Final Project
Algorithm and Programming

Name: Tiffany Widjaja

Student ID: 2502059360

Class: L1CC

Project Name: FlappaMEW

Table of Contents

Chp. 1 – Project Specification:

A. Introduction
B. Game Overview

Chp. 2 – Solution Design:

A. Game Graphics and Sound
B. Game Mechanism and Controls

Chp. 3 – Implementation and Explanation of Code:

A. Creating Sprites
B. Creating Retry Button
C. Creating Game Over Situations
D. Creating Controls
E. Pygame Functions Explained
F. Sprites and Background implemented

Chp. 4 – Evidence of Working Program:

A. Screenshots

Chp. 5 – Reflection:

A. Program Evaluation
B. Self-reflection & Experience

Chp. 1 – Project Specification

A. Introduction

The first stage was to come up with ideas. At the time I had two in mind, one was an original card game using the Tkinter library and the other was a discord bot using the Discord library. The rules I implemented for the card game however was too difficult to code, so I gave up on the idea. As for the discord bot, I couldn't think of what features to add and so I discard it as well. At some point, I came across videos of people creating their own games on Youtube with the Pygame library. I became interested to learn more about it. Since this is my first time doing a Python project, creating a game would definitely help expand my knowledge of the Pygame library and my skills in coding. Thus I decided to create a pokemon fan game that is heavily inspired by watching people recreating Flappy Bird and Space Invaders. This project was initiated on the 10th of January 2022 with Visual Studio Code as my IDE.

B. Game Overview

FlappaMEW is a fan-made Pokemon game that combines the playstyle elements of Flappy Bird and Space Invaders. Players play as MEW and their goal is to survive and win the battle against shiny pokemon in the air. Players can shoot bullets at the enemies while attempting to avoid their attacks in the air. The purpose of creating this game is to appreciate the gameplay and design of retro arcade games that may have been forgotten or not known by newer generations. Such games, although used to be popular back in the day, are now being replaced by 3D high graphics games with popular genres like RPG, FPS, and MOBA. Raising the popularity of arcade games isn't the intention, but it is to bring the nostalgic feel of retro to the current generation and raise their familiarity with such games.

Chp. 2 – Solution Design

A. Game Graphics and Sound

As said previously, the game graphics are intended to be similar to 2D retro arcade games. However, to get people interested in playing in the first place, there needs to be something from first glance that convinces them to try. This gave me the idea of adding Pokemon Sprites into the game as one of the ways to appeal. Not to mention, Pokemon is a popular video game series loved by many and still continuously developing that even newer generations nowadays know. Moreover, the sprites suit the retro theme I'm aiming for. Thus I've decided to use a Mew sprite as the player and four shiny pokemon sprites for the enemies. These sprites are images taken from the pokemon database sprites gallery: https://pokemondb.net/sprites/

As for the background, I was looking for a soft pixelated sky image to represent that MEW is indeed flying above the clouds. I ended up using an image from itch.io created by Cania which can be accessed from here:
https://caniaeast.itch.io/simple-sky-pixel-backgrounds
To complement the soft pastel background, I added a music box version of "Never gonna give you up" by Rick Astley as the background music. This is intended to make the gameplay more chill and enjoyable while the song choice is for the memes. The song can be found on this youtube channel: https://www.youtube.com/c/R3MusicBoxEnglish/aboutn while the original can be found here: https://www.youtube.com/watch?v=dQw4w9WgXcQ&ab_channel=RickAstley

For MEW's attacks or projectiles, I decided to use a pixel heart image to add a cuteness factor into the game. The image used can be seen here:
https://www.pngfind.com/mpng/TiiJiho_minecraft-heart-png-black-pixel-heart-png-transparent/
Each time the heart projectile hits an enemy, two sound effects can be heard, one right in collision with the enemy and the second one after the explosion occurs. Those two sound effects are the bonk sound effect (an internet meme) and the question mark ping sound effect taken from League of Legends.

For the enemies' attacks, I thought adding pixel shurikens as projectiles would look cool instead of just normal lasers. The image can be found here:
https://www.pngfind.com/mpng/wwmoiT_water-shuriken-water-shuriken-png-transparent-png/
Each time MEW gets hit by the shuriken projectile, an "oof" sound effect can be heard which is taken from the game Roblox. All the sound effects I've added are meme-related and it was added just for fun purposes.

Lastly, for the pixel retry button that appears whenever MEW dies, I used an image from the link below as it is simple and suits the cute retro theme:
http://pixelartmaker.com/art/ca2b008fd7a6ea9

B. Game Mechanism and Controls

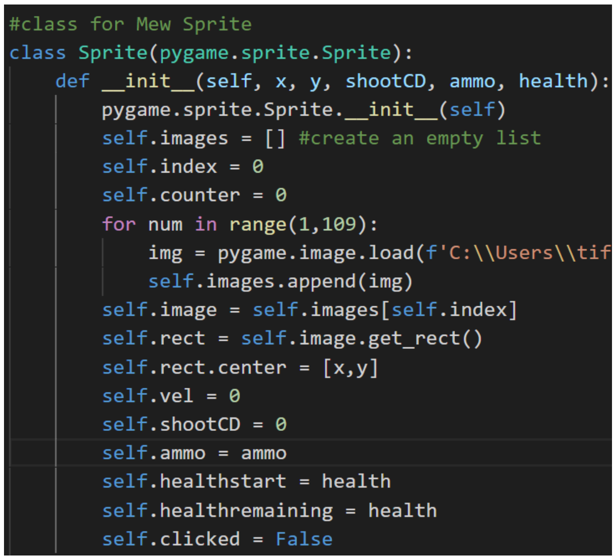
How the game works is intentionally designed to be a mix of Flappy Bird and Space Invader's gameplay. As said before, to survive MEW needs to stay alive dodging all the shuriken projectiles while killing all the Shinies off with its heart projectiles. The controls and game mechanism are designed to be simple and straightforward so that anyone at any age can play.

In order to shoot out heart projectiles, players need to press the space bar button repetitively. If the heart projectile hits an enemy, the enemy will be eliminated. The goal is to eliminate all enemies successfully and if the player succeeds, the game is over and it will get closed automatically as there are no more levels provided.

In an attempt to dodge the shuriken projectiles, players need to left-click their mouse button to get a slight boost of flight. Players are unable to constantly hold left-click their mouse and should repetitively left-click to keep on getting the boost. Otherwise, MEW will keep on falling and eventually hit the bottom of the screen. When MEW falls and hits the edge of the bottom screen, MEW will end up dying and a retry button will appear for players to retry. If players attempt to fly too high, MEW will automatically die and a retry button will appear as well. Lastly, if MEW's health bar runs out and has turned fully red, MEW will explode which will cause a retry button to appear afterward. Upon clicking the retry button, players get to retry the game will a full green health bar and all the enemies back to where they are.

