops pressing the right and left movement keys. So instead of the character having that awkward phase where it is standing but the walking animation images are playing, it will simply just carry on in the direction it is facing for the duration of the last cycle of its walking images and then come to a clean halt and start displaying the standing images. The parameter passed in by value, **xVal**, is set to being **tizoc.velocity*game.gameSpeed** for the walking animation to allow the character to carry on walking until the cycle of its walking images are complete. However, for the **standingAnimation** class method, the parameter passed in by value is set to the integer value **0** as we want no change in the x coordinate when the character is just standing.

```
71      def characterAnimation(self, game, xVal, imagesArrayLeft, imagesArrayRight, idleStatus, walkingStatus):
72          if self.statusCount == len(imagesArrayLeft)*3:
73              self.statusCount = 0
74              self.is_idle = idleStatus
75              self.is_walking = walkingStatus
76
77          if self.facing == "left":
78              game.window.blit(imagesArrayLeft[self.statusCount//3], (self.x_coord, self.y_coord))
79              self.statusCount += 1
80
81          if self.facing == "right":
82              game.window.blit(imagesArrayRight[self.statusCount//3], (self.x_coord, self.y_coord))
83              self.statusCount += 1
84
85          if self.facing == "right":
86              self.x_coord += xVal
87          if self.facing == "left":
88              self.x_coord -= xVal
```

**[32]**

My next task was to further condense the walking and standing animations. I combined both class methods into a singular class method called **characterAnimation**. There were a couple of areas of commonality between both class methods which I could generalise and condense. For example, on lines 72-75, I create an attribute called **self.statusCount** and make it equal to the walking and standing count. Furthermore, I pass in the “idleStatus” and the “walkingStatus” by value form the **mainProgram** when I am calling the **characterAnimation** method for standing and walking respectively. It is as simple as, when I want the character to perform all its standing images, I pass in tizoc.characterAnimation(game, 0, tizoc.standingImagesLeft, tizoc.standingImagesRight, True, False) to tailor the class method to standing and a similar idea for when I require the character to walk.

However, when getting all the standing images to show, there was a gap in the images where nothing showed. An index out of range error was thrown. I traced the algorithm and rearranged the count += 1 to sort out the index issue.

```
3   class Character():
4       def __init__(self, velocity, x_coord, y_coord, facing, height, width, health):
5           self.velocity = velocity
6           self.x_coord = x_coord
7           self.y_coord = y_coord
8           self.facing = facing
9           self.height = height
10          self.width = width
11          self.health = health
12          self.is_dead = False
13          self.is_idle = True
14          self.is_kicking = False
15          self.is_punching = False
16          self.is_walking = False
17          self.is_jumping = False
18          self.is_blocking = False
19          self.walkingImagesLeft = []
20          self.walkingImagesRight = []
21          self.walkingCount = 0
22          self.standingImagesLeft = []
23          self.standingImagesRight = []
24          self.standingCount = 0
25          self.kickingImagesLeft = []
26          self.kickingimagesRight = []
27          self.kickingCount = 0
28          self.punchingimagesLeft = []
29          self.punchingImagesRight = []
30          self.punchingCount = 0
31          self.jumpingimagesLeft = []
32          self.jumpingimagesRight = []
33          self.jumpingCount = 0
34          self.blockingImagesLeft = []
35          self.blockingImagesRight = []
36          self.blockingCount = 0
37          self.deadImagesLeft = []
38          self.deadImagesRight = []
39          self.deadCount = 0
40          self.specialMoveImages1Left = []
41          self.specialMoveImages1Right = []
42          self.is_special_move_1 = False
43          self.special_move_count_1 = 0
44          self.walkingSound = []
45          self.kickingSound = []
46          self.punchingSound = []
47          self.blockingSound = []
48          self.specialMoveSound = []
49          self.statusCount = self.standingCount = self.walkingCount
```

Initially we had the count += 1 line before the blitting of the character. However, this is clearly problematic as it would undoubtedly lead to an “index out of range error” as the count became too large and so when we floor the count it is indexing the images at an index which does not exist. So swapping these two lines means that first it will blit the character image then it will increase the count and then the key point is that it will check if the new count value equals the conditional statement before blitting another character image at said count index. Furthermore, the use of “elif” instead of an “if” statement caused the algorithm to only read some of the lines of code, however, we want the algorithm to check against all conditions, hence the use of “if” was required.

