Games Individual Report – Leane Matejko

Overview

Introduction

This report covers the development of the game 'Circus' using the simplegui library and vector class. Throughout this report, there will be mentions of the concepts learned within the theory portion of the modules such as using collisions, vectors and sprite sheets.

Game Description

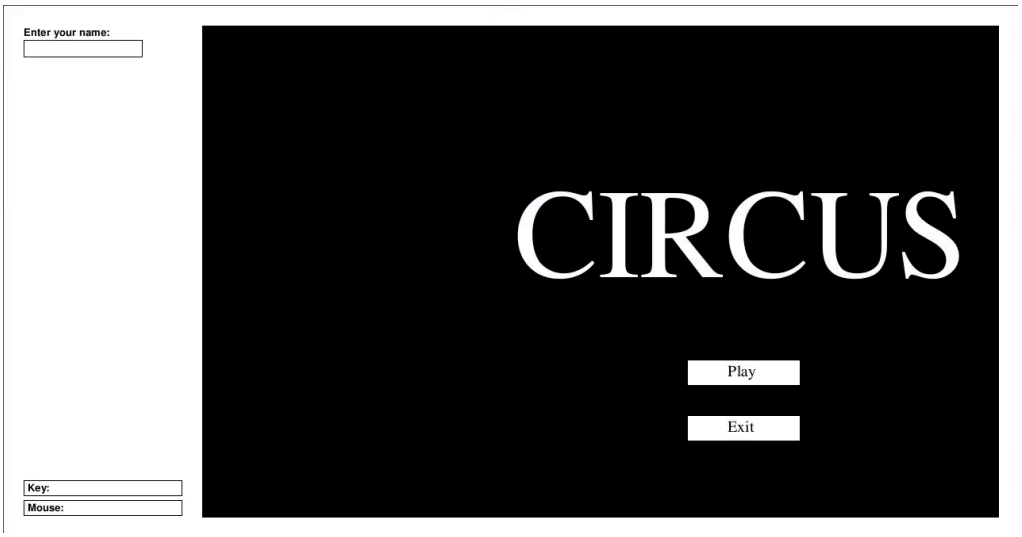
The general story of the game is the player have been kidnapped by the local Ringmaster and forced to act as a clown. In your pursuit for freedom, you face the members of the troupe, having to battle them to continue. You as the player must defeat your troupe members, leading to your escape. With a deck of cards, players are able to shoot them using the arrow keys.

User manual

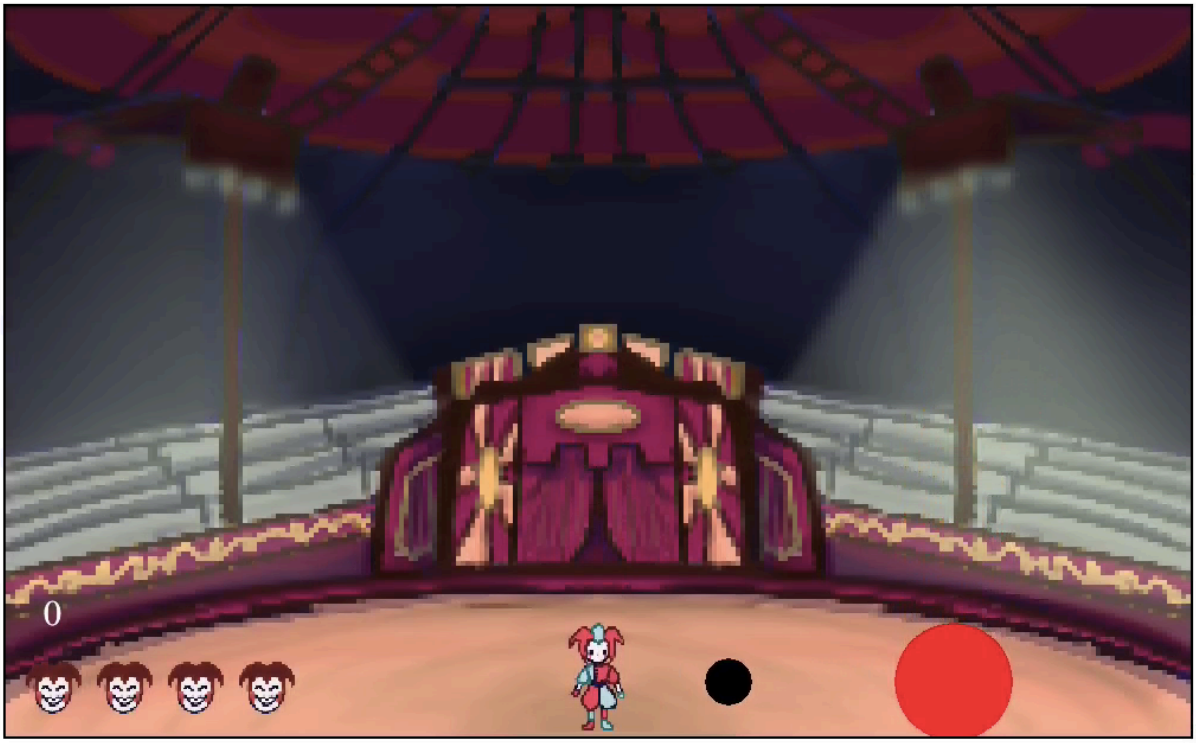
How to play:

- To open the game, users will need to download all files and makes sure that the 'SimpleGUICS2Pygame.simpleguics2pygame' file is accessible to the folder.
- Find the game file called 'Game.py', and run the file.
- Use your mouse to navigate around the window, any later controls will be displayed on the left hand panel.
    - You can enter your name to the input textbox on the panel, just make sure to press enter after.

Once deciding to click 'Play', the dialogue screen will appear. Follow the direction on the left panel of the screen. If you wish to skip through the dialogue, press and hold space, choosing options where appropriate.



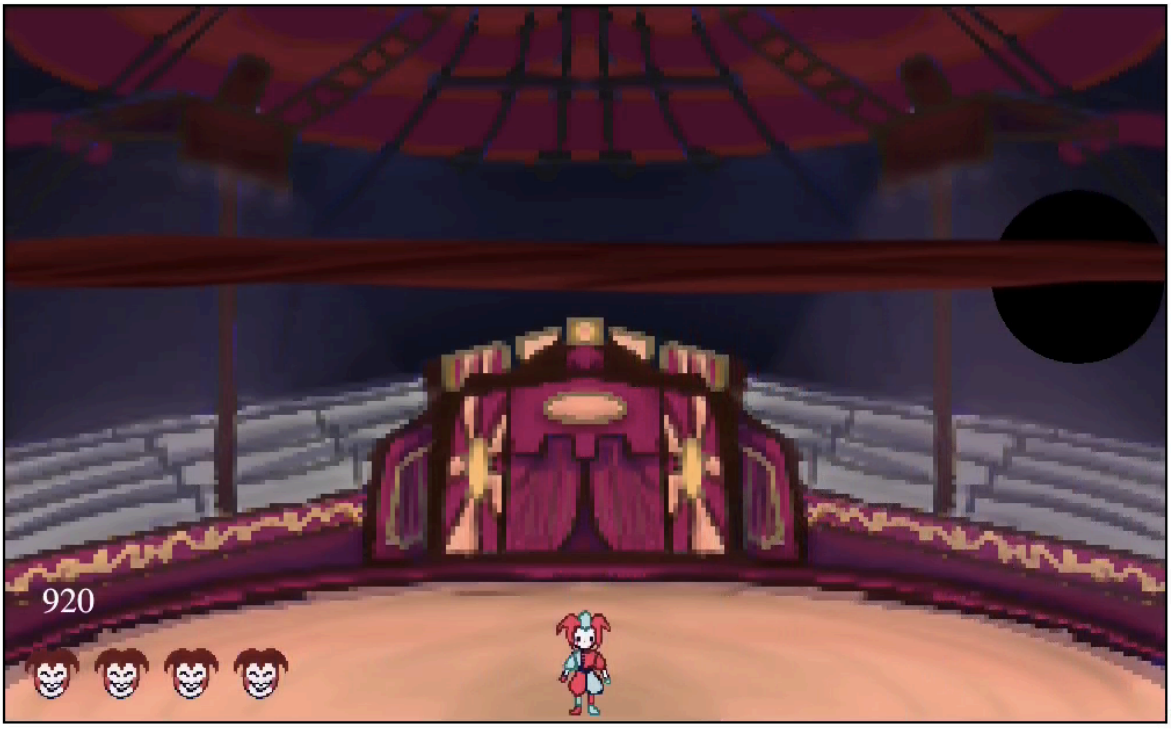
Once the dialogue has been completed, the fighting portion will begin. Here there will be the Ventriloquist attempting to attack you, so you must use A, W and D to control the player and the right arrow key to shoot the cards. Your lives and points will be shown within the lower left on the screen, with a damaging hit giving you 10 points and a critical hit giving 30 points.



After defeating the Ventriloquist, he will run off screen and begin phase 2. Continue to shoot after him while dodging his next attack. Although he can't be seen, damage can be done and points gained.

If at any stage, you lose all of your health and die, you will be taken to the death screen, where you will be shown your points and given the option to exit or replay. Replaying will bring you back to the start of the fight scene, with no points and full health.

If you manage to defeat the Ventriloquist a second time, you will be brought to the second fight scene where you will face the Aerialists. Your points and hearts will remain the same as the previous round.



Unlike the Ventriloquists, the Aerialists will be on a 6 second timer that changes their actions, acting on a cycle of three; attacking with lyra hoops, then silks and finally webs. Being hit with any of these will result in the player losing a life.