Chp. 3 – Implementation and Explanation of Code

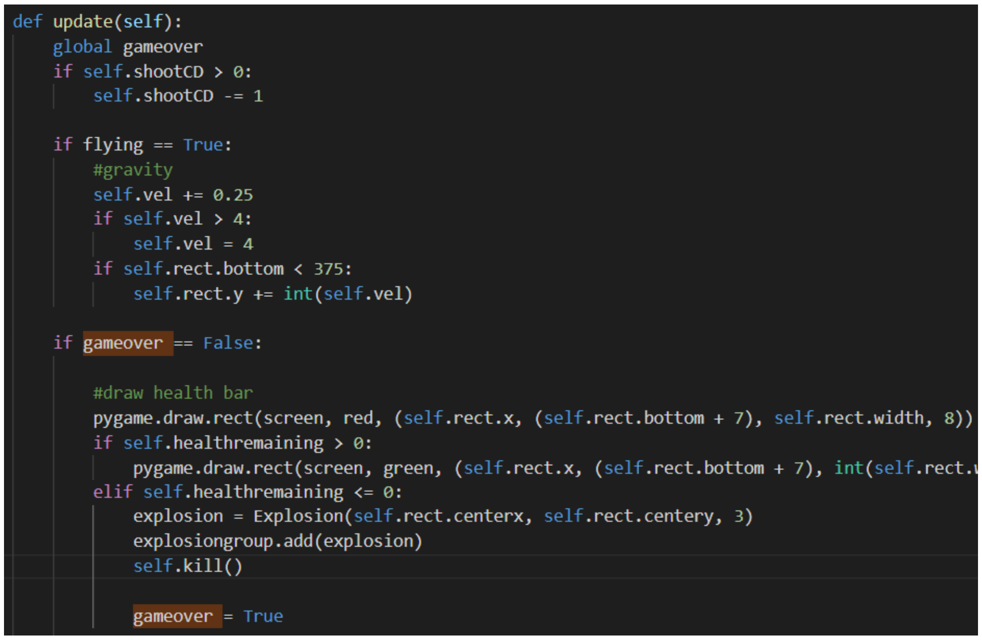
A. Creating Sprites:



```
#class for Mew Sprite
class Sprite(pygame.sprite.Sprite):
    def __init__(self, x, y, shootCD, ammo, health):
        pygame.sprite.Sprite.__init__(self)
        self.images = [] #create an empty list
        self.index = 0
        self.counter = 0
        for num in range(1,109):
            img = pygame.image.load(f'C:\\Users\\tif
            self.images.append(img)
        self.image = self.images[self.index]
        self.rect = self.image.get_rect()
        self.rect.center = [x,y]
        self.vel = 0
        self.shootCD = 0
        self.ammo = ammo
        self.healthstart = health
        self.healthremaining = health
        self.clicked = False
```

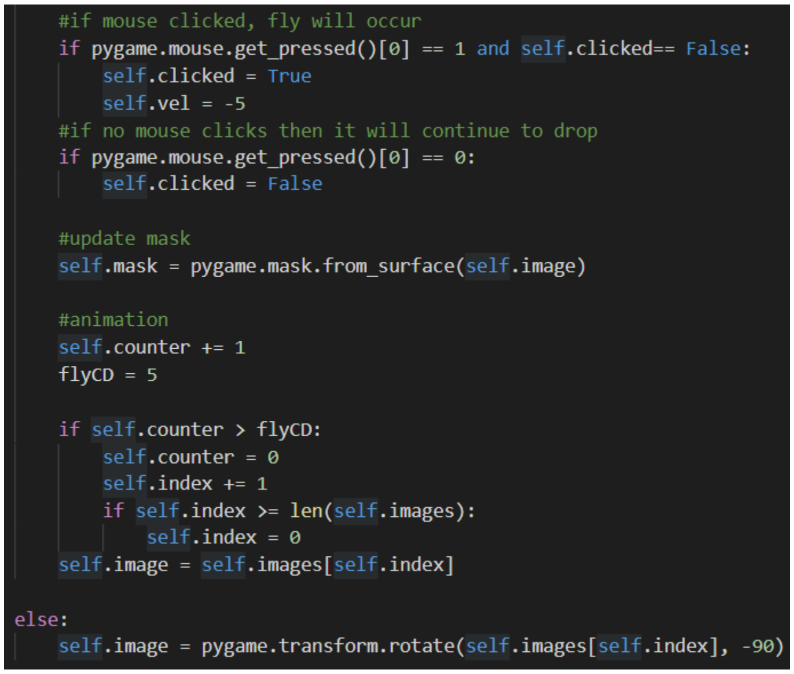
I decided to use classes to store and implement various sprites in the game. It is important to use classes as it keeps the data members and methods together in one place, keeping the program more organized. It also provides a clean structure to the code which increases the readability of the program. As can be seen from the code, to create MEW as a sprite, I had to create a Pygame's sprite class with **pygame.sprite.Sprite** to store visible game objects. Next, I needed to use the **def __init__** method to initialize the object's attributes. **pygame.sprite.Sprite.__init__(self)** allows me to inherit some of the functionality of the sprite class such as update and draw functions that are already built into them. I wanted my MEW to be an animated sprite. To do that, I took a GIF image of MEW and converted it into frames in PNG format which is then stored in a single folder. Using a for loop and in range function allows me to iterate through the MEW images from 1 - 109. **pygame.image.load** loads the image from the specified file path which is then stored into a variable called **img**. That variable is then appended into the empty list created called **images** which can be accessed later on in the update method. **self.image.get_rect()** is used to create a rectangle from the boundaries of that image. Since I want this rectangle to be positioned in the center of my Mew sprite, I wrote down **self.rect.center = [x.y]**.

def update(self) is a method that allows us to control the sprite behavior. In this part of the code, I used an **if statement** and a **boolean value** to declare the velocity of MEW. As long as MEW flies, then the gravity will be applied. If it reaches the bottom of the rectangle which I've specified, MEW will keep on dropping down with a limit. I then used another if statement to declare what happens when the gameover is False. A health bar underneath MEW is created using **pygame.draw.rect()** which draws out a rectangle. With **if** and **elif statements**, it determines what happens to MEW if its health remaining is more than 0 and when it's less than or equals to 0. It's self explanatory that when MEW's health is 0, MEW will die and that is what the **self.kill()** is for.

