INFO1111: Computing 1A Professionalism

2023 Semester 1

Self-Learning Report

Submission number: NO.2

Github link:
https://github.sydney.edu.au/INFO1111-2023/INFO1111-SelfLearning

Student name	Dianlei Huang
Student ID	520268838
Topic	Pygame
Levels already achieved	B
Levels in this report	C and D

Contents

1. Level A: Initial Understanding

1.1. Level A Demonstration

1. Importing and Initializing PyGame
2. Setting Up the Display
3. Implement event loops
4. Handling the user input
5. Draw items on the screen

1.2. Learning Approach

Step1: **Research**: First, I will start by searching for Pygame tutorials, guides and documentation online. Look for some resources that are useful for a primer.
Step2: **Install and setup:**Secondly, I install the pygame and follow the instruction on pygame official website.
Step3: **Read documentation and follow tutorials**:Thirdly, I will take some time read the official documentation to learn its features, functions and modules. In addition, I will follow the online teaching video to learn some advanced concepts like graphic and sound.
Step4: **Practice:** After learning the knowledge, in order to truly understand Pygame, I need to experiment with it on my own environment. I can first test each functions including user input, event control loop, sound and effect, and drawing key items on the screen. Experiment with different features and modues to see how they work.
Step5: **Application:** After doing practice and experiment, I can actually create my own game by applying the knowledge I learned before. In the future, I will continue to improve the game and add new features in the further study.

1.3. Challenges and Difficulties

1. When learning pygame, it is quite challenging to learn for a beginner as it requires an understanding of both Python programming and Pygame programming concepts, so this can be quite difficult to learn two things at same time.
2. Debugging can be always challenging when working with Pygame because it involves to many modules and functions working together, this will take a lot of time to figure out an error when you write about hundreds of lines of codes.
3. Sound and music is the specific element that I found more difficult to learn than others. Because I have no experience in audio programming, and it involves bgm, sound effect and MIDI. I also encountered some file incompatibility issues, so adding music to the game was particularly challenging for me.
4. Optimization a pygame project is very difficult because it is related to the effieiency and speed of your algorithm. So it will takes time to optimize your code in order to make the game run more smoothly.

1.4. Learning Sources

Youtube tutorials (1)	Learning some advanced concepts like graphic and sound and so on
Pygame documentation(2)	Learning some basic concepts and syntax of Pygame
Pygame tutorials(3)	Learning how to handle the user input and a lot of events
YouTube pygame lectures(4)	Learning how to actually create a game and apply the skills by using Pygame
Python documentation(5)	Pygame is based on python, Learning the basic syntax and functions

Link(1):https://www.youtube.com/watch?v=jO6qQDNa2UY
Link(2):https://www.pygame.org/docs/
Link(3):https://realpython.com/pygame-a-primer/
Link(4):https://www.youtube.com/watch?v=i6xMBig-pP4list=
PLzMcBGfZo4-lp3jAExUCewBfMx3UZFkh5
Link(5):https://docs.python.org/3/

1.5. Application artifacts

The whole code is in git link I provided before

1.Importing and Initializing PyGame

For the level A, firstly, I successfully import the pygame, and initialize the game library. In addition, some important keyboard is imported so we can call it directly without pygame. :

```
1  #import the pygame and initialize the game library to use all the modules
2  import pygame
3
4  #import some user input so we can directly use it without pygame.
5  from pygame.locals import (
6      K_UP,
7      K_DOWN,
8      K_LEFT,
9      K_RIGHT,
10     K_ESCAPE,
11     KEYDOWN,
12     QUIT,
13 )
14
15 pygame.init()
```

Figure 1: Initialization

2. Setting Up the Display

Then, I initialize some variables such as width, height, x-y-axis and a variable later we will use later in our event control loop. Then I set up the screen by using line 25 with width and height I created before.

```
#Set up the screen and oringinal axis
width = 900
height = 700
x = (width - 60)/2
y = 600
val = 0.5

# Create the screen by using the width and height we previous defined
screen = pygame.display.set_mode([width,height])
```