If you die you will be shown your points and presented with the option of either playing again or exiting. Playing again will bring the player back to the first fight scene.

If the players win, they will be taken to the Victory window, where they will be thanked for playing and shown their points. They will then have the option to play again (bring them back to the first fighting scene) or exit.

Key concepts
SimpleGUI is an interface that takes a combination of tkinter (a GUI used to create desktop applications/widgets), Qt (used for event loops and allowing for code to be cross-platform), WxPython(an alternative for tkinter) and Remi (translates python code to HTML). The combination of these GUIs allows for many varieties of programs that are able to interact with the user, draw images or shapes to the screen, add timers, input fields, all contained within another window.

Circus greatly relies on the mechanics of vectors and applying them for all movement of the player, enemies and their weapons. Most act like projectiles, having a speed vector that will be added to the initial position they were shot from. There are many calculations that can be done (e.g., having a weightless object vs one that has gravity applied), allowing for a form of game physics.

Colliders allow for the objects to detect other actions or objects present within the scene. Colliders have been used in conjunction with the vector class, and can change the behaviour of the object accordingly.
An example would be the green rectangle drawn around the player.

In order to have all the individual visual assets available to be shown, a series of spritesheets were used to animate the player. Loading a series of images from either through the internet or file location would cause the program performance to eventually decrease. Instead, a singular image with all animation frames is used. The current frame will be periodically changed, providing the illusion of an animation. These will be drawn to a position vector, allowing for the sprites to move around the screen animated.

Code Description

There are three windows within the game; the start window, death window and ending window. They all share a group of the same functions, however present different information. All windows draw a visual to the canvas and have play button and exit button. They also use a mouse handler that tracks the position of mouse's coordinates that are compared to the boundaries of the buttons drawn on screen. The start window starts off the whole game loop, moving onto the dialogue scene. Whereas the death and victory

windows only continue the loop from the fighting portion. All exit conditions close the window.
The rest of the code is responsible for drawing the visuals and returning condition variables, which are then used within the game loop.

```
import sys

class StartScene:
    def __init__(self, canvasWidth, canvasHeight, frame):
        self.canvasWidth = canvasWidth
        self.canvasHeight = canvasHeight

        self.startGameBool = False

        self.mouse = frame.set_mouseclick_handler(self.mouse_handler) #registering the mouse's coordinates within the canvas
        self.nameInput = frame.add_input('Enter your name: ', self.nameInput_handler, 150) #trying to get a name for the player

        self.position = [0,0] #coordinates for the mouse

        self.name = None #the name of the player

    def nameInput_handler(self, text_input): #setting the player's input as their name
        self.name = text_input

    def draw(self, canvas):
        self.update()
        #drawing the information to the screen
        canvas.draw_text(("CIRCUS"), (self.canvasWidth/2-110, self.canvasHeight/2), 160, "White")
        canvas.draw_polygon([(610,420),(750,420),(750,450),(610,450)], 1,'White', 'White')
        canvas.draw_text(("Play"), (660, 440), 20, "Black")
        canvas.draw_polygon([(610,490),(750,490),(750,520),(610,520)], 1,'White', 'White')
        canvas.draw_text(("Exit"), (660, 510), 20, "Black")

    def update(self):
        if self.position[0] > 610 and self.position[0] < 750 and self.position[1] > 420 and self.position[1] < 450: #checking if the play again button has been pressed
            self.startGameBool = True
        if self.position[0] > 610 and self.position[0] < 750 and self.position[1] > 490 and self.position[1] < 520: #checking if the exit button has been pressed
            sys.exit() #closing the window

    def mouse_handler(self, position): #setting the current position of the mouse
        self.position = position

    def startGame(self): #returning whether game will be started
        return (self.startGameBool, self.name)
```

```
class DeathScreen:
    def __init__(self, points, canvasWidth, canvasHeight, frame):
        self.canvasWidth = canvasWidth
        self.canvasHeight = canvasHeight

        self.startGameBool = False

        self.closeGameBool = False

        frame.set_mouseclick_handler(self.mouse_handler) #registering the mouse's coordinates within the canvas

        self.points = points #points that player recieved

        self.position = [0,0] #coordinates for the mouse

    def draw(self, canvas):
        self.update()
        #drawing the information to the screen
        canvas.draw_text(("YOU ARE NOW"), (self.canvasWidth/2-120, self.canvasHeight/2-50), 80, "White")
        canvas.draw_text(("THE JOKE"), (self.canvasWidth/2-50, self.canvasHeight/2), 80, "White")
        canvas.draw_polygon([(610,420),(750,420),(750,450),(610,450)], 1,'White', 'White')
        canvas.draw_text(("Play Again"), (635, 440), 20, "Black")
        canvas.draw_polygon([(610,490),(750,490),(750,520),(610,520)], 1,'White', 'White')
        canvas.draw_text(("Exit"), (660, 510), 20, "Black")
        canvas.draw_text(("They're laughing at you now"), (50, 510), 20, "White")
        canvas.draw_text(("Only " + str(self.points)+ " points"), (50, 530), 20, "White")

    def update(self):
        if self.position[0] > 610 and self.position[0] < 750 and self.position[1] > 420 and self.position[1] < 450: #checking if the play again button has been pressed
            self.startGameBool = True
        if self.position[0] > 610 and self.position[0] < 750 and self.position[1] > 490 and self.position[1] < 520: #checking if the exit button has been pressed
            self.closeGameBool = True

    def mouse_handler(self, position): #setting the current position of the mouse
        self.position = position

    def startGame(self): #returning whether game will be restarted
        return self.startGameBool

    def closeGame(self): #returning whether the game will be closed
        return self.closeGameBool
```

```
class EndingScreen:
    def __init__(self, points, canvasWidth, canvasHeight, frame):
        self.canvasWidth = canvasWidth
        self.canvasHeight = canvasHeight

        self.startGameBool = False

        self.closeGameBool = False

        self.background = "https://www.cs.rhul.ac.uk/home/zlac239/circusTitleScreen.jpg"

        frame.set_mouseclick_handler(self.mouse_handler) #registering the mouse's coordinates within the canvas

        self.points = points #points that player recieved

        self.position = [0,0] #coordinates for the mouse

    def draw(self, canvas):
        self.update()
        #drawing the information to the screen
        canvas.draw_text(("Thank you for playing!! You recieved " + str(self.points) + " points!!"), (self.canvasWidth/2, self.canvasHeight/2+100), 20, "White")
        canvas.draw_polygon([(680,450),(820,450),(820,480),(680,480)], 1,'White', 'White')
        canvas.draw_text(("Play Again"), (705, 470), 20, "Black")
        canvas.draw_polygon([(680,520),(820,520),(820,550),(680,550)], 1,'White', 'White')
        canvas.draw_text(("Exit"), (730, 540), 20, "Black")

    def update(self):
        if self.position[0] > 680 and self.position[0] < 820 and self.position[1] > 450 and self.position[1] < 480: #checking if the play again button has been pressed
            self.startGameBool = True
        if self.position[0] > 680 and self.position[0] < 820 and self.position[1] > 520 and self.position[1] < 550: #checking if the exit button has been pressed
            self.closeGameBool = True

    def mouse_handler(self, position): #setting the current position of the mouse
        self.position = position

    def startGame(self): #returning whether game will be restarted
        return self.startGameBool

    def closeGame(self): #returning whether the game will be closed
        return self.closeGameBool
```

As mentioned before, the spritesheet class is used to provide animations to the game. The draw method finds the particular section of the spritesheet as the current frame. Next_frame() finds the next frame, adding a delay and slowing the animation down. Some sprites are not meant to be repeatedly animated and should stop once reaching the end, so they will be stopped and reset.

```
import SimpleGUICS2Pygame.simpleguics2pygame as simplegui

class Spritesheet:
    def __init__(self, imageURL, sheetWidth, sheetHeight, columns, rows, xPos,yPos, numFrames, spriteWidth, spriteHeight, looping, clock):
        self.image = simplegui.load_image(imageURL)

        self.sheetWidth = sheetWidth
        self.sheetHeight = sheetHeight
        self.columns = columns
        self.rows = rows

        self.spriteWidth = sheetWidth/columns
        self.spriteHeight = sheetHeight/rows
        self.spriteCentreX = self.spriteWidth/2
        self.spriteCentreY = self.spriteHeight/2
        self.frameInd = [0,0]
        self.sourceDim = (self.spriteWidth, self.spriteHeight)

        self.x = xPos
        self.y = yPos

        self.currentFrame = (self.x, self.y)

        self.scale = 2
        self.xScale = self.scale*spriteWidth
        self.yScale = self.scale*spriteHeight

        self.spriteSize = (self.xScale,self.yScale)
        self.spritePosition = (xPos, yPos)

        self.clock = clock

        self.looping = looping

        self.currentImage = 0 #tracking the current image that we're on
        self.maxFrames = numFrames #the final frame of the incomplete spritesheet

    def draw(self, canvas, x, y):
        #drawing the sprite to the canvas while the spritesheet has not been finished
        # setting up the current frame of the sprite
        self.currentFrame = (self.spriteWidth * self.frameInd[0] + self.spriteCentreX, self.spriteHeight * self.frameInd[1] + self.spriteCentreY)
        self.spritePosition = (x, y)
        # drawing the sprite to the canvas
        canvas.draw_image(self.image, self.currentFrame, self.sourceDim, self.spritePosition, self.spriteSize)

    def next_frame(self):
        if self.done() == True: #checking if the animation is finished
            if self.clock.transition(15): #adding a delay
                self.reset() #resets the values of the spritesheet

        # if the sprite has not reached the final frame
        if self.done() == False:
            self.currentImage += 1 #keeping track of the current frame
            # moving along the sprite frame appropriately
            self.frameInd[0] = (self.frameInd[0] + 1) % self.columns
            if self.frameInd[0] == 0: #resetting the index frame of the animation
                self.frameInd[1] = (self.frameInd[1] + 1) % self.rows
```

```
    def done(self):
        if self.looping == True: #preventing the animation from stopping
            return False
        # checking whether the sprite has reached its final frame
        if self.currentImage == self.maxFrames+1:
            return True
        else:
            return False

    def reset(self):
        self.frameInd = [0,0]
        self.currentImage = 0
```

In order to create the dialogue scene, a series of information is needed to fed into a main program, which can then draw the corresponding scene to the screen. Therefore, there are three major components to the code; the dialogue sprites, the dialogue scene information and the actual main code to run and interact with the scene.