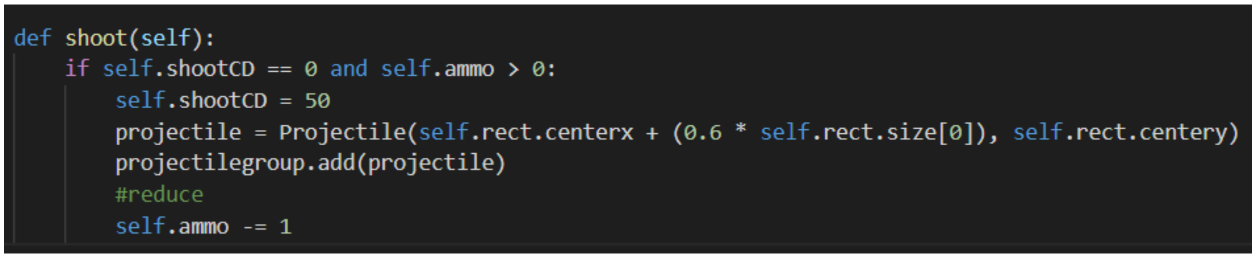


```
def update(self):
    global gameover
    if self.shootCD > 0:
        self.shootCD -= 1

    if flying == True:
        #gravity
        self.vel += 0.25
        if self.vel > 4:
            self.vel = 4
        if self.rect.bottom < 375:
            self.rect.y += int(self.vel)

    if gameover == False:

        #draw health bar
        pygame.draw.rect(screen, red, (self.rect.x, (self.rect.bottom + 7), self.rect.width, 8))
        if self.healthremaining > 0:
            pygame.draw.rect(screen, green, (self.rect.x, (self.rect.bottom + 7), int(self.rect.
        elif self.healthremaining <= 0:
            explosion = Explosion(self.rect.centerx, self.rect.centery, 3)
            explosiongroup.add(explosion)
            self.kill()

            gameover = True
```

To apply the Flappy bird jump concept, I used the **pygame.mouse.get_pressed()** function which detects whether any mouse clicks have happened or not. It returns a list that corresponds to the left, middle, and right mouse buttons. I want my jump/fly to be controlled by a left click, which means I need to access the 0 index, which is always the first one. If it equates to 1 while **self.clicked** is still false, a jump will occur and **self.clicked** will be True. If it detects no left mouse clicks, then **self.clicked** will be False and so MEW will continue to drop. Since I've appended the images into the empty list earlier, it can now be accessed to create the animation. How counter works is it limits or controls the rate at which the animation cycles through. The counter is increased by one for every iteration. The index will be increased by one each time which will go through the images from index 0 until the end it. If the index is more or equal to the length of the list of images, the index will start all the way to 0. This will allow the images to be continuously looped making MEW look animated. Next, I added in an else statement where MEW gets rotated 90 degrees using the **pygame.transform.rotate function** which only happens when it's neither flying == True and gameover == False.

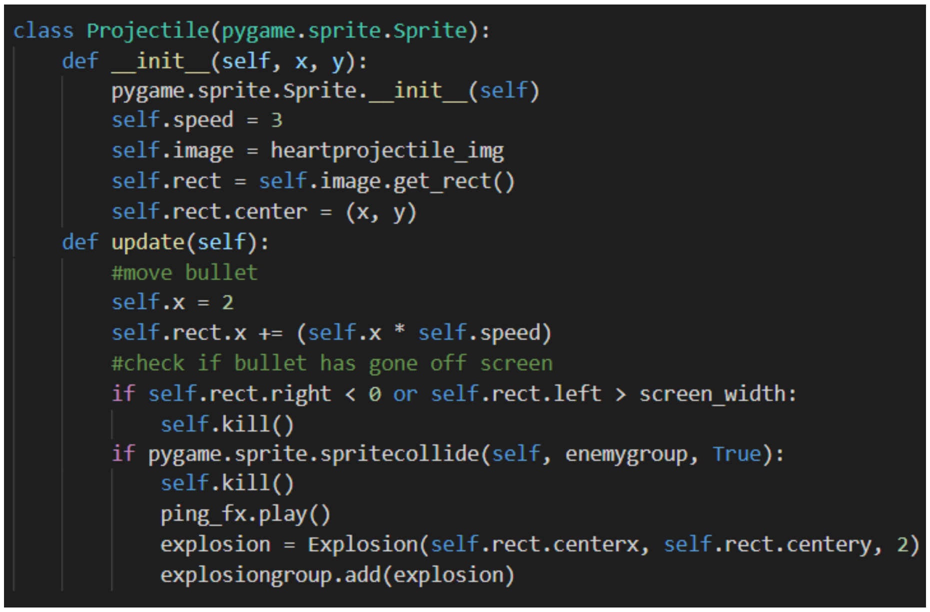


```
    #if mouse clicked, fly will occur
    if pygame.mouse.get_pressed()[0] == 1 and self.clicked== False:
        self.clicked = True
        self.vel = -5
    #if no mouse clicks then it will continue to drop
    if pygame.mouse.get_pressed()[0] == 0:
        self.clicked = False

    #update mask
    self.mask = pygame.mask.from_surface(self.image)

    #animation
    self.counter += 1
    flyCD = 5

    if self.counter > flyCD:
        self.counter = 0
        self.index += 1
        if self.index >= len(self.images):
            self.index = 0
    self.image = self.images[self.index]

else:
    self.image = pygame.transform.rotate(self.images[self.index], -90)
```



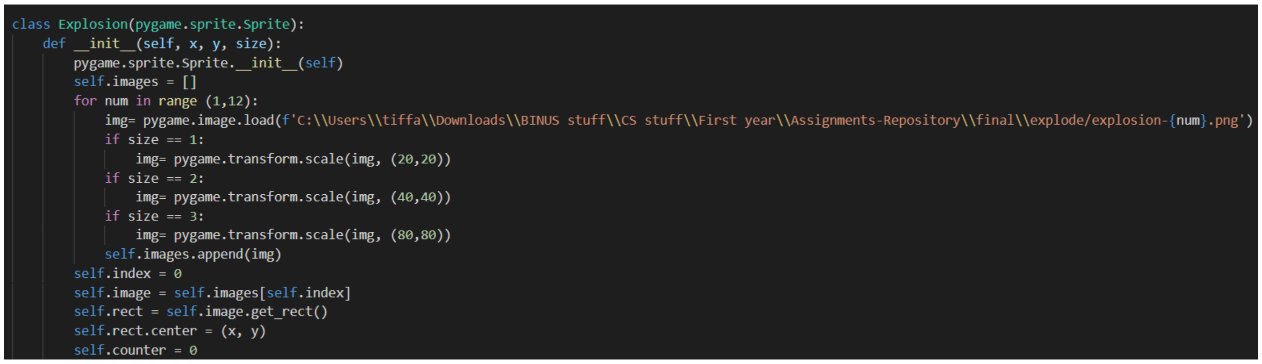
```
def shoot(self):
    if self.shootCD == 0 and self.ammo > 0:
        self.shootCD = 50
        projectile = Projectile(self.rect.centerx + (0.6 * self.rect.size[0]), self.rect.centery)
        projectilegroup.add(projectile)
        #reduce
        self.ammo -= 1
```

Lastly, for the MEW sprite, I created a separate method for shooting projectiles where the cooldown of how many you can shoot at a time and the speed is limited specifically. The Projectile class which contains the heart bullets for MEW is being assigned to the variable called projectile. This is then added into the projectile group which is a pygame.sprite.Group(), a container class that holds and manages multiple Sprite objects.