Figure 2: Setting Up the Display

3. Implement event loops

Implement the event loop is very important in pygame, basically this loop will run forever unless you hit the close tab or escape keyword. First, we set a variable run to be true, and using a for loop to detect whether we reach a event. So I use a key variable that is equal to the key.getPressed(), therefore I can check whether this key entered is escape or not, if it is escaple button or close button, the run will become false and the whole program will just end.

```
# Create the screen by using the width and height we previous defined
screen = pygame.display.set_mode([width,height])

# The main event loop
run = True
while run:
    # When user click the window close buttob, the run will become false
    # the program loop will end
    # Also, when a key escape is pressed, the program will also quit
    for event in pygame.event.get():
        key =  pygame.key.get_pressed()
        if key[K_ESCAPE]:
                run = False
        elif event.type == QUIT:
            run = False

```

Figure 3: Implement event loops

4. Handling the user input

This part, we are initialize the variable called keys which represent the key that get pressed. So this part is in the main event loop, whenever a user press left, right, up or down, the x or y axis will be changed according to the user input, so this x and y axis will be later used in the program.

```
    # Handling the user inpput whenever a user press left,right,up or down button
    # Item will move according to the button pressed in the loop
    keys = pygame.key.get_pressed()
    if keys[K_LEFT]:
        x -= val
    if keys[K_RIGHT]:
        x += val
    if keys[K_UP]:
        y -= val
    if keys[K_DOWN]:
        y += val
```

Figure 4: Handling the user input

5. Draw Items on the screen

In this part, we are actually drawing the items on the screen and quit the program if the user enter escape or close button. Now I will explain the following code line by line. line 53 just fill the screen with alice blue color. Line 56, I create an item with size(60*60) by using the pygame.Surface(module from pygame), then I just draw 4 rectangle with same blue color as a boundary of the screen, pygame.draw.rect() use three parameters, first is the screen I want to draw, and the second parameter is the RGB color value, and third parameter include 4 value , first two value is the position on the screen I want to draw, and last two values are the size of this rectangle. After that , I also drawed four cirle by using pygame,draw.circle. and the last parameter is the radius(50) of this circle. The line 72 use blit function that we call using screen, so this can put the item I created bfore on screen at the location of (x,y), the position will be updated in the main event loop because whenever user press a button[up,down,left,right], the item will move in the screen, the reason why item will keep moving is because in the main event loop and it will loop forever unless the user close it. Therefore, this simple pygame program can implemented

```
52      # Fill screen with alice blue color
53      screen.fill((240,248,255))
54
55      # Creare an item with size(70*70) by using surface
56      item = pygame.Surface((60,60))
57      # Fill the item with red color
58      item.fill((255,0,0))
59
60      # Draw some rectangle as a boundary of scrren with blue color
61      pygame.draw.rect(screen, (30,144,255),(0,0,900,25))
62      pygame.draw.rect(screen, (30,144,255),(0,0,25,700))
63      pygame.draw.rect(screen, (30,144,255),(0,675,900,25))
64      pygame.draw.rect(screen, (30,144,255),(875,0,25,700))
65      # Draw some circle on the screen
66      pygame.draw.circle(screen, (138,43,226), (200, 100), 50)
67      pygame.draw.circle(screen, (255,211,155), (700, 100), 50)
68      pygame.draw.circle(screen, (152,245,255), (250, 250), 50)
69      pygame.draw.circle(screen, (118,238,0), (500, 250), 50)
70
71      # Draw the item on screen at the location of (x,y)
72      screen.blit(item, (x,y))
73
74      #Update the scrren
75      pygame.display.flip()
76
77  #Quit the pygame whe the loop is finished
78  pygame.quit()
```

Figure 5: Draw Items on the screen

4 functions listed in 1.1, and you can download it from my github link to try it.