All classes ending DialSprites are all the objects for the character's information. Here you will would find their name, textbox colour and a series of expressions. Unfortunately, due to timing the expressions are just a circle with text, instead of being animated. The array would have been used to store the spritesheet objects and instantiated when needed for the scene, as a draw method would have been added as well.

```
class DDDialSprites:
    def __init__(self):
        self.name = "DareDevil" #for the nametag

        self.colour = "#de4201" # the colour of the textbox

        self.currentImage = 0 #the current image to be drawn to the screen

        self.dDConfident = "Confident" # Case 0
        self.dDSmug = "Smug" # Case 1
        self.dDAnnoyed = "Annoyed" # Case 2
        self.dDAngry = "Angry" # Case 3
        self.dDismissive = "Dismissive" # Case 4

        self.ImageList = [self.dDConfident, self.dDSmug, self.dDAnnoyed, self.dDAngry, self.dDismissive] #list of all the daredevil sprites
```

The DialogueOptions class is responsible for storing the information about each part of the scenes. The information changes format depending on; if it is for a menu, if there only one person in the scene.
For the menu, its formatted is [is menu, number of choices, dialogue text, option 1 text, option 2 text].
For one person, the format is [is menu, only one character onscreen, dialogue text, character object 1, character expression, textbox colour]
And then for two people, the format is [is menu, only one character onscreen, dialogue text, character object 1, character object 2, character expression 1, character expression 2, textbox colour].

```
class DialogueOptions:
    def __init__(self,name):
        #all sprite options
        self.playerSprite = PlayerDialSprites(name) #Main character dialogue sprite object
        self.RMSprite = RMDialSprites() #Ring Master dialogue sprite object
        self.DDSprite = DDDialSprites() #DareDevil dialogue sprite object
        self.LTSprite = LTDialSprites() #Lion Tamer dialogue sprite object
        self.CSprite = CDialSprites() #Contortionist dialogue sprite object
        self.VSprite = VDialSprites() #Ventriloquist dialogue sprite object

        #all scenes and required information
        self.dialogueList = [(False, True, "I look around to see other pedrestians minding their own business and going about their days.", self.playerSprite, self.playerSprite.ImageList[0], "White"),
        (False, True, "Am I just like them? Just another random person in this world? Will I ever amount to anything?", self.playerSprite, self.playerSprite.ImageList[0], "White"),
        (False, True, ("I am lost in thought. I trip over some rubbish that the lady on the corner left out days ago."), self.playerSprite, self.playerSprite.ImageList[2], "White"),
        (False, True, ("This is the third time this has happened this week... and now I'm covered in rubbish yet again."), self.playerSprite, self.playerSprite.ImageList[2], "White"),
        (False, True, ("And now I have no choice but to return back home and change. All I want to do is go to the shops."), self.playerSprite, self.playerSprite.ImageList[2], "White"),
        (False, True, ("The whole way home I fret about which clothes to change into. I really should be more aware of my surroundings..."), self.playerSprite, self.playerSprite.ImageList[1], "White"),
        (False, True, ("I continue walking, and accidentally bump into someone on the way."), self.playerSprite, self.playerSprite.ImageList[2], "White"),
        (False, True, ("Sorry! I really didn't mean it!"), self.playerSprite, self.playerSprite.ImageList[1], self.playerSprite.colour),
        (False, False, ("You should watch your step."), self.playerSprite,self.RMSprite, self.playerSprite.ImageList[1], self.RMSprite.ImageList[0], self.RMSprite.colour),
        (False, False, ("You're right. I'm sorry, I should have been walking backwards or something so I could see where I'm going."), self.playerSprite,self.RMSprite, self.playerSprite.ImageList[1], self.RMSprite.ImageList[0], self.playerSprite.colour),
        (False, False, ("I'll try to be better for next time."), self.playerSprite,self.RMSprite, self.playerSprite.ImageList[1], self.RMSprite.ImageList[0], self.playerSprite.colour),
        (False, True, ("What a rude man."), self.playerSprite, self.playerSprite.ImageList[2], "White"),
        (False, True, ("He walks off into the other direction. I hope I never see him again."), self.playerSprite, self.playerSprite.ImageList[0], "White"),
        (False, True, ("It's people like him who really get me annoyed. They don't care about other people."), self.playerSprite, self.playerSprite.ImageList[0], "White"),
        (False, True, ("I sigh. I'm getting too irritated over a little thing like that."), self.playerSprite, self.playerSprite.ImageList[0], "White"),
        (False, True, ("Although, if I was bumped into by someone covered in mysterious subtances, I'd be kind of annoyed too."), self.playerSprite, self.playerSprite.ImageList[3], "White"),
        (False, True, ("I guess it is a nice day out. It's nice to see so many friendly faces out and about today."), self.playerSprite, self.playerSprite.ImageList[5], "White"),
        (False, True, ("...Huh? Is that a baby crying? It's in an alleyway too..."), self.playerSprite, self.playerSprite.ImageList[3], self.playerSprite.colour),
        (True, 2, "Aproach the baby?", "Yes", "No"),
        (False, True, ("I should try and be smart about this."), self.playerSprite, self.playerSprite.ImageList[3], self.playerSprite.colour),
        (False, True, ("There is nothing not suspicious about a random baby in an alleyway. I'm not dumb."), self.playerSprite, self.playerSprite.ImageList[3], self.playerSprite.colour),
        (False, True, ("Yeah. Especially not after Tao got kidnapped. I'm not falling for these tricks."), self.playerSprite, self.playerSprite.ImageList[3], self.playerSprite.colour),
```