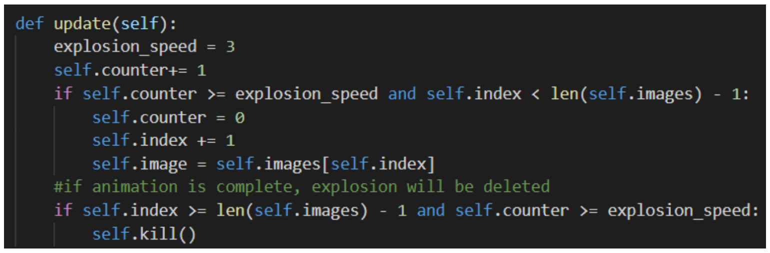
```
class Projectile(pygame.sprite.Sprite):
    def __init__(self, x, y):
        pygame.sprite.Sprite.__init__(self)
        self.speed = 3
        self.image = heartprojectile_img
        self.rect = self.image.get_rect()
        self.rect.center = (x, y)
    def update(self):
        #move bullet
        self.x = 2
        self.rect.x += (self.x * self.speed)
        #check if bullet has gone off screen
        if self.rect.right < 0 or self.rect.left > screen_width:
            self.kill()
        if pygame.sprite.spritecollide(self, enemygroup, True):
            self.kill()
            ping_fx.play()
            explosion = Explosion(self.rect.centerx, self.rect.centery, 2)
            explosiongroup.add(explosion)
```

The next class I created was for MEW's projectile or heart bullets. What's important to explain here is how I specified which direction the bullet moves. Since MEW's enemies are going to be on the right side of the screen, MEW needs to shoot the bullets towards the positive x-direction and how fast it's going is determined by the speed specified in the code. When the bullets shoot off, it's important to remove the bullets once it goes off the screen otherwise it's just going to keep collecting the bullets shot. To do that, after using an if condition, **self.kill()** is implemented to remove the projectile sprite off. Another if statement here is specified with **pygame.sprite.spritecollide** which when the bullet hits the enemy sprites upon collision, the enemies will be removed off the screen with the **self.kill()**. I also added a sound effect to play whenever the collision occurs along with the explosion image of the heart projectile by adding the Explosion class into the sprites group.



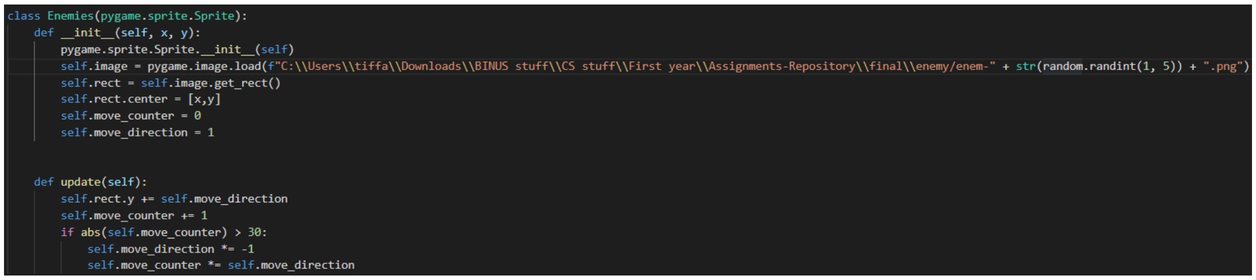
```
class Explosion(pygame.sprite.Sprite):
    def __init__(self, x, y, size):
        pygame.sprite.Sprite.__init__(self)
        self.images = []
        for num in range (1,12):
            img= pygame.image.load(f'C:\\Users\\tiffa\\Downloads\\BINUS stuff\\CS stuff\\First year\\Assignments-Repository\\final\\explode/explosion-{num}.png')
            if size == 1:
                img= pygame.transform.scale(img, (20,20))
            if size == 2:
                img= pygame.transform.scale(img, (40,40))
            if size == 3:
                img= pygame.transform.scale(img, (80,80))
            self.images.append(img)
        self.index = 0
        self.image = self.images[self.index]
        self.rect = self.image.get_rect()
        self.rect.center = (x, y)
        self.counter = 0
```

Speaking of the Explosion class, as can be seen, I used the same way to make the explosion animated by using the for in range function and appending it into an empty list. Before that, I've scaled the images into three different sizes using the pygame.transform.scale(). The first size is for the explosion when the enemy's projectile hits MEW. The second size is for the explosion when MEW's projectile hits the enemy. The last size is for the explosion that occurs when MEW runs out of health.



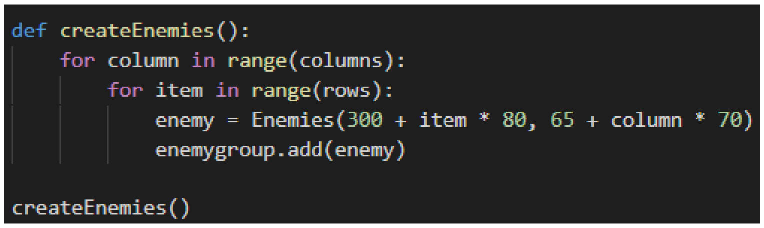
```
def update(self):
    explosion_speed = 3
    self.counter+= 1
    if self.counter >= explosion_speed and self.index < len(self.images) - 1:
        self.counter = 0
        self.index += 1
        self.image = self.images[self.index]
    #if animation is complete, explosion will be deleted
    if self.index >= len(self.images) - 1 and self.counter >= explosion_speed:
        self.kill()
```

In the Explosion class update method, I added the speed on how fast I want the animation to go through. If the index number is equals to or bigger than the length of images and when the counter is equals or larger than the explosion speed, the explosion sprite will be removed since the animation has been completed once.