Example

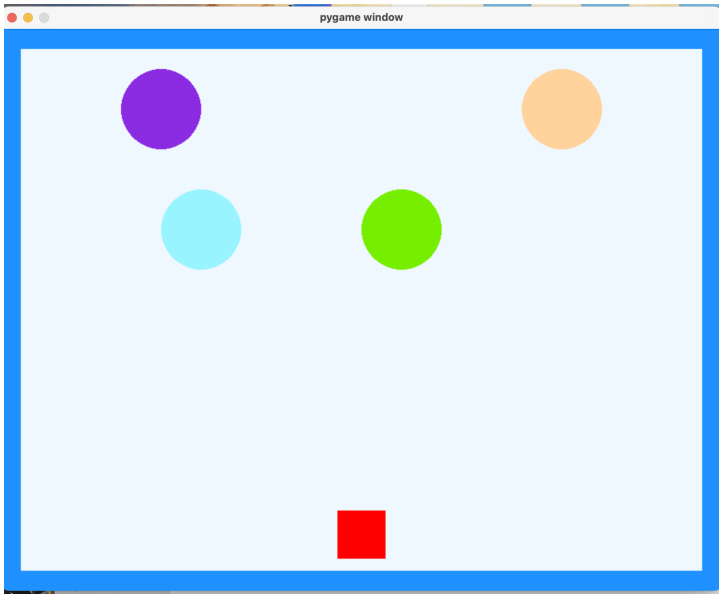


Figure 6: Example1

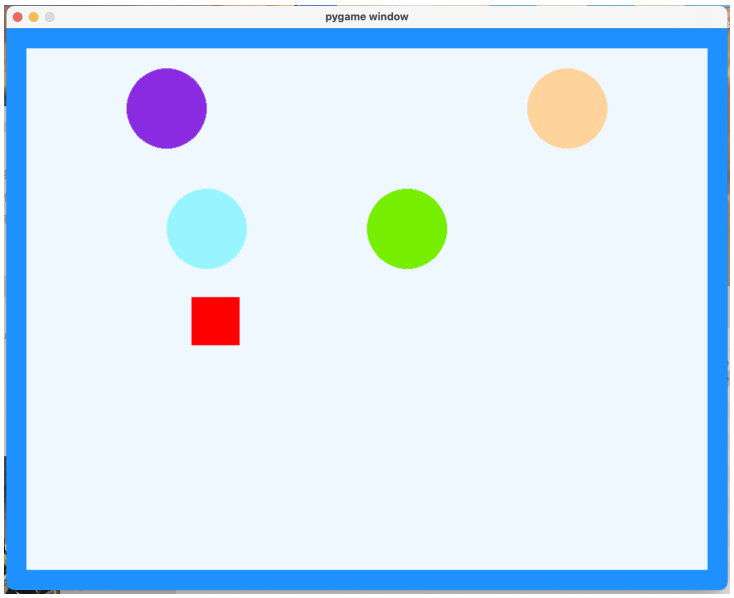


Figure 7: Example2

2. Level B: Basic Application

2.1. Level B Demonstration

For this level, I will be actually using the skills list above to create a game demo. Basically , game is a classic 2D plane game and the name is called "Flying Ace". So basically the user will use keyboard as user input to control the plane that on the screen, and the enemy are coming from the top of the screen. The aim for this game is to avoid the enemy , and later in level D, the user will be able to shoot the enemy. To be specific, the player plane and enemy plane (as sprite) are drew on the screen, and background music is added in order to improve gamer experience. In addition, the program can respond the event such as mouse-click , or keyboard pressed to end the game. Last but not least, the enemy plane is generated by using the user event which is created by myself. The idea of this game is based on an online tutorial, but the whole program is done by myself[1], and the actual game is also my own work.

2.2. Application artifacts

One of the most important concept in pygame, which is sprite. Basically, sprite is just a 2d representation of something, and it is a very powerful module that has a lot of built in function.[1] Firstly I will introduce the Player.py which contain a Plane class that extends the sprite in its constructor.

```
1  import pygame
2  # Define player by extend the sepecifc class Sprite,
3  # and use super to call the constructor of parent
4
5  #import some user input so we can use those key easier.
6  from pygame.locals import (
7      K_UP,
8      K_DOWN,
9      K_LEFT,
10     K_RIGHT,
11     K_ESCAPE,
12     KEYDOWN,
13     QUIT,
14 )
15
16 # Define a player object that extends Sprite(2D representation)
17 # Use picture of plane to replace the surface
18 class Plane(pygame.sprite.Sprite):
19     def __init__(self):
20         super(Plane, self).__init__()
21         # Load the picture from directory and use convert to optimize the surface
22         self.image = pygame.image.load("plane.png").convert_alpha()
23         self.image = pygame.transform.scale(self.image,(80,80))
24         # Use RLEACCEL (slow down the image) to optimize the game
25         self.image.set_colorkey((255, 255, 255), pygame.RLEACCEL)
26         # rect is used for get location of image, so we can change it by input
27         # get_rect() returns the rectangle of image, and we set the original position
28         self.rect = self.image.get_rect(left = 200, bottom = 750)
```