All of this then comes together within the DialogueScene class where it is presented to the user through the canvas. Initially, the update method checks to see if any updates have been made, checking for specific cases, for any jump statements or different valid inputs. If the end of the list has been reached, then the finished variable is set to true, causing the main game loop to move onto the fighting section.
Otherwise, then the information will be written to a series of variables, which will be later used later to draw everything to the game's canvas, depending on the information presented. Initially there was a delay using the clock class and transition method. However, this led to inconsistencies with input. A timer is now used instead.

```
#timer handler used to prevent the keyboard from being too responsive
def delay_handler(self):
    self.delayBool = False

def update(self):
    if (len(self.dialogueOptions.dialogueList) < self.currentDialogue): #checking whether the list containing all the scene information has reached the end
        self.finished = True #stopping the dialogue scene
    else:
        if (self.finished == False): #checking if we have made it to the end of the cutscene
            if self.delayBool == False: #adding a delay to the next button
                self.delayBool = True #restarting the delay
                if ((self.keyboard.next == True) or (self.keyboard.option1 == True) or (self.keyboard.option2 == True) ) and ((self.currentDialogue == 18) or (self.currentDialogue == 22) or (self.currentDialogue == 38)): #creating jump statement for specfic cases or if the us
                    if self.currentDialogue == 18: #option menu
                        if(self.keyboard.option1 == True): #option for yes
                            self.currentDialogue =23 #jumping to the continuation for this option of the story
                        elif(self.keyboard.option2 == True): #option for no
                            self.currentDialogue += 1
                            self.delay.start()
                    elif self.currentDialogue == 22: #reaching the end of option 1
                        self.currentDialogue = 34 #jumpging to the continuation of the story
                        self.delay.start()
                    elif self.currentDialogue == 38: #switching the scene of the character
                        self.place = "BackStage"
                        self.currentDialogue += 1
                        self.delay.start()
                else:
                    if(self.keyboard.next == True): # checking if the next button is pressed
                        self.currentDialogue += 1 #moving onto the next scene
                        self.delay.start()

                if (len(self.dialogueOptions.dialogueList) <= self.currentDialogue): #checking if the end of the list has been reached
                    self.delay.stop()
                    self.finished = True

                if(self.finished == False): #fetching the new information for the current scene
                    self.currentSceneInfo = self.dialogueOptions.returnDialogueInfo(self.currentDialogue) #setting the new scene
                    if(self.currentSceneInfo[0] == True): #if the scene is a menu
                        #setting up menu variables
                        self.menu = True #changing the scene format for a menu
                        if(self.currentSceneInfo[1] == 2):
                            self.dialogueText = self.currentSceneInfo[2]
                            self.option1 = self.currentSceneInfo[3]
                            self.option2 = self.currentSceneInfo[4]

                    else: #if the scene contains characters
                        self.menu = False #resetting the scene format
                        if(self.currentSceneInfo[1] == True): #checking how many people are in the scene
                        #setting up variables for one character
                            self.dialogueText = self.currentSceneInfo[2]
                            self.character1 = self.currentSceneInfo[3]
                            self.expression1 = self.currentSceneInfo[4]
                            self.character2 = None
                            self.textBoxColour = self.currentSceneInfo[5]
                        else:
                            #setting up variables for two characters
                            self.dialogueText = self.currentSceneInfo[2]
                            self.character1 = self.currentSceneInfo[3]
                            self.character2 = self.currentSceneInfo[4]
                            self.expression1 = self.currentSceneInfo[5]
                            self.expression2 = self.currentSceneInfo[6]
                            self.textBoxColour = self.currentSceneInfo[7]

def drawBase(self, canvas): #drawing the base for the scene for any characters present
    canvas.draw_polygon([(20,(self.canvasHeight)-25),((self.canvasWidth)-25,(self.canvasHeight)-25),((self.canvasWidth)-25,(self.canvasHeight)-185),(20,(self.canvasHeight)-185)], 1, self.textBoxColour, self.textBoxColour)
    canvas.draw_polygon([(20,(self.canvasHeight)-186),((self.canvasWidth)-750,(self.canvasHeight)-186),((self.canvasWidth)-750,(self.canvasHeight)-215),(20,(self.canvasHeight)-215)], 1,'Red', 'Red')
    canvas.draw_polygon([(20,50),(250,50),(250,20),(20,20)], 1,'Red', 'Red')
    canvas.draw_text(self.dialogueText,((25,(self.canvasHeight)-105)),20, 'Black')
    canvas.draw_text(self.place,((20,45)),30, 'Black')
    canvas.draw_text(self.character1.name,[20,425],30, 'Black')

def drawMenuBase(self, canvas): #drawing the base for the scene for a manu
    canvas.draw_polygon([(20,(self.canvasHeight)-25),((self.canvasWidth)-25,(self.canvasHeight)-25),((self.canvasWidth)-25,(self.canvasHeight)-185),(20,(self.canvasHeight)-185)], 1,'White', 'White')
    canvas.draw_text(self.dialogueText,((20,(self.canvasHeight)-105)),20, 'Black')
    if (self.currentSceneInfo[1] == 2): #drawing the scene for a menu with two options
        canvas.draw_circle(((self.canvasWidth/3)-30,(self.canvasHeight/4)*2), 60, 1, 'Blue', 'Red')
        canvas.draw_text(self.option1,(((self.canvasWidth/3)-50,(self.canvasHeight/4)*2)),30, 'Black')
        canvas.draw_circle(((self.canvasWidth/3)*2)-30,((self.canvasHeight/4)*2)), 60, 1, 'Red', 'Blue')
        canvas.draw_text(self.option2,(((self.canvasWidth/3)*2)-50,(self.canvasHeight/4)*2)),30, 'Black')
    canvas.draw_text(self.place,((20,45)),50, 'Black')

def draw(self,canvas):
    self.update()
    if(self.finished == False): # checking if the scene has finished
        if self.menu == False:
        #scene to draw for one character, only check for the second character
            if self.character2 == None: #for one character
                canvas.draw_circle((self.canvasWidth/2, self.canvasHeight/2), 50, 1, self.character1.colour, self.character1.colour) #drawing character1 to the scene
                canvas.draw_text(self.expression1,[(self.canvasWidth/2)-25,(self.canvasHeight/2)+5],20, 'Black')
                self.drawBase(canvas)
            #scene to draw for two character
            else: #for two characters
                #for character 1
                canvas.draw_circle((self.canvasWidth/3-60, self.canvasHeight/2), 60, 1, self.character1.colour, self.character1.colour) #drawing character1 to the scene
                canvas.draw_text(self.expression1,[(self.canvasWidth/3)-100,self.canvasHeight/2],20, 'Black')
                #for character 2
                canvas.draw_polygon[[(self.canvasWidth-255,(self.canvasHeight)-186),((self.canvasWidth)-25,(self.canvasHeight)-186),((self.canvasWidth)-25,(self.canvasHeight)-215),(self.canvasWidth-255,(self.canvasHeight)-215)], 1,'Red', 'Red') #
                canvas.draw_text(self.character2.name,[self.canvasWidth-245,(self.canvasHeight)-196],20, 'Black')
                canvas.draw_circle((((self.canvasWidth/3)*2)+50, self.canvasHeight/2), 60, 1, self.character2.colour, self.character2.colour) #drawing character2 to the scene
                canvas.draw_text(self.expression2,[((self.canvasWidth/3)*2)+20,self.canvasHeight/2],20, 'Black')
                #drawing the base
                self.drawBase(canvas)
        else: #for a menu
            self.drawMenuBase(canvas) #drawing the menu to the screen
    else:
        self.delay.stop()#stoping the timer
```

All bullet objects (Bullet, EnemyBullet, Lyra) other than the Silk and Web have largely the same function. They are used to store the particular bullet attributes (position, radius, speed) as well as the draw method and launch method. The Web object is similar as it has all previously mentioned features, however it can accelerate as it falls (by multiplying the speed vector by 1.05). The Silk class has two additional methods; getWithinSilk() that finds the hitbox of the silks, and reachedEnd() which checks if the silk has hit the end of the screen. The image sprite is called as an argument, to prevent each individual instance needing to load the image.

```python
import SimpleGUICS2Pygame.simpleguics2pygame as simplegui
from Spritesheet import Spritesheet

class Bullet:
    def __init__(self, launchPos, bulletSpeed, clock):
        self.launchPos = launchPos #the bullet current position
        self.radius = 5 #the size of the collider
        self.speed = bulletSpeed #the vector moving the bullet on the screen

        #the animated sprite used for the card
        self.bulletSprite = self.attackRight = Spritesheet("https://www.cs.rhul.ac.uk/home/zlac239/cardBullet.png", 320, 64, 5, 1, self.launchPos.x, self.launchPos.y, 5,16,  16, True, clock)
        self.clock = clock

    def launch(self):
        self.launchPos.add(self.speed) #changing the position of the bullet based on the direction it was shoot in

    def draw(self, canvas):
        if(self.clock.transition(5)): #creating a slight delay before switching to the next sprite
            self.bulletSprite.next_frame()
        self.bulletSprite.draw(canvas, self.launchPos.x, self.launchPos.y) #drawing the bullet to the screen
```

```python
class Silk:
    def __init__(self, launchPos, silkImage, canvasWidth = 1000, canvasHeight = 616):
        self.launchPos = launchPos #the bullet current position
        self.radius = 20
        self.speed = Vector(self.randNum(), 1)

        self.silkSprite = silkImage

        self.canvasWidth = canvasWidth
        self.canvasHeight = canvasHeight

    def launch(self):
        if(self.reachedEnd() == False):
            self.launchPos.add(self.speed) #changing the position of the bullet based on the direction it was shoot in#changing the position of the bullet based on the direction it was shoot in

    def getWithinSilk(self, playerPos, radius): #checking if something is within range of the silk
        if(playerPos.x + radius > self.launchPos.x and playerPos.x -radius < self.canvasWidth) and (playerPos.y + radius > self.launchPos.y - self.radius and playerPos.y - radius < self.launchPos.y - self.radius):
            return True
        return False

    def reachedEnd(self): #checking if the silk has reached the end of the screen
        if(self.launchPos.x <= 0):
            return True
        return False

    def randNum(self): #generating a random speed for the silk
        return random.randint(-15,-10)

    def draw(self, canvas): #drawing the silk image to the screen
        canvas.draw_image(self.silkSprite, (525,32), (1050,64), (self.launchPos.x,self.launchPos.y), (2500,64))
```

```python
class Web:
    def __init__(self, launchPos, webSprite):
        self.launchPos = launchPos #the bullet current position
        self.radius = 40
        self.speed = Vector(0,self.randNum(1,5))
        self.gravity = 1.05 #the acceleration factor

        self.webSprite = webSprite

    def launch(self):
        self.speed.multiply(self.gravity)# making the web accelerate towards the ground
        self.launchPos.add(self.speed) #changing the position of the bullet based on the direction it was shoot in

    def randNum(self, num1, num2): #generating a random number wiht a certain boundary
        return random.randint(num1,num2)

    def draw(self, canvas): #drawing the web sprite to the canvas
        canvas.draw_image(self.webSprite, (32,32), (64,64), (self.launchPos.x,self.launchPos.y), (128,128))
```

```python
class Lyra:
    def __init__(self, launchPos,  lyraSprite):
        self.launchPos = launchPos #the bullet current position
        self.radius = 40
        self.border = 1
        self.speed = Vector(self.randNum(),1) #each lyra has a random speed

        self.lyraSprite = lyraSprite #the lyra's sprite image

    def launch(self):
        self.launchPos.add(self.speed) #changing the position of the bullet based on the direction it was shoot in

    def randNum(self): #random number generator
        return random.randint(-5,-1)

    def draw(self, canvas): #drawing the lyra sprite to the canvas
        canvas.draw_image(self.lyraSprite, (64,64), (128,128), (self.launchPos.x,self.launchPos.y), (128,128))
```

The clock class is pretty straight forward, it is used to track intervals of time using a modulus calculation. Within the game, the clock is incremented every update cycle, which is then used to create delays for animations or shooting objects.

```
class Clock:
    def __init__(self):
        self.time = 0 #a counter used by the clock, incremented every game cycle

    def tick(self):
        self.time += 1 #incrementing the clock

    def transition(self, frameDuration): #controlling a delay, the longer the frame duration, the longer the transition time
        if self.time % frameDuration == 0:
            return True
        else:
            return False
```