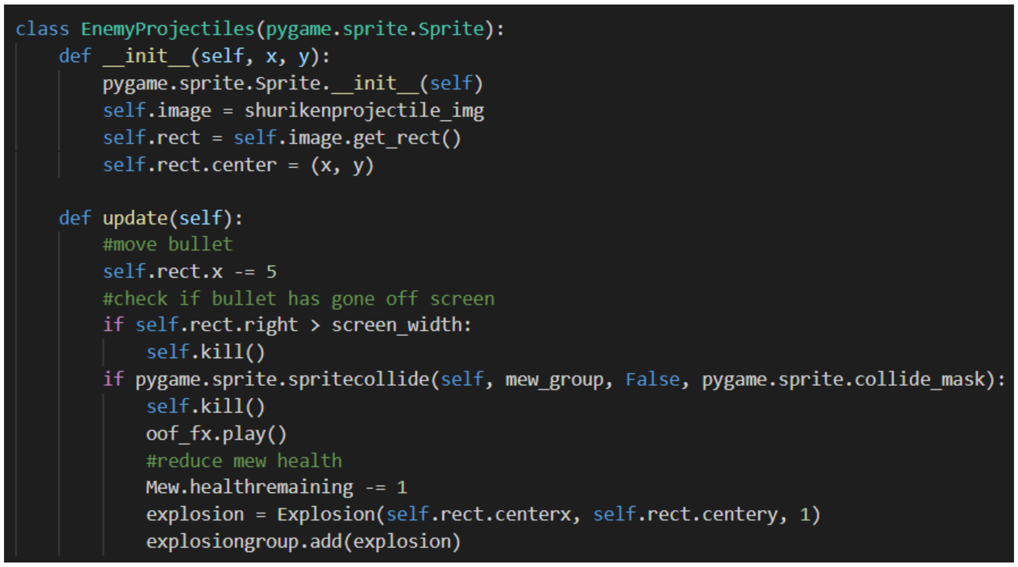
```
class Enemies(pygame.sprite.Sprite):
    def __init__(self, x, y):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.image.load(f"C:\\Users\\tiffa\\Downloads\\BINUS stuff\\CS stuff\\First year\\Assignments-Repository\\final\\enemy/enem-" + str(random.randint(1, 5)) + ".png")
        self.rect = self.image.get_rect()
        self.rect.center = [x,y]
        self.move_counter = 0
        self.move_direction = 1


    def update(self):
        self.rect.y += self.move_direction
        self.move_counter += 1
        if abs(self.move_counter) > 30:
            self.move_direction *= -1
            self.move_counter *= self.move_direction
```

For the enemies, I also created a class to store their sprites. I added in 4 shiny pokemon images which I named enem-1, enem-2, and so on. From importing the built-in random module, I used random.randint which returns a random integer number selected element from the specified range. I also added in a move counter and a direction for the enemies. The direction of which the enemies can move is determines by its y axis. Here I used the abs() function which is used to return the absolute value of a number. If it's gone past 30, it will move towards the opposite direction and this movement will keep on repeating back and forth. Outside of the Enemies class, I created a function to position the sprites of the enemies into rows and columns which is then added into the sprite group. The width and height spacing has also been determined in the code.



```
def createEnemies():
    for column in range(columns):
        for item in range(rows):
            enemy = Enemies(300 + item * 80, 65 + column * 70)
            enemygroup.add(enemy)

createEnemies()
```

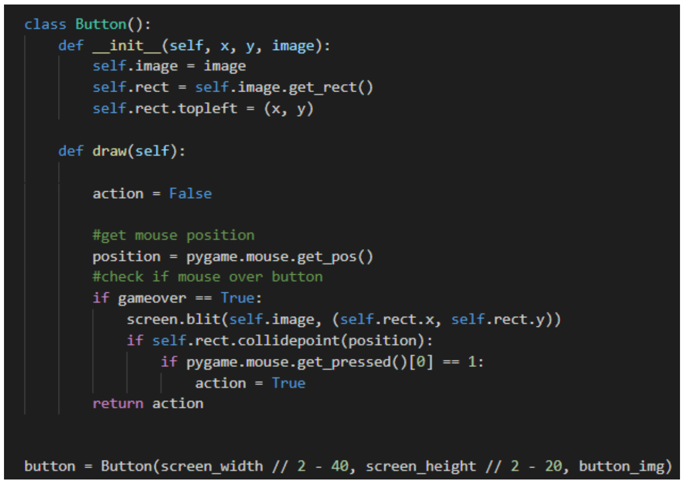
Lastly to create the enemy projectiles, I had to make another class which consisted of the shuriken sprite. Since we want the enemy projectiles to shoot towards the left side of the screen, we can determine the rect.x -= 5. Each time the enemy projectiles hit's MEW, the projectile will be removed instantly. pygame.sprite.spritecollide finds sprites in a group that has intersected with another sprite. It is a way to show whether the projectile has hit MEW or not. I've added in a sound effect to play whenever the enemy projectile hits MEW. Each time it does, MEW's health will be reduced by 1 until it reaches 0. I've also added the explosion sprite into the group so that each time MEW gets hit by the projectile, an animated explosion will occur. While the game runs, I've created an if statement which consist of an attackingenemy variable. This variable consist of the enemygroup sprites that contains the Enemy class objects. random.choice() selects a random sprite to attack a shuriken projectile.



```python
class EnemyProjectiles(pygame.sprite.Sprite):
    def __init__(self, x, y):
        pygame.sprite.Sprite.__init__(self)
        self.image = shurikenprojectile_img
        self.rect = self.image.get_rect()
        self.rect.center = (x, y)

    def update(self):
        #move bullet
        self.rect.x -= 5
        #check if bullet has gone off screen
        if self.rect.right > screen_width:
            self.kill()
        if pygame.sprite.spritecollide(self, mew_group, False, pygame.sprite.collide_mask):
            self.kill()
            oof_fx.play()
            #reduce mew health
            Mew.healthremaining -= 1
            explosion = Explosion(self.rect.centerx, self.rect.centery, 1)
            explosiongroup.add(explosion)
```