**[33]**

```
229         elif screen.screen_status == 'level 1':
230             screen.levelDisplay()
231             if tizoc.is_idle == True:
232                 tizoc.characterAnimation(game, 0, tizoc.standingImagesLeft, tizoc.standingImagesRight, True, False)
233 
234             if tizoc.is_walking == True:
235                 tizoc.characterAnimation(game, tizoc.velocity * game.gameSpeed, tizoc.walkingImagesLeft, tizoc.walkingImagesRight, True, False)
236 
237             if keys[pygame.K_d]:
238                 tizoc.is_idle = False
239                 tizoc.is_walking = True
240                 tizoc.facing = "right"
241 
242             elif keys[pygame.K_a]:
243                 tizoc.is_idle = False
244                 tizoc.is_walking = True
245                 tizoc.facing = "left"
246 
247         if (screen.screen_status != "menu" and screen.screen_status != "splash") and keys[pygame.K_BACKSPACE]:
248             screen.screen_status = "menu"
```

**[34]**

This code was in my **mainProgram** and is used to call the character animation class methods in my **Sprites** file. Line 229 is a simple **ELIF** statement continuing on from a few **ELIF** statements before this asking essentially what screen the game is on. So, if the **screen_status** class attribute is set to "**level 1**" then call the **levelDisplay** class method to blit the first level.

Then the animation code is run. Line 231 asks whether the character **Tizoc** is standing completely still. I created various different attributes of the **Character** class such as: **is_idle**, **is_walking** etc. Basically just asking what state of movement a given character is in. As you can see on line 231, I ask if **tizoc.is_idle** was True. **tizoc** is the instantiation of my **Tizoc** class which inherits from my **Character** class. So if **tizoc.is_idle** is **True** then it calls the **characterAnimation** method inside the **Tizoc** class with various parameters.

The first parameter used is **game**. **game** is an instantiation of the **GameComponents** class which I created for all the building blocks of my game. This includes the **window** attribute. Now since **GameComponents** or **game** was in my **mainProgram** and Tizoc's **characterAnimation** was in my **Sprites** file, I had to pass in **game** by value in order to use the **window.blit** function with Tizoc.

The second parameter is velocity and since Tizoc is idle, he has 0 velocity. The third and fourth parameters are the left and right images of a character which are again passed in by value. I will shortly explain why I have differentiated them like this. Finally, the fifth parameter is **idleStatus** and the sixth **walkingStatus**. As this **IF** block is concerning Tizoc being idle, **idleStatus** is set as **True** and **walkingStatus** as **False**.

However, I quickly realised adding more parameters for every move would be extremely inefficient as I would have to add in **jumpingStatus** then **blockingStatus** and so on and pass them all in by value. I have explained under ( ) where I moved these True and False boolean statements as they are crucial for the animation to work smoothly. From lines 234-235, I repeat this process but for Tizoc if he is in a walking state.