The keyboard class is mainly just used to track the buttons used on the players keyboards. Once these buttons have been pressed, their variable will be set to true, which can be used for calculations within the scene classes.

```
import SimpleGUICS2Pygame.simpleguics2pygame as simplegui

class Keyboard:
    def __init__(self):
        self.right = False
        self.left = False
        self.jump = False
        self.damage = False
        self.shoot = False
        self.shootUp = False
        self.next = False
        self.option1 = False
        self.option2 = False
        self.option3 = False

    # a and d to move left and right, space to jump
    def keyDown(self, key):
        if key == simplegui.KEY_MAP['d']: # d - right
            self.right = True
        if key == simplegui.KEY_MAP['a']: # a - left
            self.left = True
        if key == simplegui.KEY_MAP['w']: #w - jump
            self.jump = True
        if key == simplegui.KEY_MAP['right']: #right - weapon
            self.shoot = True
        if key == simplegui.KEY_MAP['up']: #up - vertical weapon
            self.shootUp = True
        if key == simplegui.KEY_MAP['space']: #space - nextDialogueOption
            self.next = True
        if key == simplegui.KEY_MAP['1']: #1 - Dialogue Option 1
            self.option1 = True
        if key == simplegui.KEY_MAP['2']: #2 - Dialogue Option 2
            self.option2 = True
        if key == simplegui.KEY_MAP['3']: #2 - Dialogue Option 3
            self.option3 = True

    def keyUp(self, key):
        if key == simplegui.KEY_MAP['d']:
            self.right = False
        if key == simplegui.KEY_MAP['a']:
            self.left = False
        if key == simplegui.KEY_MAP['w']:
            self.jump = False
        if key == simplegui.KEY_MAP['right']:
            self.shoot = False
        if key == simplegui.KEY_MAP['up']:
            self.shootUp = False
        if key == simplegui.KEY_MAP['space']:
            self.next = False
        if key == simplegui.KEY_MAP['1']:
            self.option1 = False
        if key == simplegui.KEY_MAP['2']:
            self.option2 = False
        if key == simplegui.KEY_MAP['3']:
            self.option3 = False
```

The player class is responsible for tracking the player's position, health and projectile weapon. The update method moves any bullets within the weapon's array, removing any that have hit the enemy or have gone out of bounds. The player will then be moved using the speed vector (which has been set within the fight scene as it reacts to the input of the keyboard class). The player is also checked to be grounded, adding the gravity vector otherwise.
ShootBullet() and shootVerticalBullet() creates new bullet objects using the player's current position and direction, then adding them to the projectile weapon array, so they can be tracked with the update.

IsDead() checks the player's health to see if it has reached 0. This used within the game loop and fight scene to decide if the scene continues or shows the death window.
TakeDamage() is used by the enemy objects to hurt the player if the delay has been stopped. It will remove a heart object form the health array.
getPlayerHitBox() acts as the player's collider and can check if the player has interacted with anything.
Draw() is responsible for drawing any bullets or heart to the screen, as the actual player animations are handled within the PlayerSprites class.

```
import SimpleGUICS2Pygame.simpleguics2pygame as simplegui, math
from Vector import Vector
from Heart import Heart
from Bullet import Bullet

class Player:
    def __init__(self, position, canvasWidth, canvasHeight, clock):
        #initialising variables for the circle collider
        self.playerPosition = position
        self.speed = Vector()
        self.radius = 50
        self.canvasWidth = canvasWidth
        self.canvasHeight = canvasHeight

        self.IMG_POS_TEMP = [self.playerPosition.x, self.playerPosition.y]

        #initialising the variable for the sprite to jump
        self.isGrounded = True
        self.gravity = Vector(0,5)
        self.jumpHeight = Vector(0,10)
        self.isJumping = False
        self.jumpCounter = 0

        self.facingRight = True #checking if the player is facing the right

        self.points = 0

        self.delay = simplegui.create_timer(250,self.delay_handler)
        self.delay.start()
        self.delayBool = False

        self.health = [Heart((40,576)),Heart((100,576)),Heart((160,576)), Heart((220,576)), Heart((280,576))] #the player's health
        self.projWeapon = [] #an array holding all the bullet objects
        self.clock = clock

    def delay_handler(self):
        self.delayBool = False
        self.delay.start()

    #drawing the bullets and hearts onto the canvas
    def draw(self, canvas):
        if(self.isDead() == True): #chekcing if the dead window needs to be shown
            canvas.draw_text('You are Dead :(',(self.canvasWidth/2,self.canvasHeight/2), 64, 'Black')
        if(len(self.health) > 0): #drawing the hearts to the screen
            for i in range(len(self.health)): #drawing all possible hearts to the canvas
                self.health[i].draw(canvas)
        if(len(self.projWeapon) > 0):  #drawing all bullets and maintaining the list, so long as the weapon is not empty
            for x in range(len(self.projWeapon)):
                self.projWeapon[x].launch()
                self.projWeapon[x].draw(canvas) #updating the bullet's animation
```

```
    def getPlayerHitBox(self, position, radius):
        if (position.x+radius > self.playerPosition.x-self.radius*0.6) and (position.x-radius < self.playerPosition.x+self.radius*0.6) and (position.y+radius > self.playerPosition.y-self.radius*0.6) and (position.y-radius < self.playerPosition.y+self.radius*0.6):
            return True
        return False

    def takeDamage(self):
        if (self.delayBool == False):
            self.delayBool = True
            if(len(self.health) > 0): #preventing a null pointer error
                self.health.pop(len(self.health)-1) #removing the heart from the health array

    def getPlayerCurrentPos(self): #getting the current position of the player, returns a tuple
        return (self.playerPosition.x, self.playerPosition.y)

    def shootBullet(self): #creates a new bullet object and adds it to the rest of the weapon's array
        bulletPosition = Vector(self.getPlayerCurrentPos()[0], self.getPlayerCurrentPos()[1]) #starting the bullet from the player's position
        if self.facingRight == True: #sending bullets in the right direction
            bulletDirection = Vector(10,0)
        else: #sending bullets in the left direction
            bulletDirection = Vector(-10, 0)
        newBullet = Bullet(bulletPosition, bulletDirection, self.clock) #creating the new bullet object
        self.projWeapon.append(newBullet) #adding the new bullet object to the weapon's array

    def shootVerticalBullet(self): #creates a new bullet object and adds it to the rest of the weapon's array
        bulletPosition = Vector(self.getPlayerCurrentPos()[0], self.getPlayerCurrentPos()[1]) #starting the bullet from the player's position
        newBullet = Bullet(bulletPosition, Vector(0, -10), self.clock) #creating the new bullet object
        self.projWeapon.append(newBullet) #adding the new bullet object to the weapon's array

    def isDead(self): #checking if the health array ha reached zero
        self.delay.stop()
        return (len(self.health) == 0)

    def update(self, enemy):
        # changing the position vector to move in the correct position
        self.delayBool = False
        for y in range(len(self.projWeapon)-2): #going throught the weapon's array and removing any bullets that are no longer within the canvas
            if (self.projWeapon[y].launchPos.x > self.canvasWidth) or (self.projWeapon[y].launchPos.x < 0) or (self.projWeapon[y].launchPos.x < 0) or (enemy.enemyDamage(self.projWeapon[y].launchPos, self.radius)): #only checking the x coord as the bullets only move ve
                self.projWeapon.remove(self.projWeapon[y])

        self.playerPosition.add(self.speed) #moving the player arrow the canvas
        self.IMG_POS_TEMP = (self.playerPosition.x, self.playerPosition.y)

        # if the wheel y position is greater than the floor, it is no longer grounded
        if(self.playerPosition.y < (self.canvasHeight-(self.radius/2) )):
            self.isGrounded = False

        # if the wheel y position is on the floor, the wheel is grounded and the jump counter is reset
        if (self.playerPosition.y >= (self.canvasHeight- (self.radius))):
            self.isGrounded = True
            self.jumpCounter = 0

        # if the wheel is not grounded and not jumping, apply gravity
        if (self.isGrounded == False) and (self.isJumping==False):
            self.playerPosition.add(self.gravity)

    # returning function for the img_pos
    def getIMG_POS(self):
```

The EnemyVentrilo class was written by Reisha and holds the code for the enemy of fight scene 1. Some calculations are completed within the fight scene class however, this is for any interactions between other classes.
getEnemyHitBox() is similar to the getPlayerHitBox() but it has been adjusted of the enemies attributes, returning true or false whether there has been an interaction.

```
def getEnemyHitBox(self, position, radius):
    #the calculation for whether the enemy has been hit by the player's bullet. If so, return true. Otherwise, return false
    if (position.x+radius > self.enemyPosition.x-self.radius*0.5) and (position.x-radius < self.enemyPosition.x+self.radius*0.5) and (position.y+radius > self.enemyPosition.y-self.radius*0.5) and (position.y-radius < self.enemyPosition.y+self.radius*0.5):
        return True
    return False
```