Figure 8: Plane class

Basically, We create a Plane object that is a sprite, and it has its own constructor. For line21-22, we load the image of the plane from the directory by using the built in function **pygame.image.load**, and self.image is the variable that load the image, self is just a keyword in python that refer to the object we created later[2]. Then we want to use convert_alpha to optimize the surface so it will run more smoothly. In addition, the transform.scale is used to change the size of that image. Furthermore, we use RLEACCEL to optimize the game, so the function RLEACCEL will optimize the hardware[1]. the line 28 is to set a variable called rect which will returns the rectangle of image, so we can

actually know the position of this plane, and later we can change the get_rect() according to the user input. The original position of this plane is set at left = 200, bottom =750, which means the left boundary of surface will be located at 200 on x-axis, and bottom boundary of surface will be located at 750 on y-axis.

```
29
30      # Update the position of sprite by detecting the user input
31      # move_ip stands for move in place, with x and y axis as parameter
32      def update(self, pressed_key):
33          if pressed_key[K_UP]:
34              self.rect.move_ip(0, -1)
35          if pressed_key[K_DOWN]:
36              self.rect.move_ip(0, 1)
37          if pressed_key[K_LEFT]:
38              self.rect.move_ip(-1, 0)
39          if pressed_key[K_RIGHT]:
40              self.rect.move_ip(1, 0)
41
42          # Keep our plane in the screen
43          # If it hit left corner, we keep the value of
44          # the X-coordinate of the left side of the rectangle be 0
45          if self.rect.left < 0:
46              self.rect.left = 0
47          # If it hit right corner, we keep the value of
48          # the X-coordinate of the right side of the rectangle be width
49          if self.rect.right > 500:
50              self.rect.right = 500
51          # If it hit top corner, we keep the value of
52          # the Y-coordinate of the top side of the rectangle be 0
53          if self.rect.top <= 0:
54              self.rect.top = 0
55          # If it hit bottom corner, we keep the value of
56          # the Y-coordinate of the bottom side of the rectangle be height
57          if self.rect.bottom >= 750:
58              self.rect.bottom = 750
```

Figure 9: Handling the user input

After the constructor is built, we can write another method called update to update the position of our plane we just created by using rect and load image. So basically the function only take one parameter which is pressed_key, so the update method will detect the key that user pressed each time, if it is up error key, the move_ip(move in place) will be executed[3], and it will update the rectangle of image with position of (0, -1), that means the plane will not move in x-axis, but it will move in y axis with 1 step up.Line 35-40 does the same function, the only difference is they check different pressed key and move the plane according to the key. Basically, this is how we handling the use input in pygame. In addition, we want to keep our plane inside the screen, line 45-58 check the position of our plane in the update method, and make our plane inside the screen. For line 45-46, it check if the plane hit the left corner, which means the position of rect.left is smaller than 0, we just make it equal to 0, so it will never go out of the screen. Line 49-50 means if it hit right corner, we keep the value of X-coordinate of the right side of rectangle to be width which is 500. Line 53-54, check if it hit top corner, we will keep the value of the Y-coordinate of the top side rectangle to be 0. Last we check the bottom, the pygame allow us to assess the top,bottom, right and left position of a rectangle, so we can easily keep the player inside the screen.

Secondly(figure10), I will introduce the enemy class which is also a sprite but we can not control it. Here, we are doing the same thing as Plane class, we define a Enemy object that extends Sprite, but we initialize a variable called y as a local variable, which