Lines 237-245 is where I code the link between the keyboard and performing a certain move. The **IF** block on line 237 is for when the user presses **d** and the **ELIF** block below is for if the user presses **a**. So, if the user presses **d**, then I want him to move in the right direction in accordance with my success criteria number 5.1.2. Hence, I logically set **is_idle** to **False** and **is_walking** to **True** and I simply set the **facing** attribute of Tizoc to "**right**". This **facing** attribute is then used in character animation in an **IF** statement to determine whether to blit the left or right facing images. I then repeat this code for when the user presses **a** but I set **facing** as "**left**" instead.

```
229         elif screen.screen_status == 'level 1':
230             screen.levelDisplay()
231 
232             #calling characterAnimation
233             if tizoc.is_idle == True:
234                 tizoc.characterAnimation(game, 0, tizoc.standingImagesLeft, tizoc.standingImagesRight, True, False, False)
235 
236             if tizoc.is_walking == True:
237                 tizoc.characterAnimation(game, tizoc.velocity * game.gameSpeed, tizoc.walkingImagesLeft, tizoc.walkingImagesRight, True, False, False)
238 
239             if tizoc.is_blocking == True:
240                 tizoc.characterAnimation(game, 0, tizoc.blockingImagesLeft, tizoc.blockingImagesRight, True, False, False)
241             #Character moving extremely fast when I press "j" and then gives an index out of range error
242 
243             # controls
244             if keys[pygame.K_d]:
245                 tizoc.is_idle = False
246                 tizoc.is_walking = True
247                 tizoc.facing = "right"
248 
249             elif keys[pygame.K_a]:
250                 tizoc.is_idle = False
251                 tizoc.is_walking = True
252                 tizoc.facing = "left"
253 
254             elif keys[pygame.K_j]:
255                 tizoc.is_idle = False
256                 tizoc.is_walking = False
257                 tizoc.is_blocking = True
258 
```

**[35]**

(A9) My method for calling the walking and idle (standing) images for Tizoc seemed to work. However, I missed out a crucial point. When I added code for Tizoc's blocking images, an index out of range error occurred. This is because I was displaying the blocking and walking images at the same time. There was a point in my code where both **is_blocking** and **is_walking** were **True**. This then meant that both sets of left and right images for both blocking and walking were being passed in. And since there were a different number of images corresponding to these moves, the count used in **characterAnimation** became extremely jumbled and the maths used no longer worked (for more on the maths, see the first paragraph under "Code 8: Working on Tizoc's walking animation")

```
73      def characterAnimation(self, game, xVal, imagesArrayLeft, imagesArrayRight, idleStatus, walkingStatus, blockingStatus):
74          if self.statusCount == len(imagesArrayLeft)*3:
75              self.statusCount = 0
76              self.is_idle = idleStatus
77              self.is_walking = walkingStatus
78              self.is_blocking = blockingStatus
79 
80          if self.facing == "left":
81              game.window.blit(imagesArrayLeft[self.statusCount//3], (self.x_coord, self.y_coord))
82              self.statusCount += 1
83 
84          if self.facing == "right":
85              game.window.blit(imagesArrayRight[self.statusCount//3], (self.x_coord, self.y_coord))
Exception has occurred: IndexError
list index out of range
  File "/Users/zainm/Desktop/coursework/Ancient Combat (By Val Sorted)/lib/sprites.py", line 85, in characterAnimation
    game.window.blit(imagesArrayRight[self.statusCount//3], (self.x_coord, self.y_coord))
  File "/Users/zainm/Desktop/coursework/Ancient Combat (By Val Sorted)/lib/mainProgram.py", line 234, in <module>
    tizoc.characterAnimation(game, 0, tizoc.standingImagesLeft, tizoc.standingImagesRight, True, False, False)
```

**[36]**

```
229         elif screen.screen_status == 'level 1':
230             screen.levelDisplay()
231 
232             #calling characterAnimation
233             if tizoc.is_idle:
234                 tizoc.characterAnimation(game, 0, tizoc.standingImagesLeft, tizoc.standingImagesRight)
235 
236             if tizoc.is_walking and not tizoc.is_blocking:
237                 tizoc.characterAnimation(game, tizoc.velocity * game.gameSpeed, tizoc.walkingImagesLeft, tizoc.walkingImagesRight)
238 
239             if tizoc.is_blocking and not tizoc.is_walking:
240                 tizoc.characterAnimation(game, 0, tizoc.blockingImagesLeft, tizoc.blockingImagesRight)
241             #Character moving extremely fast when I press "j" and then gives an index out of range error
242 
243             # controls
244             if keys[pygame.K_d]:
245                 tizoc.is_idle = False
246                 tizoc.is_walking = True
247                 tizoc.facing = "right"
248 
249             elif keys[pygame.K_a]:
250                 tizoc.is_idle = False
251                 tizoc.is_walking = True
252                 tizoc.facing = "left"
253 
254             elif keys[pygame.K_j]:
255                 tizoc.is_idle = False
256                 tizoc.is_blocking = True
257 
258 
259         if (screen.screen_status != "menu" and screen.screen_status != "splash") and keys[pygame.K_BACKSPACE]:
260             screen.screen_status = "menu"
```