IsDead() slightly differs as once the Ventriloquist enemy has been killed, it will start phase 2 and regenerate with half its health, and reset the weapon array. After being defeated a second time, the enemy is returned to be dead.

```
def isDead(self): #checking if the health array has reached zero
    if (self.health <= 0):
        self.health = 0
        self.counter += 1
        # changing of the phases, triggered by how many times enemy has died
        if (self.counter == 2): # if the second phase has been reached...
            self.projWeapon = [] #reset the enemy weapon array, so no bullets from the previous scene remain on scene
            self.background = self.background2 #changing the current background
            self.health = 150 #the enemy health is set to 150, half of its original value

            #this is phase 2, so the phase 2 variable will be set to true with all the others set to false
            self.phase1 = False
            self.phase2 = True
            self.phaseVictory = False
        elif (self.counter == 3): #if the enemy has been defeated, the counter has incremented. this is reflected in which variables are true
            self.health = 0
            self.phase1 = False
            self.phase2 = False
            self.phaseVictory = True
        return True
    return False
```

enemyDamage() uses the clock to create a delay. After the delay, the enemy will be damaged and the player will be awarded points. If the bullet hits within 50% of the enemy, the enemy will lose 10 health and the player will receive 10 points. But if the hit is within 25% of the enemy, the enemy will lose 15 health and the player will receive 30.

```
#calculating the points when the enemy has been hit
def enemyDamage(self, position, radius):
    if (self.clock.transition(25)) and (self.isDead() == False): #should only be true if there have been 25 frames gone past and the enemy is not dead
        if self.getEnemyHitBox(position, radius):
            #calculations for the type of hits which will affect how many points the user obtains
            if (position.x+radius > self.enemyPosition.x-self.radius*0.25) and (position.x-radius < self.enemyPosition.x+self.radius*0.25) and (position.y+radius > self.enemyPosition.y-self.radius*0.25) and (position.y-radius < self.enemyPosition.y+self.radius*0.25):
                self.health -= 15
                self.player.points += 30 # critical hit
            else:
                self.health -= 10
                self.player.points += 10 # damaging hit
```

Draw() is responsible for moving and drawing the enemy to the screen for both phases and any bullets. The management of the bullet array is also maintained here (removing bullets, moving them on the screen, making the player take damage).

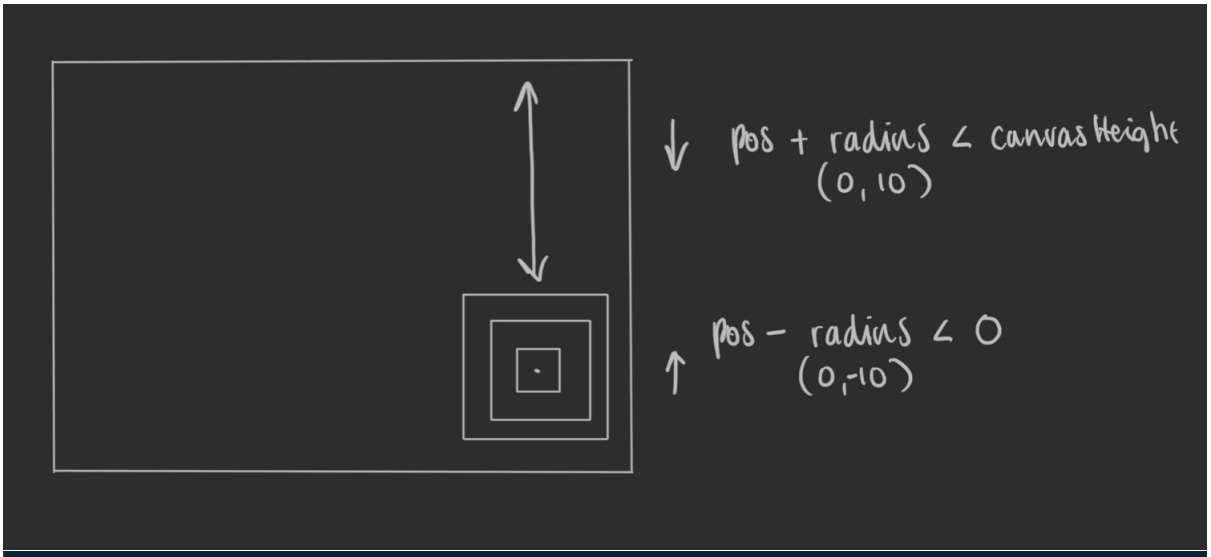
```
#drawing the bullets onto the canvas
def draw(self, canvas):
    if (self.phase1 == True): #if the first phase has been reached...
        canvas.draw_circle((self.enemyPosition.x, self.enemyPosition.y), self.radius, 1, "Red", "Red") #draw the enemy at its default position

        if(len(self.projWeapon) > 0):  #drawing all bullets and maintaining the list, so long as the weapon is not empty
            for x in range(len(self.projWeapon)-1): #for all the bullets in the list
                self.projWeapon[x].launch() #launch the bullet
                # canvas.draw_circle((self.projWeapon[x].launchPos.x, self.projWeapon[x].launchPos.y), self.projWeapon[x].radius, self.projWeapon[x].border, self.projWeapon[x].colour,self.projWeapon[x].colour)
                self.projWeapon[x].draw(canvas)#draw the bullet
                if (self.player.getPlayerHitBox(self.projWeapon[x].launchPos, self.projWeapon[x].radius) == True): #if the enemy bullet has collided with the player...
                    self.player.takeDamage() #the player will take damage
                    self.projWeapon.remove(self.projWeapon[x]) #there is no more use for the bullet. Remove it from the screen.


    elif (self.phase2 == True): #if the second phase has been reached...
        if (self.enemyPosition.x < (canvasWidth)): #Until the circle has visually moved off the screen
            self.moveCirc += 2.5 #this is the variable that increments 2.5, so the circle gradually moves off of the screen
            canvas.draw_circle(((self.enemyPosition.x + self.moveCirc), self.enemyPosition.y), self.radius, 1, "Red", "Red") #draw the circle with its everchanging position

        if(len(self.projWeapon) > 0):  #drawing all bullets and maintaining the list, so long as the weapon is not empty
            for x in range(len(self.projWeapon)-1): #for all the bullets in the list
                self.projWeapon[x].launch() #launch the bullet
                # canvas.draw_circle((self.projWeapon[x].launchPos.x, self.projWeapon[x].launchPos.y), self.projWeapon[x].radius, self.projWeapon[x].border, self.projWeapon[x].colour,self.projWeapon[x].colour) #draw the bullet
                self.projWeapon[x].draw(canvas)
                if (self.player.getPlayerHitBox(self.projWeapon[x].launchPos, self.projWeapon[x].radius) == True): #if the enemy bullet has collided with the player...
                    self.player.takeDamage() #the player will take damage
                    self.projWeapon.remove(self.projWeapon[x]) #there is no more use for the bullet. Remove it from the screen.
```

ShootBullet() changed depending on the current phase. For phase 1, a bullet is periodically shot out, created using the enemy's position and added to the weapon array. For phase 2, the bullets have a randomly generated position along the top of the screen, along with a random velocity.

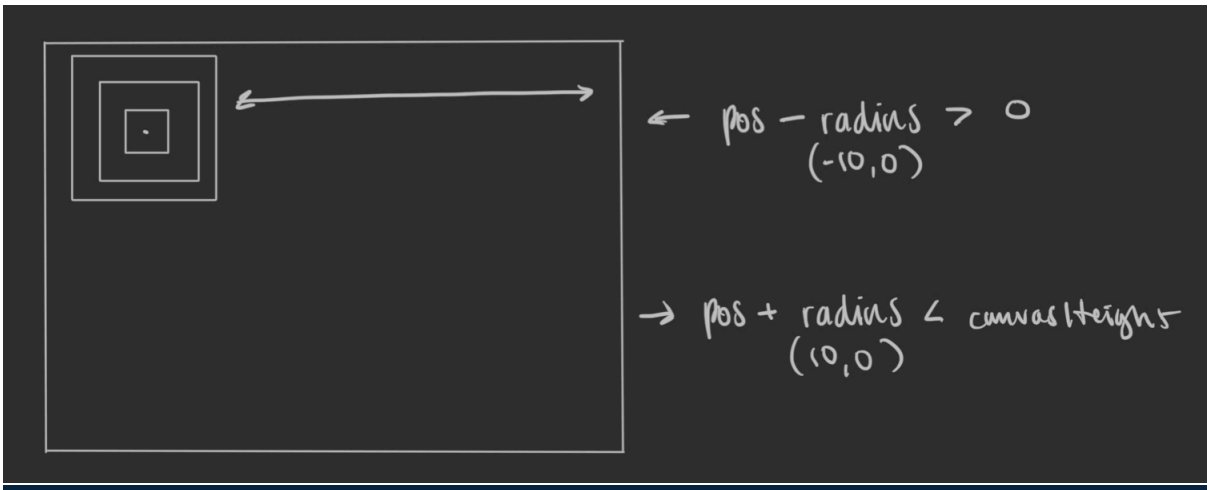
```
def shootBullet(self):
    #the method for shooting the enemy bullet

    playerLocation = self.player.getPlayerCurrentPos() #getting the users location

    if (self.phase1 == True): #if we are currently in the first phase

        if self.clock.transition(5):
            bulletPosition = Vector(self.enemyPosition.x, self.enemyPosition.y) #set the bullet position to a vector

            if (playerLocation[0] > (self.enemyPosition.x)): #if player is on the right of the enemy
                bulletDirection = Vector(5, 0) #send the bullet to the right
            else:
                bulletDirection = Vector(-5, 0) #send the bullet to the left
            newBullet = EnemyBullet(bulletPosition, bulletDirection, self.toothImage) #creating the new bullet object
            self.projWeapon.append(newBullet) #adding the new bullet object to the weapon's array

    elif (self.phase2 == True): #if we are currently in the second phase

        bulletPosition = random.randrange((playerLocation[0] - 500), (playerLocation[0] + 500)) #the position of the bullet will be random within a certain range of the player
        bulletPosition = Vector(bulletPosition) #change the bullet position into a vector
        bulletVelo = random.randrange(2, 15) #randomly changing the velocity within the range of (2, 15)
        bulletDirection = Vector(0, (bulletVelo)) #the bullet direction will be shooting down at the speed of the velocity
        if (self.clock.transition(10)):
            newBullet = EnemyBullet(bulletPosition, bulletDirection, self.toothImage) #create a new bullet by calling the enemy bullet class
            self.projWeapon.append(newBullet) #add the bullet to the weapon array
```

SetBackground() draws the background image to the canvas.
Update() is only used for phase 1 and it finds the distance of the player to the enemy and will move left or right, so a certain distance can be maintained. The bullet array will be checked for any bullet that are out of bounds will be removed.