represent the starting position of enemy[2]. Because the enemy always come from the top of the screen, so we let y be 0 in the constructor. line 24-26 is used to load image of the enemy and also change the size. rect variable is a little bit different between player, so we want a enemy that randomly appear in the map we created, so we also use self.image.get_rect, but with different parameter. This time, we let the position of center of this enemy become **(random.randint(20,480),random.randint(0,0))**[4]. We want a enemy that always appear at the top of the screen, so the y-coordinate is 0, but it can appear appear anywhere on the X-axis [20,480] because we need to consider the size of enemy. In addition, the update method will take one parameter that is screen, so every time the update method is called, the y axis is adding 0.5, which means the the enemy will go down 0.5 pixel. Then the screen.blit is used again to actually draw the enemy on the screen with location of **(self.rect.centrex,self.y)**[2]. Also, we check if the enemy is going out of the screen, the sprite will kill itself if it is going out of the screen.

```
17  # Define a player object that extends Sprite(2D representation)
18  # Use picture of plane to replace the surface
19  class Enemy(pygame.sprite.Sprite):
20      y = 0
21      def __init__(self,y = 0):
22          super(Enemy, self).__init__()
23          # Load the picture from directory and use convert to optimize the surface
24          self.image = pygame.image.load("enemy.png").convert_alpha()
25          # Change the size of image
26          self.image = pygame.transform.scale(self.image,(40,40))
27          # rect is used for get location of image, so we can change it by input
28          # get_rect() returns the rectangle of image, and we set the random position to enemy
29          self.rect = self.image.get_rect(center = (random.randint(20,480),random.randint(0,0)))
30
31      # Update the position of enemy by using screen.blit
32      def update(self,screen):
33          self.y += 0.5
34          screen.blit(self.image,(self.rect.centerx,self.y))
35          #The plane will move from top to bottom, and when it hits the bottom
36          # it will just be removed by kill()
37          if self.rect.bottom > 700:
38              self.kill()
```

Figure 10: Enemy class

Thirdly, the main program will be explained.

```python
# Import the pygame and some key user input so we can directly use it
import pygame
from Player import Plane
from Enemy import Enemy

from pygame.locals import (
    K_UP,
    K_DOWN,
    K_LEFT,
    K_RIGHT,
    K_ESCAPE,
    KEYDOWN,
    QUIT,
)

# Initialize the pygame environment
pygame.init()


#Set up the screen and oringinal axis
width = 500
height = 750
x = (width - 60)/2
y = 600
val = 0.5


# Set up the screen and set up the name for this game
screen = pygame.display.set_mode([width,height])
pygame.display.set_caption("Flying Ace ")

# Set up the game icon by load the image and use set_icon
icon = pygame.image.load("plane.png")
pygame.display.set_icon(icon)

# Set up the backgroun image
background = pygame.image.load("back2.jpg")

# Load a background sound clip and begin playing it.
# The sound never end by setting the named parameter loops=-1.
pygame.mixer.music.load("bkgmusic.wav")
pygame.mixer.music.play(-1)
```

Figure 11: Main program setup

First(Figure11), we need to import the pygame. Then from **Player.py,** we import Plane, and from Enemy.py, we need to import Enemy class from **Enemy.py.** After doing that, we import some local variable and initialize the pygame environment by using **pygame.init()** [2] Then, the width and height variable are initialized for screen set up. Line 29 - 30 just setup the screen with built in function, and the two parameters are given to set up the screen just like level A. After that, the pygame.display.set_caption [2] can be used to setup the title of my game which is called "Flying Ace". After that, the game icon is loaded by using the picture that in the directory. The general rule of loading a image from directory is using **pygame.image.load()** [2]. Line 34 uses set_icon function to actually implement the icon. After setting this up, line 37 loads the background image from the folder in the same way.

The game need music that can improve the user experience, so the background music is added by using pygame built in function **mixer.music.load()**[2]. Then we need the music to loop until the end of the game, so we can set up an infinite loop and make the parameter of loop (-1), so that the music can be played throughout the game.

After finishing the basic setup, now we need to set up the player and enemy before start the main even loop

```
44  # Creating the object from Player
45  p = Plane()
46
47  # Create a list of enemy
48  ls_enemy = []
49
50  # Define an event that is called add_enemy
51  add_enemy = pygame.USEREVENT
52  # The add_enemy event will be occured every 0.5 second
53  pygame.time.set_timer(add_enemy, 500)
54
55  # The main event loop
56  run = True
57  while run:
58      # When user click the window close buttob, the run will become false
59      # the program loop will end
60      # Also, when a key escape is pressed, the program will also quit
61      for event in pygame.event.get():
62          key =  pygame.key.get_pressed()
63          if key[K_ESCAPE]:
64              run = False
65          elif event.type == QUIT:
66              run = False
67          # Also when the left mouse button is pressed, the game will also quit
68          elif event.type == pygame.MOUSEBUTTONDOWN:
69              if event.key == 1:
70                  run = False
71          # Every 1 second, the add_enemy event will occur, and add one enemy
72          elif event.type == add_enemy:
73              e = Enemy()
74              ls_enemy.append(e)
75
```