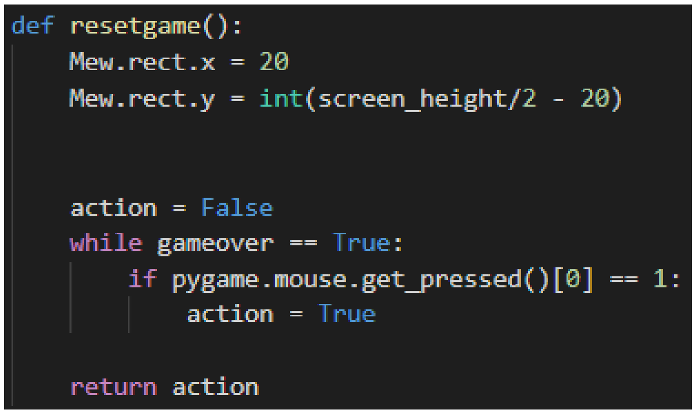
B. Creating Retry Button

For the retry button, I also created a class for it. Since the button is actually an image that I've loaded outside of the class, first I need to get the image's rectangle. Then to draw the button and display it on the screen, we can use screen.blit(). To make the button work, first we have to make sure that the mouse is hovering over the button. To check for the mouse position, we can use



```python
class Button():
    def __init__(self, x, y, image):
        self.image = image
        self.rect = self.image.get_rect()
        self.rect.topleft = (x, y)

    def draw(self):

        action = False

        #get mouse position
        position = pygame.mouse.get_pos()
        #check if mouse over button
        if gameover == True:
            screen.blit(self.image, (self.rect.x, self.rect.y))
            if self.rect.collidepoint(position):
                if pygame.mouse.get_pressed()[0] == 1:
                    action = True
        return action


button = Button(screen_width // 2 - 40, screen_height // 2 - 20, button_img)
```

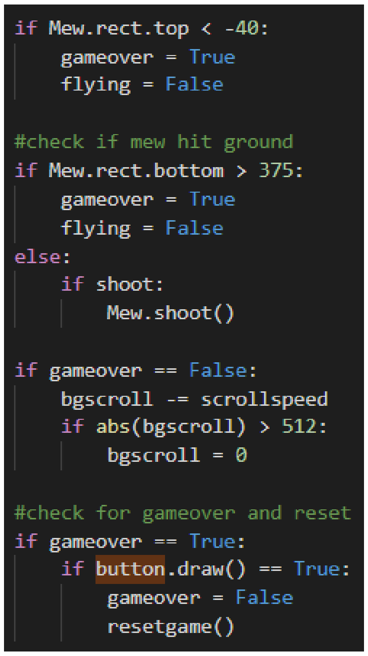
pygame.mouse.get_pos(). Since we want this button to be displayed whenever the game is over, we can use an if statement that declares gameover == True. To identify that the mouse is hovering specifically inside the button, we can use self.rect.collidepoint() to test whether the mouse position is inside the rectangle or not. pygame.mouse.get_pressed() checks whether the button has been pressed or not and if it is equals to 1, the action is returned. Outside of the class button, I decided to make a variable called button which contains the Button class properties and the image itself. I used it to determine where I want to position the button, which in this case is the middle of the screen. I also created a function called resetgame() which puts MEW back into the initial location of when MEW first spawns. This will only occur when the mouse click is registered at 1, which can only happen when the player clicks on the retry button.



```python
def resetgame():
    Mew.rect.x = 20
    Mew.rect.y = int(screen_height/2 - 20)


    action = False
    while gameover == True:
        if pygame.mouse.get_pressed()[0] == 1:
            action = True

    return action
```

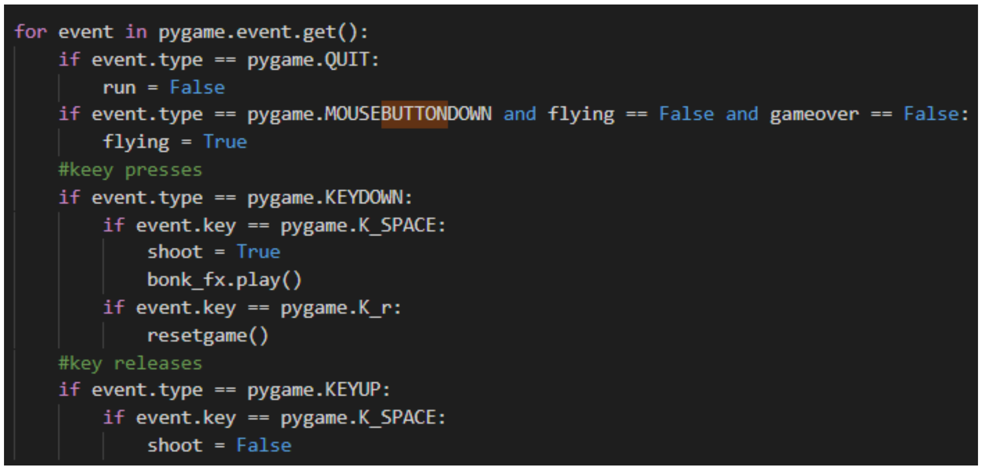
C. Creating Game Over Situations

There are three situations for the game to end, the first one is when MEW flies too high which is specified already in the code. The second situation is when MEW falls all the way to the bottom. The third one is when MEW's health reaches 0 due to the enemies' projectiles, it will explode. Once that happens, the retry button created will appear on the screen. Once the button appears, gameover == False since we want the game to continue. When MEW is not in any of those situations, MEW can still shoot out the heart projectiles. The background will keep on scrolling as well as long as the game isn't over. When the gameover condition is True, the reset button will be drawn out and if it is also TRUE, the resetgame() is called out.



```python
if Mew.rect.top < -40:
    gameover = True
    flying = False

#check if mew hit ground
if Mew.rect.bottom > 375:
    gameover = True
    flying = False
else:
    if shoot:
        Mew.shoot()

if gameover == False:
    bgscroll -= scrollspeed
    if abs(bgscroll) > 512:
        bgscroll = 0

#check for gameover and reset
if gameover == True:
    if button.draw() == True:
        gameover = False
        resetgame()
```

D. Creating Controls



```
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        run = False
    if event.type == pygame.MOUSEBUTTONDOWN and flying == False and gameover == False:
        flying = True
    #keey presses
    if event.type == pygame.KEYDOWN:
        if event.key == pygame.K_SPACE:
            shoot = True
            bonk_fx.play()
        if event.key == pygame.K_r:
            resetgame()
    #key releases
    if event.type == pygame.KEYUP:
        if event.key == pygame.K_SPACE:
            shoot = False
```

pygame.event.get pretty much registers all events from the user into an event queue. Here, I used the for loop and if statements to determine what happens if an event occurs. An event.type is the type of event or we can say, the action it receives. pygame.QUIT happens when the user clicks the window's "X" button which is why run == False. pygame.MOUSEBUTTONDOWN is an event generated whenever a mouse button is pressed or held down. I used an if statement here when the mouse button gets pressed down, MEW will be able to fly/jump. For MEW to be able to shoot projectiles, I used the event key SPACE which is the spacebar of our keyboard. Since we don't want MEW to keep on shooting projectiles continuously, when the player released the spacebar button, projectiles are disabled. The key r is for players to press when they want to restart the game.