The EnemyAerialist class work slight differently to the EnemyVentrilo class as the phases change in a different way, working on a cycle that changes with a timer.
There are four timer handlers allow for there to be delays for damage and initiating the weapons. The timer_handler() also resets certain variables which can be used for the next phase.
UpdatePhase() calls the correct update method depending on the current phase.
movingUpwards() and movingSideways() are used to move the enemy either along the right hand side of the screen or the top of the screen.




```
def movingUpwards(self):#moving the aerialist along the right side of the canvas
    if ((self.aerialPosition.y + self.radius) >= self.canvasHeight):
        self.facingUpwards = True
    if (self.aerialPosition.y - self.radius < 0):
        self.facingUpwards = False

    if (self.facingUpwards == True):
        self.speed = Vector(0, -2) #moving upwards
    else:
        self.speed= Vector(0, 2) #moving downwards

    self.aerialPosition.add(self.speed) #repositioning the enemy
```




```
def movingSideways(self):#moving the aerialist along the top of the canvas
    if ((self.aerialPosition.x + self.radius) == self.canvasWidth):
        self.facingRight = False
    if (self.aerialPosition.x - self.radius < 0):
        self.facingRight = True

    if (self.facingRight == True):
        self.speed = Vector(2, 0) #moving right
    else:
        self.speed= Vector(-2, 0) #moving left

    self.aerialPosition.add(self.speed) #repositioning the enemy
```

UpdatePhase1() moves the enemy vertically and creates a new lyra if the delay has been stopped.
UpdatePhase2() moves the enemy vertically and creates a new silk, then resets the enemy weapon.
UpdatePhase3() moves the enemy horizontally and creates a new web if the delay has been stopped.

```
def updatePhase1(self):
    self.movingUpwards()
    if self.weaponDelayBool == False: #creating a new lyra if the delay has stopped
        self.weaponDelayBool = True
        newLyra = Lyra(Vector(self.aerialPosition.x, self.aerialPosition.y), self.lyraImage) #creating the new lyra object
        self.weapon1.append(newLyra)

def updatePhase2(self):
    self.movingUpwards()
    if(self.newSilkDelayBool == True): #creating a new silk if the delay has stopped
        self.newSilkDelayBool = False
        self.newSilkDelay.stop()
        newSilk = Silk(Vector(self.aerialPosition.x, self.aerialPosition.y), self.silkImage)
        self.weapon2 = newSilk

def updatePhase3(self):
    self.movingSideways()
    if self.weaponDelayBool == False: #creating a new web if the delay has stopped
        self.weaponDelayBool = True
        newWeb = Web(Vector(self.aerialPosition.x, self.aerialPosition.y), self.webImage) #creating the new lyra object
        self.weapon3.append(newWeb)
```

IsDead() checks if the enemy has died, if so leading to the player winning.
enemyDamage() and SetBackground() works the same as in the EnemyVentrilo class.
Draw() is responsible for updating the enemy's position, then drawing, updating and maintaining the current weapon, depending of the current phase. This is where the player damage is calculated as well.

```
def draw(self, canvas):
    if(self.isDead() == False) or (self.player.isDead() == False): #checking if either the player or aerialists are dead
        self.updatePhase() #updating the aerialist's information
        canvas.draw_circle((self.aerialPosition.x,self.aerialPosition.y), self.radius, 1, "Black","Black") #drawing the enemy to the screen
        if(self.currentPhase%3 == 0) and (len(self.weapon1) > 0):  #drawing all bullets and maintaining the list, so long as the weapon is not empty
            for x in range(len(self.weapon1)-1): #drawing all the lyras to the screen and checking if they have damaged the player
                self.weapon1[x].launch()
                self.weapon1[x].draw(canvas)
                if(self.player.getPlayerHitBox(self.weapon1[x].launchPos, self.weapon1[x].radius)):
                    self.player.takeDamage()
                    self.weapon1.remove(self.weapon1[x])
                elif(self.weapon1[x].launchPos.x - self.weapon1[x].radius < 0 or self.weapon1[x].launchPos.x +self.weapon1[x].radius > self.canvasWidth) or (self.weapon1[x].launchPos.y + self.weapon1[x].radius < 0 or self.weapon1[x].launchPos.y - self.weapon1[x].radiu
                    self.weapon1.remove(self.weapon1[x]) #removing all lyras that are out of bounds or hit the player
        elif(self.currentPhase%3 == 2) and (len(self.weapon3) > 0):  #drawing all the webs and maintaining the list, so long as the weapon is not empty
            for x in range(len(self.weapon3)-1):
                self.weapon3[x].launch()
                self.weapon3[x].draw(canvas)
                if(self.player.getPlayerHitBox(self.weapon3[x].launchPos, self.weapon3[x].radius)):
                    self.player.takeDamage()
                    self.weapon3.remove(self.weapon3[x])
                elif(self.weapon3[x].launchPos.y + self.weapon3[x].radius > self.canvasHeight):
                    self.weapon3.remove(self.weapon3[x]) #removing all webs that are out of bounds or hit the player
        elif(self.currentPhase%3 == 1) and (self.weapon2 != None):
            self.weapon2.launch() #moving the silk across the canavs
            self.weapon2.draw(canvas) #drawing the silk to the canvas
            if(self.weapon2.reachedEnd()== True): #a delay before creating the next silk
                self.newSilkDelay.start()
            if (self.weapon2.getWithinSilk(self.player.playerPosition, self.player.radius*0.6) == True) and (self.delayBool == False): #cause damage to the player if they are hit with a silk
                self.delayBool = True
                self.player.takeDamage()
```

The fight scene class holds all the interactions between the enemyVentrilo, player and keyboard.
addPoints() retrieves the current points from the player object.
setPlayerSprite() returns the player animation object position depending on certain conditions with the hierarchy being attack – jumping -running- idle. This was originally within the update method, however the elif statements did not allow for compound movement, reducing the player's mobility.

```
def setPlayersprite(self):
    if(self.keyboard.shoot):
        if(self.player.facingRight):
            return 6
        return 5
    elif (self.keyboard.jump):
        if(self.player.facingRight):
            return 4
        return 3
    if ((self.keyboard.right) or (self.player.getPlayerCurrentPos()[0] <= (70))):
        return 2
    elif ((self.keyboard.left) or (self.player.getPlayerCurrentPos()[0] >= (self.canvasWidth - (self.player.radius/2)))):
        return 1
    return 0
```

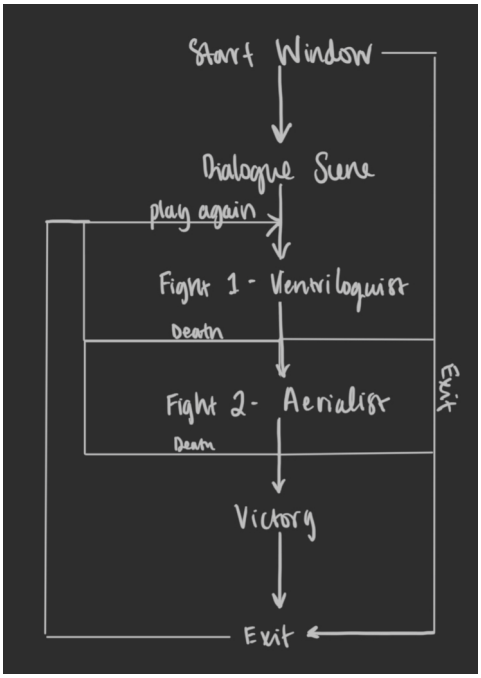
Draw() draws the current points to the screen.
Update() initially sets up the weapon for th enemy. For phase 1, a random timer is set for the clock, which will then shoot a bullet. Phase 2 will shoot a bullet every cycle, regardless of any timer. The rest of the code is responsible for checking the keyboard for any input then responding correspondingly. Due to using if statements, compound movements are allowed. If either the player or enemy dies, a flag will be set which will be used by the game loop to move on.

```
def update(self, canvas):
    self.enemyVentrilo.setBackground(canvas)
    self.clock.tick() #incrementing the clock each cycle
    self.addPoints()
    self.draw(canvas)

    if (self.player.isDead() == False) and (self.enemyVentrilo.isDead() == False):
        if (self.enemyVentrilo.phase1 == True):
            bulletTimer = random.randrange(20, 50)
            if self.clock.transition(bulletTimer): #adding a delay to shooting the bullets
                self.enemyVentrilo.shootBullet()
        elif(self.enemyVentrilo.phase2 == True):
            self.enemyVentrilo.shootBullet()
        if ((self.keyboard.right) or (self.player.getPlayerCurrentPos()[0] <= (70))): # makes the player move right so long it is within boundary
            self.player.facingRight = True #marking to check if the player is facing right
            self.player.playerPosition.add(Vector(2, 0)) #moving the character to the right
            if (self.player.getPlayerCurrentPos()[0] <= (75)):
                self.player.playerPosition.add(Vector(2,0))
        if ((self.keyboard.left) or (self.player.getPlayerCurrentPos()[0] >= (self.canvasWidth - (self.player.radius/2)))): # makes the player move left so long it is within boundary
            self.player.facingRight = False #marking to check if the player is facing left
            self.player.playerPosition.add(Vector(-2, 0))
            if (self.player.getPlayerCurrentPos()[0] >= (530)):
                self.player.playerPosition.add(Vector(-2,0))
        if (self.keyboard.jump):
                if (self.player.jumpCounter <= 30): # if the jump counter is not exceeded, add the jump height vector to the player's position
                    self.player.isJumping = True
                    self.player.playerPosition.add(-(self.player.jumpHeight))
                    self.player.jumpCounter += 1
                    self.player.on_ground = False
            # used to prevent the response from the w button from being too sensitive
        if(self.keyboard.shoot):
                if self.clock.transition(5): #adding a delay to shooting the bullets
                    self.player.shootBullet() #creating a new bullet
        self.playerSprites.currentSprite = self.setPlayersprite()
        self.player.isJumping = False
    else:
        if (self.player.isDead()):
            self.playerDead = True
        if (self.enemyVentrilo.phaseVictory == True):
            canvas.draw_text("Victory!!!!",(canvasWidth/2,canvasHeight/2), 36, "Black")
        self.finished = True
```