**[37]**

This was my first attempt at solving the problem. As I knew it was the matter of the **characterAnimation** class method being called multiple times with different parameters, I added some extra conditions to the **IF** statements on lines 236 and 239. This was to ensure that the **characterAnimation** class method was only being called at one point at any given time in Tizoc's movement. However, this was not the way to go about this problem. If I only want one **characterAnimation** class method to be called at one point, then there is a much simpler way of doing this.

```
229         elif screen.screen_status == 'level 1':
230             screen.levelDisplay()
231 
232             #calling characterAnimation
233             if tizoc.is_idle:
234                 tizoc.characterAnimation(game, 0, tizoc.standingImagesLeft, tizoc.standingImagesRight)
235 
236             elif tizoc.is_walking:
237                 tizoc.characterAnimation(game, tizoc.velocity * game.gameSpeed, tizoc.walkingImagesLeft, tizoc.walkingImagesRight)
238 
239             elif tizoc.is_blocking:
240                 tizoc.characterAnimation(game, 0, tizoc.blockingImagesLeft, tizoc.blockingImagesRight)
241             #Character moving extremely fast when I press "j" and then gives an index out of range error
242 
243             # controls
244             if keys[pygame.K_d]:
245                 tizoc.is_idle = False
246                 tizoc.is_walking = True
247                 tizoc.facing = "right"
248 
249             elif keys[pygame.K_a]:
250                 tizoc.is_idle = False
251                 tizoc.is_walking = True
252                 tizoc.facing = "left"
253 
254             elif keys[pygame.K_j]:
255                 tizoc.is_idle = False
256                 tizoc.is_blocking = True
```

**[38]**

My solution was to just use an **IF** statement and then all the conditional statements below will be **ELIF**s. This meant that the program will check one by one down the code to see if the conditional statement is satisfied. If the first **IF** statement is not **True**, then the code will check with the next **ELIF** statement then the next and so on and so on. This was the most effective way to solve my index out of range problem.

## Code 8: Animating Tizoc's movement (Jumping)

```
82      def characterAnimation(self, game, xVal, imagesArrayLeft, imagesArrayRight):
83
84          if self.statusCount == len(imagesArrayLeft)*3:
85              self.statusCount = 0
86              self.is_idle = True
87              self.is_walking = False
88              self.is_blocking = False
89              self.is_kicking = False
90              self.is_punching = False
91
92          if self.facing == "left":
93              game.window.blit(imagesArrayLeft[self.statusCount//3], (self.x_coord, self.y_coord))
94              self.x_coord -= xVal
95              self.statusCount += 1
96
97          if self.facing == "right":
98              game.window.blit(imagesArrayRight[self.statusCount//3], (self.x_coord, self.y_coord))
99              self.x_coord += xVal
100             self.statusCount += 1
101
102         if self.is_jumping:
103             if self.jumpingCount >= 0:
104                 self.y_coord -= (self.jumpingCount**2) * 0.3
105                 self.jumpingCount -= 1
106             elif self.jumpingCount >= -10:
107                 self.y_coord += (self.jumpingCount**2) * 0.3
108                 self.jumpingCount -= 1
109             else:
110                 self.jumpingCount = 10
111                 self.is_jumping = False
112                 self.statusCount = 0
```

**[39]**

(A10) I originally had a problem where randomly the character would disappear after he had performed a jump move. This was because there was a moment in time where no state was True. In other words, there was a period in my code where **is_jumping** and **is_idle** were False. This meant that no images were being displayed, hence the disappearing character. To solve this problem, I added a statement in the else block in my code reiterating that **self.is_idle = True**. This did in fact work, however temporarily. This is because it eliminated that period in time where no state was True, however, conversely, it created a period of time where **is_jumping** and **is_idle** were both True together. This then resulted in an index out of range error as both jumping images and standing (idle) images were being inputted into the class method causing an index out of range error. I then realised that to inhibit this error from occurring, I just needed to add a statement resetting **statusCount** to 0 so that this collision doesn't occur.