Figure 12: Creating object and setup the event

(Figure12) The plane object is created by using p = Plane(). For the enemy, we want multiple enemy, so a list is needed to store all the enemy objects. Pygame allow us to create our own event which is called USEREVENT, and it can be defined py using pygame.USEREVENT[1]. The event is called add_enemy, so basically every time this event occur, we want add an enemy object in the list, then we can set a timer, so the line 53 is basically telling the add_enemy will be occurred every 500 millisecond.

The main event loop is starting ar line 56, as usual, the variable run is initialized to be true. The first for loop in that event loop is loop through every event from pygame.event.get()[2]. From line 62-66, the varaiable key is the key that user enter during the game, and if key escape is pressed or the close tap is clicked, the run will become false, which means the main game loop is over. In addition, the mousebuttondown event is detected, if the type of event is equal to mousebuttondown, and the event.key is eqaul to 1, that means when the left mouse button is clicked, the game will also quit. The last elif statement in event control loop is check whenever the add_enemy event is occurred, the Enemy object is created and being added to the list[4]. So the whole part I done here is called responding to event.

```
77      # Fit the background
78      screen.blit(background,(0,0))
79
80      # Detect the keys that are pressed
81      keys = pygame.key.get_pressed()
82      # Put player at the specific postion we want and
83      # Update the postion according to the user input
84      screen.blit(p.image,p.rect)
85      p.update(keys)
86
87      # Display all the enemy currently in the list
88      for i in ls_enemy:
89          i.update(screen)
90
91
92      pygame.display.update()
93
94
95
96
97  pygame.quit()
98
```

Figure 13: Draw and update the player and enemy

Finally(Figure13), for the game part, the screen is filled by background image we initialised before at the center of the screen. line 81 set a variable called keys to detected which key is pressed, because we already write setup and update method in Player, we can just call the attributes and method by using variable p. The player object is located at the specific position we want, line 84 pass two parameters, the first one is the image of plane, and second one is the location of that plane. Then, the player's position is updated according to the key user entered. Last thing, we do is display all the enemy currently in the list by using a for loop. This can be done by calling the **update(screen)** function in the enemy class. After doing that, the screen is updated by using **pygame.display.update()**[2].

The whole program finished all the requirements in level B, and the codes can be found in github link, and all references can be found in bibliography which contains both python and pygame documentation and some online tutorials. The example picture is given below