E. Pygame Functions Explained:

pygame.init()= initialize all imported pygame modules

pygame.mixer.pre_init()= preset the mixer init arguments

mixer.init()= initialize the mixer module

pygame.display.set_mode()= initialize a window or screen for display

pygame.display.set_caption()= sets the current window's caption

pygame.display.update()= updates portion of the screen for software displays

pygame.quit()= shuts down pygame including the window

pygame.mixer.music.load()= loads the music file in wav format.

pygame.mixer.music.play(-1)= loops and plays the music

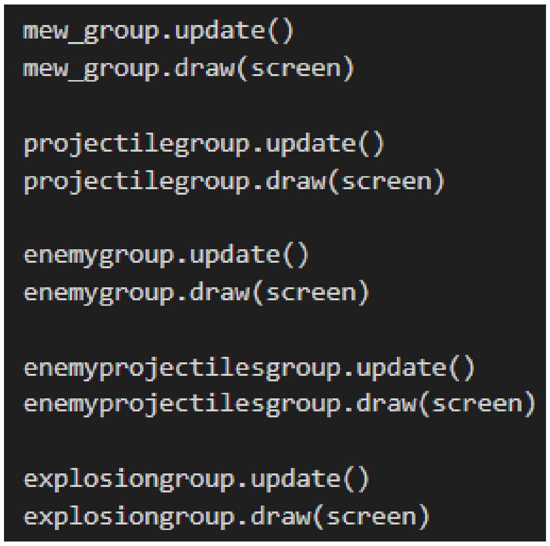
pygame.mixer.music.set_volume()= set and change volume

pygame.mixer.Sound()= Create a new Sound object from a file

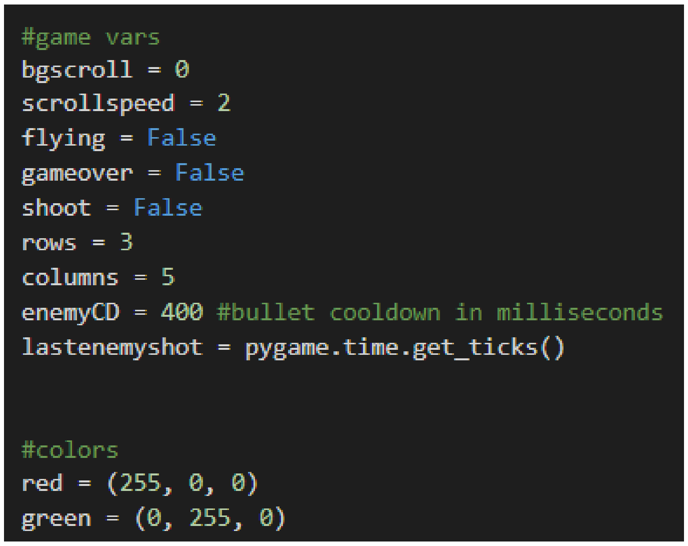
pygame.time.Clock()= used to create a clock object which can be used to keep track of time

pygame.time.get_ticks()= get the time in milliseconds

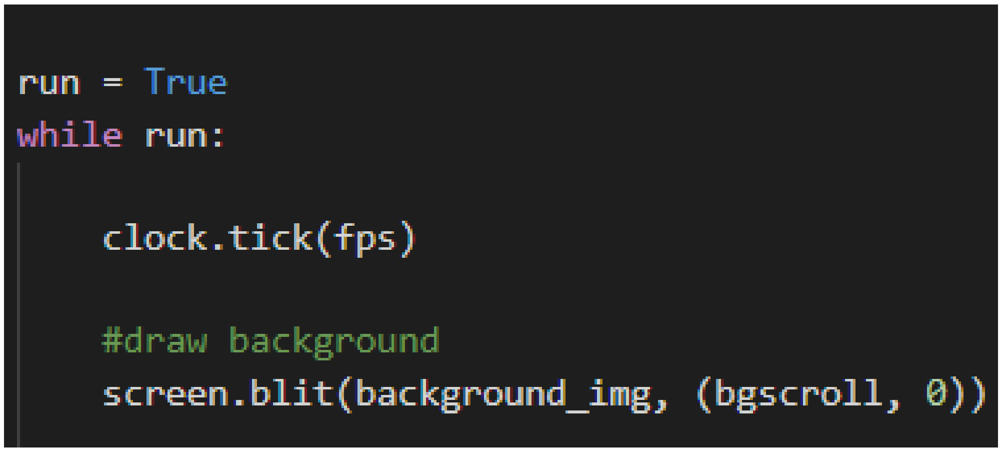
F. Sprites and Background Implemented (bonus game variables):



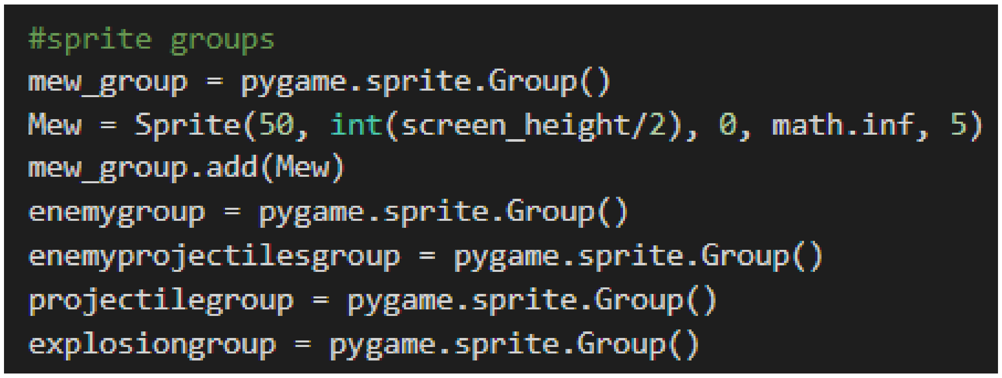
```
mew_group.update()
mew_group.draw(screen)

projectilegroup.update()
projectilegroup.draw(screen)

enemygroup.update()
enemygroup.draw(screen)

enemyprojectilesgroup.update()
enemyprojectilesgroup.draw(screen)

explosiongroup.update()
explosiongroup.draw(screen)
```



```
#game vars
bgscroll = 0
scrollspeed = 2
flying = False
gameover = False
shoot = False
rows = 3
columns = 5
enemyCD = 400 #bullet cooldown in milliseconds
lastenemyshot = pygame.time.get_ticks()


#colors
red = (255, 0, 0)
green = (0, 255, 0)
```