Another issue I had was to do with not being able to move and jump at the same time. I realised that while in the air, the character could move left and right. This meant that he was able to jump and move but not move and jump. To solve this, I simply changed the order in which I placed my elif and if statements in my mainProgram.

# Test Table

**Key**

Purple Text = Success Criteria I have not yet completed

1. Setting up the game
    1.1. Setting up the game window

| Test No. | Test | SC No. | SC (Success Criteria) | Actual Outcome Pass/Fail? | Action- Any improvements needed |
|---|---|---|---|---|---|
| 1 | Move and click the mouse to see if any interference occurs and press the “x” button to see if that quits the game. | 1.1.1 | Mouse not to be used except for the ability to press the red “x” button to close the game | Pass | |
| 2 | I chose a screen size of 600 by 380 pixels. So the test was to see if it would load. | 1.1.2 | Reasonable window size | Fail | Yes, improvements needed - please refer to the implementation section under (A1) and (A2) |
| 3 | I chose a screen size of 600 by 380 pixels. So the test was to see if it would load. | 1.1.2 | Reasonable window size | Pass | No, improvement not needed |
| 4 | See if the programme quits when the ESC button is pressed on any page. | 1.1.3 | Press ESC anywhere to quit the program | Fail | Yes, improvements needed - please refer to the implementation section under (A3) |
| 5 | See if the programme quits when the ESC button is pressed on any page. | 1.1.3 | Press ESC anywhere to quit the program | Pass | No, improvement not needed |
| 6 | | 1.1.4 | BACKSPACE to go to the previous screen. | Pass | |

1.2. Planning out the various game screens and their order
    1.2.1. Splash Screen

| | | | | | |
|---|---|---|---|---|---|
| 1 | Test to see when the code is run if the **Splash** screen is the first page that appears. | 1.2.1.1 | First screen of the game | Fail | Yes, improvement needed - please refer to the implementation section under (A4) |
| 2 | Test to see when the code is run if the **Splash** screen is the first page that appears. | 1.2.1.1 | First screen of the game | Pass | No, improvement not needed |
| 3 | | 1.2.1.2 | Background music | | |
| 4 | Test to see when the game is run whether the main characters and name of the game appear clearly | 1.2.1.3 | Main characters and name of the game clearly presented. | Pass | |

| | | | | | |
|---|---|---|---|---|---|
| 5 | See if there's a clear instruction on how to progress to the next screen. | 1.2.1.4 | Indicator on how to start the game. | Pass | |

### 1.2.2. Menu Page

| | | | | | |
|---|---|---|---|---|---|
| 1 | See if it is clear to the user what the different options on the **Menu** page are. | 1.2.2.1 | Clear options to navigate throughout the game. "Start Game", "Instructions", "View Highscores", "Settings" and "Exit". | Pass | |
| 2 | Test to see if W and S make the illuminating indicator go up and down. | 1.2.2.2 | If a user uses W (up), S (down) keys to go on a button, the button should illuminate to indicate that that is the option the user is currently on. | Pass | |
| 3 | Test to see if pressing RETURN on an illuminated button takes you to its respective page. | 1.2.2.3 | RETURN key on an illuminated button should select it. | Pass | |
| 4 | | 1.2.2.4 | Different background music. | | |