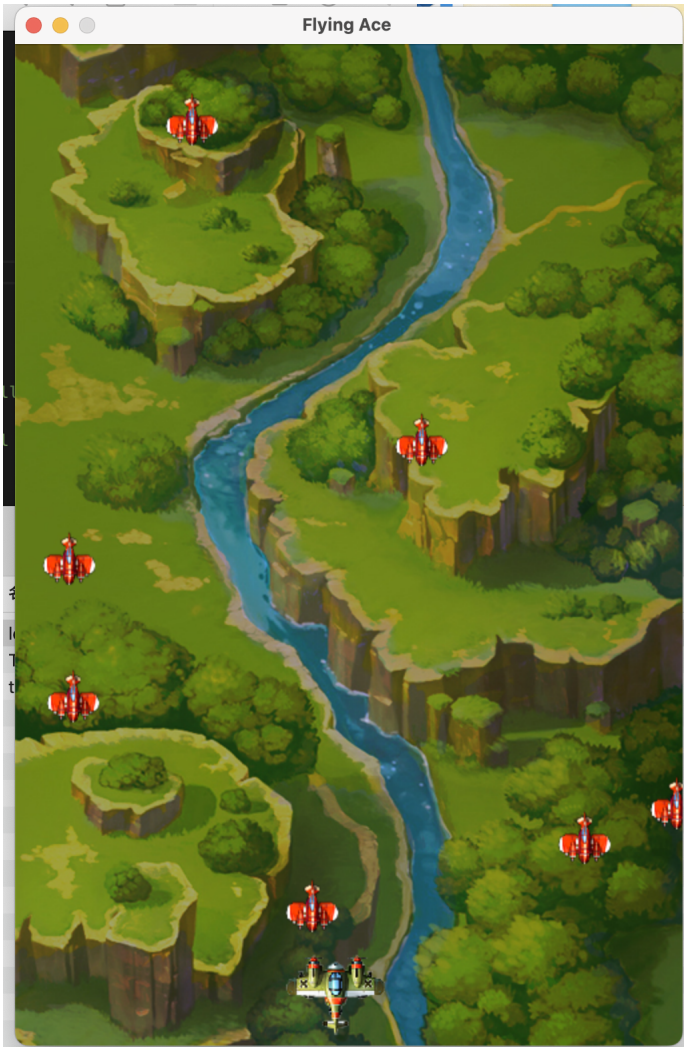


Figure 14: Example1

3. Level C: Deeper Understanding

3.1. Strengths

Firstly, Compare to other game programming language such as java, Unity and C++, it is easy for beginner to learn because the syntax and library of pygame are much easier than other language.[5]
Secondly, pygame can basically run in any platform including Windows, mac Os and Linux due to its independence.
Thirdly, since the game can be easily developed in pygame, pygame's simplicity allows game designer to create prototype rapidly, and making it an ideal choice for game jams, educational projects, or small games.
Lastly, Pygame has a huge open source library that allows user to modify the source code without worrying about the fees.[2]

3.2. Weaknesses

Firstly, compare to Unity3D and Unreal engine, Pygame is not the first choice for high-performance game such as some 3A games. Because of functionality of pygame, the frame rates and performance will be reduced when handling complex games.[6]
Secondly, Pygame 's graphic and physical engine are relatively simple, so it can not be used to create 3D games.
Thirdly, Pygame is not designed for commercial projects such as some professional game development team, because it lacks many of the advanced features such as 3D graphics, physical engine and low_level sound effect.[6]
Lastly, Pygame can not be used to design mobile games.

3.3. Usefulness

Pygame can be particularly useful in creating a game designed to teach programming concepts to some beginner programmers. [7]The simplicity make pygame an ideal choice for this scenario. Also, it can provides many useful game design ideas for beginners who want to work in the game industry.
In this scenario, because of the platform independence, any beginner can use any computer to learn the programming knowledge within pygame in a much more fun way. In addition, the game can be easily and quickly designed in pygame, so the beginner gain a sense of accomplishment quickly and become more interested in learning programming in the future.

3.4. Key Question 1

Pygame is a library that primarily designed for creating 2D games using Python syntax[2], so it provides functionality for graphics, audio, and input handling, which simplifies the game development process for 2D games. However, it is not specifically built for creating 3D first-person games.
But it is possible for developer to create 3D first-person games in pygame, because it dose have some support for 3D graphic by using OpenGL[8], but comparing to other powerful tool such as Unity3D and unreal engine, Pygame is not a best choice for creating 3D game due to its primary library and lack of performance.
In conclusion, it is not useful for creating 3D first-person games, but you can still do it.

3.5. Key Question 2

In general, both python and pygame are beginner friendly, so it definitely require a basic understanding of python programming concepts such as variables, loops, functions and class. However, Pygame has a lot of built-in functions and modules for developing games,

so these additional knowledge will not be covered while learning Python. Therefore, in order to use Pygame effectively, a beginner just need to learn some basic python programming concepts and syntax and then learn a range of modules about Pygame such as sprite and graphics.
In conclusion , a beginner does not need to be proficient in python, but only needs to understand the OOP(Objected Oriented Programming) concepts and be able to use some of the core python syntax, so that he use pygame more effectively.