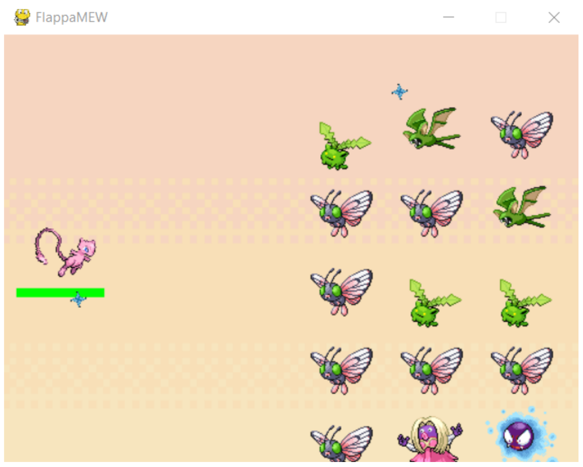
```
run = True
while run:

    clock.tick(fps)

    #draw background
    screen.blit(background_img, (bgscroll, 0))
```

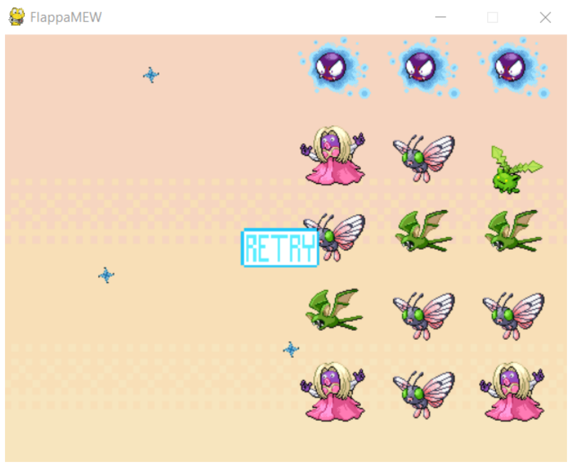


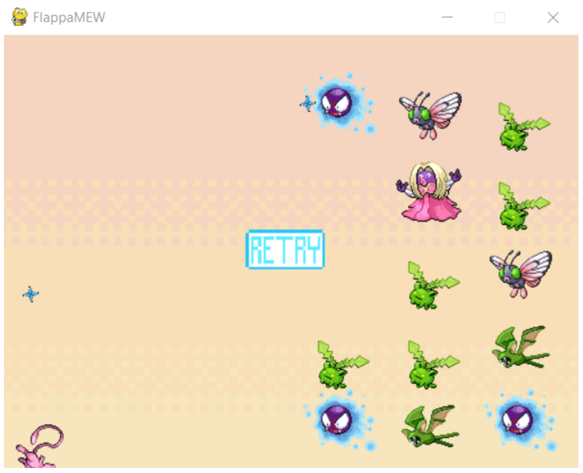
```
#sprite groups
mew_group = pygame.sprite.Group()
Mew = Sprite(50, int(screen_height/2), 0, math.inf, 5)
mew_group.add(Mew)
enemygroup = pygame.sprite.Group()
enemyprojectilesgroup = pygame.sprite.Group()
projectilegroup = pygame.sprite.Group()
explosiongroup = pygame.sprite.Group()
```

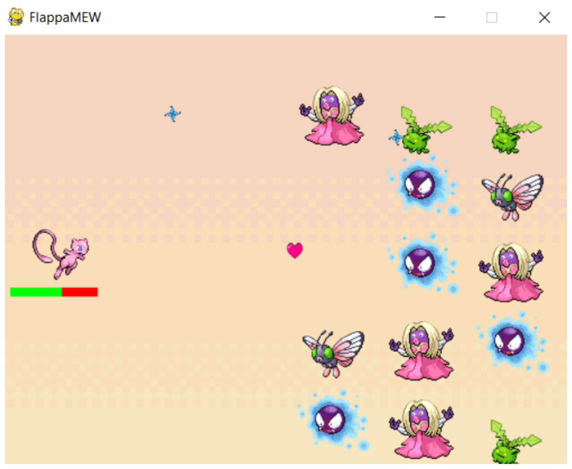
Chp. 4 – Evidence of Working Program:

A. Screenshots:









Chp. 5 – Reflection:

A. Program Evaluation

The game ends up working as expected with a few flaws in it. Here is what works:

- MEW can fly / jump by pressing the left mouse button
- MEW can fall upon not clicking the left mouse button
- MEW can shoot heart projectiles by pressing the space bar that kills the enemy upon collision
- The enemies can hit MEW with their own projectiles which causes MEW to lose some health.
- The enemies moves up and down in a loop
- The enemies keeps on shooting random projectiles as long as they're alive.
- The retry button always appears after MEW's death
- MEW dies whenever it flies too high or reaches the bottom
- When MEW's health bar becomes fully red, MEW dies
- The retry button works upon clicking as it resets MEW back to it's initial spawn point

However there are a few bugs I found upon testing:

- Upon pressing the retry button, it will bring back MEW to it's spawn point however the health doesn't get reset and enemies that are killed won't spawn back.
- When MEW dies due to the projectiles, MEW can't spawn back even after pressing the retry button
- Sometimes when MEW drops all the way down, MEW doesn't die and instead gets stuck at the bottom of the screen

Improvements that can be made:

- Add in more levels or make the enemies respawnable each time it dies
- Add in a score counter so players can keep track of how many enemies they can kill
- Add in a main menu in the beginning so players have the time to prepare before playing

B. Self Reflection & Experience

The project overall was a real challenge for me since I had no experience in programming a game myself or any experience in coding before taking this major. I've also added an extra challenge(and stress) for myself which is to avoid asking any questions/assistance from anyone I know, even though I probably should have. I wanted to limit test myself to see how capable I am on my own. It's only been 4 months since I took this major but I've learned a lot from this single project. Not only that my knowledge on coding has improved, but my knowledge was also put into practice thus improving my problem solving skills. Since this project involved a lot of uses with OOP, I'm comfortable enough now to say that my understanding of it is pretty solid. One thing I wished I did was to start off the project earlier. I started thinking of ideas a week before the project was due. Created the game within 3 days at the cost of sacrificing many hours of sleep while still having to write the documentation report which also took about 3 days to finish. The stress and pressure really took a toll on me but I'm glad it's all over now. To wrap it up, I'm proud of how this project turned out even though there are still a few things I'm unsatisfied about such as the bugs I couldn't fix alone. Since I'm a perfectionist, after this project gets submitted, I'm planning to fix the bugs after finals exams are over and perhaps add the improvements I listed.

references:

https://www.pygame.org/docs/index.html