The fightscene2 class holds all the interactions between the enemyAreialists, player and keyboard.
addPoints(), draw() and setPlayerSprite() all work the same as in the previous fight scene.
Update() is only responsible for checking the keyboard for any input and having the player react. All the movement and damage cause to and by the enemy is calculated within the enemy class.

```
def update(self, canvas):
    self.clock.tick() #incrementing the clock each cycle
    self.addPoints() #getting the players points
    self.draw(canvas)

    if(self.player.isDead() == False) and (self.aerialists.isDead() == False): #checking is the player is dead
        self.aerialists.draw(canvas) #updating the aerialists
        if ((self.keyboard.right) or (self.player.getPlayerCurrentPos()[0] <= (70))):
            self.player.facingRight = True #marking to check if the player is facing right
            self.player.playerPosition.add(Vector(5, 0)) #moving the character to the right
        # makes the player move left so long it is within boundary
        if ((self.keyboard.left) or (self.player.getPlayerCurrentPos()[0] >= (self.canvasWidth - (self.player.radius/2)))):
            self.player.facingRight = False #marking to check if the player is facing left
            self.player.playerPosition.add(Vector(-5, 0))

        # if the jump counter is not exceeded, add the jump height vector to the player's position
        if (self.keyboard.jump):
            if (self.player.jumpCounter <= 30): #setting the jumping sprite object
                self.player.isJumping = True
                self.player.playerPosition.add(-(self.player.jumpHeight))
                self.player.jumpCounter += 1
                self.player.on_ground = False #allowing for gravity to be added the player
        if(self.keyboard.damage):
            if self.delayBool == False: #adding a delay to the damage
                self.player.takeDamage() #removing one of the hearts
        if(self.keyboard.shoot):
            if self.delayBool == False:  #adding a delay to shooting the bullets
                self.delayBool = True
                self.player.shootBullet() #creating a new bullet
        if(self.keyboard.shootUp):
            if self.delayBool == False:  #adding a delay to shooting the bullets
                self.delayBool = True
                self.player.shootVerticalBullet() #creating a new bullet
        self.playerSprites.currentSprite = self.setPlayersprite() #setting the player's current sprite
        self.player.isJumping = False #allowing gravity to be applied
    else:
        if (self.player.isDead()== True): #setting the death screen
            self.delay.stop()
            self.aerialists.stopAreialists()
            self.playerDead = True
            self.finished = True
        else:
            self.delay.stop()
            self.aerialists.stopAreialists()
            self.finished = True #setting the victory screen
```

The Game class is responsible for running the main game loop of the game. All needed class are instantiated within the constructor, and set where appropriate. Some variables are initially set to none as the variable needs to be referenced throughout the program, however is waiting for some more information.
Initially, all the key components were separate and had their own windows, but everything was reworked so there is only one draw handler.



Using the diagram shown, I attempted to create a cycle so the correct objects are created and updated.
Update() uses an int variable, currentScene, to create a series of if statements allowing for the correct objects to be updated. Here are the corresponding numbers to objects; 0 – start window, 1 – Dialogue scene, 2 – fight scene 1, 3 – fight scene , 4 – death window, 5 – victory window.
If the specific condition has been met, such as the player dying or finishing a scene, then the next possible scene will be prepared. Otherwise, the scene will be updated and continued. This way, players are able to replay sections or repeat the game loop, as the objects will just be reset.

```
def update(self, canvas):
    if self.currentScene == 0: #running the start window and waiting to move on
        if self.startDialogue == True:
            if (self.name  != None): #preparing the dialogue scene with a name from the user
                self.dialogueScene = DialogueScene(self.keyboard, self.canvasWidth, self.canvasHeight, self.frame, self.name) #initialising the dialogue scene
                self.currentGameObject = self.dialogueScene #setting the new current game object
                self.currentScene += 1
            else: #preparing the dialogue scene without a name from the user
                self.dialogueScene = DialogueScene(self.keyboard, self.canvasWidth, self.canvasHeight, self.frame)
                self.currentGameObject = self.dialogueScene
                self.currentScene += 1
        else:
            gameValues = self.currentGameObject.startGame()
            self.startDialogue = gameValues[0] #checking if the game should be started
            self.name = gameValues[1] #checking if a name has been entered by the user
    elif self.currentScene == 1: #running the dialogue scene and waiting to move on
        if self.dialogueScene.finished == True: #preparing the first fight scene
            self.player = Player((Vector(self.canvasWidth/2,self.canvasHeight/2)), self.canvasWidth, self.canvasHeight, self.clock)
            self.playerSprites = PlayerSprites(self.player, self.clock)
            self.enemyVentrilo = EnemyVentrilo(self.canvasWidth, self.canvasHeight, self.player, self.clock)
            self.fightScene1 = FightScene(self.player, self.clock, self.playerSprites, self.keyboard, self.canvasWidth, self.enemyVentrilo, self.frame)
            self.currentGameObject = self.fightScene1 #setting the new current game object
            self.currentScene +=1
    elif(self.currentScene == 2): #running fight scene 1 and waiting to move on
        if(self.fightScene1.finished == True):
            if(self.fightScene1.playerDead == True): #preparing the death window
                self.deathWindow = DeathScreen(self.player.points, self.canvasWidth, self.canvasHeight, self.frame)
                self.currentGameObject = self.deathWindow #setting the new current game object
                self.currentScene =  4
            else: #preparing the second fight scene
                self.enemyAerialists = EnemyAerialists(self.canvasWidth, self.canvasHeight, self.player, self.clock)
                self.fightScene2 = FightScene2(self.player, self.enemyAerialists, self.clock, self.playerSprites, self.keyboard, self.canvasWidth, self.player.points, self.frame)
                self.currentGameObject = self.fightScene2 #setting the new current game object
                self.currentScene += 1
        else: #updating the first fight scene
            self.fightScene1.update(canvas)
            self.player.update(self.enemyVentrilo)
            self.player.draw(canvas)
            self.playerSprites.draw(canvas)
            self.enemyVentrilo.update()
            self.enemyVentrilo.draw(canvas)
    elif(self.currentScene == 3): #running fight scene 2 and waiting to move on
        if(self.fightScene2.finished == True):
            if(self.fightScene2.playerDead == True): #preparing the death window
                self.deathWindow = DeathScreen(self.player.points, self.canvasWidth, self.canvasHeight, self.frame)
                self.currentGameObject = self.deathWindow #setting the new current game object
                self.currentScene +=  1
            else: #preparing the victory window
                self.victoryWindow = EndingScreen(self.player.points, self.canvasWidth, self.canvasHeight, self.frame)
                self.currentGameObject = self.victoryWindow #setting the new current game object
                self.currentScene = 5
        else: #updating the second fighting scene
            self.enemyAerialists.setBackground(canvas)
            self.fightScene2.update(canvas)
            self.player.update(self.enemyAerialists)
            self.player.draw(canvas)
            self.playerSprites.draw(canvas)
            if(self.player.isDead() or self.enemyAerialists.isDead()):
```

Reflection

Looking back on the project, I believe that the initial plan was quite overly ambitious and failed to take into account the amount of time we really had. Despite this, I pretty happy with the end product, although certain aspects had to be removed due to timing.

In general, the group worked pretty poorly and the amount of work completed was extremely unbalanced. Two group members communicated very little, leading to uncertainty and problems with dividing the work, as it was very difficult to generally contact these team members.

Ali continued to not communicate nor contribute to the project despite our attempts, until two weeks before the project was due. Although, I could have extended more communication to Ali, this became extremely difficult at such a late stage as I was coding/building a large part of the game myself and had no time outside of that, and I was beginning to struggle with my own work.

Luckily, Reisha and I were able to complete the game but in future, a more rigid plan would made with alternatives routes in case of issues/bugs arising.

Generally, I did not have a largely hard time coding as I attempted to plan out any code I would write beforehand. One aspect that was particularly difficult to conceptualise was trying to get the sprites to change depending on the situation. Although this took some time to figure out, I used the same concept to create the game's control flow.

Conclusion

In conclusion, although the project was completed, better communication and more fair distribution of work would provide a better experience for all involved. While Reisha and I are happy with the outcome, this group work experience has been generally negative and stressful. In future, I will attempt to create a stronger line of communication and see if this can help bring the group together.