### 1.2.3. Settings Page

| | | | | | |
|---|---|---|---|---|---|
| 1 | Test to see if pressing RETURN on the **Settings** button takes you to the **Settings** page. | 1.2.3.1 | If RETURN is pressed on the **Settings** button, display the **Settings** page. | Fail | Yes, improvement needed - please refer to the implementation section under (A5) and (A6) |
| 2 | Test to see if pressing RETURN on the **Settings** button takes you to the **Settings** page. | 1.2.3.1 | If RETURN is pressed on the **Settings** button, display the **Settings** page. | Pass | No, improvement not needed |
| 3 | See if all the changeable options are displayed on the **Settings** page and that they can actually be changed. | 1.2.3.2 | Display all options the user can change. Sound effects on/off, controller settings (WASD to arrow keys), game speed (slow, medium or fast), screen annotation on/off. | Fail | Yes, improvement needed - please refer to the implementation section under (A7) |
| 4 | See if all the changeable options are displayed on the **Settings** page and that they can actually be changed. | 1.2.3.2 | Display all options the user can change. Sound effects on/off, controller settings (WASD to arrow keys), game speed (slow, medium or fast), screen annotation on/off. | Pass | No, improvement not needed |

### 1.2.4. High Score Page

| | | | | | |
|---|---|---|---|---|---|
| 1 | Test to see if pressing RETURN on the **View High Scores** button takes you to the **High Scores** page. | 1.2.4.1 | If RETURN is pressed on the **View High Scores** button, display the **High Scores** page. | Pass | |
| 2 | | 1.2.4.2 | Display top 5 scores highest to lowest. | | |
| 3 | | 1.2.4.3 | Rank, Score, Name (3 letters), Date (e.g 13/09/2020) subtitles. | | |

### 1.2.5. Instructions Page

| | | | | | |
|---|---|---|---|---|---|
| 1 | Test to see if pressing RETURN on the **Instructions** button takes you to the **Instructions** page. | 1.2.5 .1 | If RETURN is pressed on the **Instructions** button, display the **Instructions** page. | Pass | |
| 2 | See if the buttons for each move and navigation are displayed clearly on the **Instructions** page. | 1.2.5 .2 | Navigation keys and game moves displayed as well as in-game controls. | Pass | |

2. Character Selection

| | | | | | |
|---|---|---|---|---|---|
| 1 | Test to see if pressing RETURN on the **Start Game** button takes you to the **Character Select** page. | 2.1 | If RETURN is pressed on the **Start Game** option, display the **Character Select** page. | Pass | |
| 2 | See if the two main characters are visually and clearly represented on the screen. | 2.2 | Main characters to choose from must be clearly shown. | Pass | |
| 3 | See if the arrow moves up (using W) and down (using S) to indicate what character you are currently selecting. | 2.3 | Move an arrow up and down (using W and S) to select a character. | Pass | |
| 4 | | 2.4 | Different background music. | | |

3. Game Logistics

| | | | | | |
|---|---|---|---|---|---|
| 1 | | 3.1 | 360 second timer. The user has 360 seconds to defeat 21 enemies to move on to the next level. | | |
| 2 | | 3.2 | Every 15 seconds a new enemy appears. | | |
| 3 | | 3.3 | 100 health points to begin with.<br>Enemies have 20 health points to begin with. | | |
| 4 | | 3.4 | Enemies have 20 health points to begin with. | | |
| 5 | | 3.5 | Enemies must always follow you. | | |
| 6 | | 3.6 | All users' scores will be recorded and saved to a database. | | |
| 7 | | 3.7 | Different sound effects for each character. | | |

4. In-game Screen Layout

| | | | | | |
|---|---|---|---|---|---|
| 1 | Test to see that when a character is chosen, the **level 1 background image** loads. | 4.1 | Once the character is chosen, display the **level 1 background image**. | Pass | |
| 2 | Test to see if the chosen character displays on the **level 1 image**. | 4.2 | Display chosen character. | Pass | |
| 3 | | 4.3 | Timer on the top middle of the screen, health bar in the top left, kill count in the top right. Special move bar underneath health bar. Level indicator under the timer. | | |
| 4 | | 4.4 | Enemy's health and special move bars on top of their heads. | | |
| 5 | See if the obstacles created display on the screen and if they can be picked up and used by the characters. | 4.5 | Bins and boxes scattered throughout the game to be used as a projectile or to destroy. | Fail | Yes, improvement needed - please refer to the implementation section under (A13) |
| 6 | See if the obstacles created display on the screen and if they can be picked up and used by the characters. | 4.5 | Bins and boxes scattered throughout the game to be used as a projectile or to destroy. | Pass | No, improvement not needed |

## 5. Progressing in the game

### 5.1. Moves

| | | | | | |
|---|---|---|---|---|---|
| 1 | See if the character animates standing still | 5.1.1 | Standing | Fail | Yes, improvement needed - please refer to the implementation section under (A11) |
| 2 | See if the character animates standing still | 5.1.1 | Standing | Pass | No, improvement not needed |
| 3 | See if the character animates walking | 5.1.2 | Left and right movement | Fail | Yes, improvement needed - please refer to the implementation section under (A8) |
| 4 | See if the character animates walking | 5.1.2 | Left and right movement | Pass | No, improvement not needed |
| 5 | See if the character animates punching | 5.1.3 | Punching | Pass | |
| 6 | See if the character animates kicking | 5.1.4 | Kicking | Pass | |
| 7 | See if the character animates jumping | 5.1.5 | Jumping | Fail | Yes, improvement needed - please refer to the implementation section under (A10) |
| 8 | See if the character animates jumping | 5.1.5 | Jumping | Pass | No, improvement not needed |
| 9 | See if the character animates blocking | 5.1.6 | Blocking | Fail | Yes, improvement needed - please refer to the implementation section under (A9) |

| | | | | | |
|---|---|---|---|---|---|
| 10 | See if the character animates blocking | 5.1.6 | Blocking | Pass | No, improvement not needed |
| 11 | See if the character animates his special move | 5.1.7 | Special Move | | Yes, improvement needed - please refer to the implementation section under (A12) |
| 12 | See if the character animates his special move | 5.1.7 | Special Move | Pass | No, improvement not needed |

### 5.2. Interactions within the game

| | | | | | |
|---|---|---|---|---|---|
| 1 | | 5.2.1 | Landing a punch or kick on the enemy loses them 5 health points. Special move on the enemy deducts 10 health points. Throwing an obstacle at the enemy deducts 5 health points. | | |
| 2 | | 5.2.2 | If an enemy lands a punch or kick, you lose 2 health points. Special move loses you 6 health points. | | |
| 3 | | 5.2.3 | Enemies cannot pick up obstacles. | | |
| 4 | | 5.2.4 | If anyone blocks, no damage is deducted. | | |
| 5 | | 5.2.5 | Every 10 health points decrease, the health bar will be updated to display the new health. | | |

### 5.3. Special Move

| | | | | | |
|---|---|---|---|---|---|
| 1 | | 5.3.1 | Special move bar with 5 segments. Landing a punch, a kick, blocking an enemy's attack must fill the special move bar by a segment. | | |
| 2 | | 5.3.2 | Special move uses up all 5 segments. | | |
| 3 | | 5.3.3 | Enemies' special move bar has 4 segments. Landing a punch or a kick must fill the special move bar by a segment. | | |

### 5.4. Progressing onto the next level

| | | | | | |
|---|---|---|---|---|---|
| 1 | | 5.4.1 | If the user is successful in defeating the 21 enemies in the time given, then the next level will begin. 5 levels in total. | | |
| 2 | | 5.4.2 | The next wave of enemies will have 5 more health than the previous level and deduct 2 more damage in all attacks. The time allowed for that level will increase by 15 seconds. | | |
| 3 | | 5.4.3 | If you complete all 5 levels, there will be an end credit and you will return to the **Menu** page. | | |

### 5.5. Dying in the game

| | | | | | |
|---|---|---|---|---|---|
| 1 | | 5.5.1 | If the user is unsuccessful, then a **GAME OVER** message will display and return the user back to the Menu screen. | | |

# AQA A-LEVEL COMPUTER SCIENCE COURSEWORK:
## ANCIENT COMBAT

**Zain Mobarik**
**13 CS1**