4. Level D: Evolution of skills

4.1. Level D Demonstration

At this level, the entire game's features have been greatly improved, and many new features have been added to the game.Firstly, the game now includes a start screen where players can use their mouse to click and choose to either start or close the game. Once the game begins, players can now control the aircraft to dodge enemies and shoot bullets to destroy them. Each time an enemy is destroyed, the score increases by 100 points. The game has also added collision detection to identify collisions between the player and enemies. When any enemy collides with the player, the player loses one life. Players have a total of three lives, displayed at the bottom right corner of the screen.
There are two types of enemies in the game: small aircraft, which are slow and small, and medium-sized aircraft, which are fast and larger. The game has added explosion sound effects and bullet firing sound effects. If the player dies more than three times, the game over screen will appear, giving the player the option to either restart the game or exit. In addition to that , user can either press keyboard 1 or button on the right top of the screen to pause the game and press resume button to resume. Once the player reach the point of 3000, the screen will ask the user to either continue to hard level or quit the game. All enemies can now shoot bullets and generate them at a faster rate, as well as move faster.
In conclusion, the following applications are developed in Level D:
Add sound effect
Add bullet to the plane, so the bullet can eliminate enemy
Add multiple lives
Add collision so the player will die after colliding with enemy
Add Boss enemy
Add death screen and opening screen
Add pause and stop the game
Add boss enemy and they can attack in hard level
Add the different level of the game
Add beginning page and end page

4.2. Application artifacts

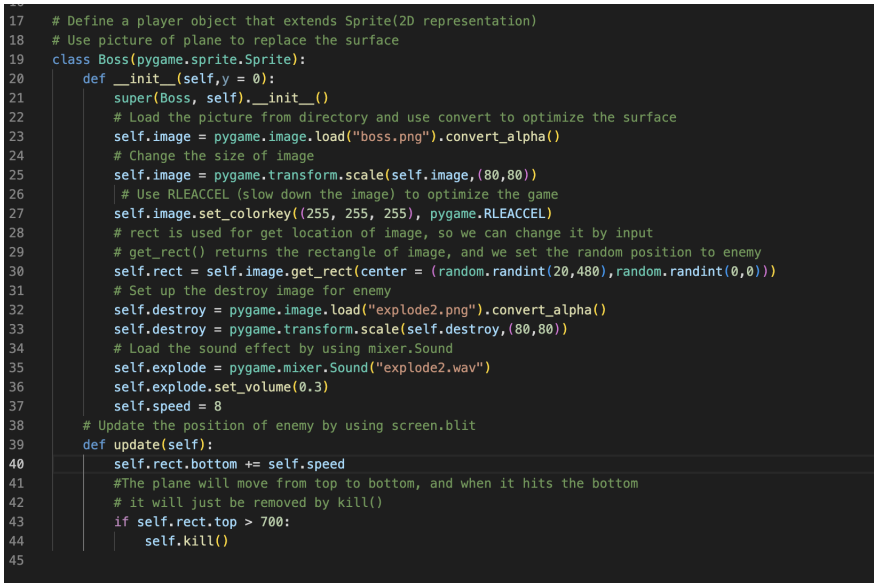
To begin with, the bullet class has been created as a sprite class so the Plane can fire bullets to attack the enemy.

```
3  class Bullet(pygame.sprite.Sprite):
4      # Define player by extend the sepecifc class Sprite,
5      # and use super to call the constructor of parent
6      def __init__(self,position_x,position_y,direction):
7          pygame.sprite.Sprite.__init__(self)
8
9          #Load the picture from directory and use convert to optimize the surface
10         self.image = pygame.image.load("bullet.png").convert_alpha()
11         # Change the size of the image
12         self.image = pygame.transform.scale(self.image,(20,35))
13          # Use RLEACCEL (slow down the image) to optimize the game
14         self.image.set_colorkey((255, 255, 255), pygame.RLEACCEL)
15         # Get the rectangle of this image
16         self.rect = self.image.get_rect()
17         # Setup the speed
18         self.speed = 5
19         # Setup the primary position of bullet
20         self.rect.centerx = position_x
21         self.rect.centery = position_y
22         #Setup the direction of the bullet
23         self.direction = direction
24
25
26     # Update the bullet position according to the direction
27     def update(self):
28         # When self.direction is true, the bulluet is moving upward
29         if self.direction:
30             self.rect.top -= self.speed
31             # If the bullet hit the top of the screen, it will be removed
32             if self.rect.top < 0:
33                 self.kill()
34         else:
35             self.rect.top += self.speed
36             if self.rect.top > 700:
37                 self.kill()
38
```

Figure 15: The Bullet class

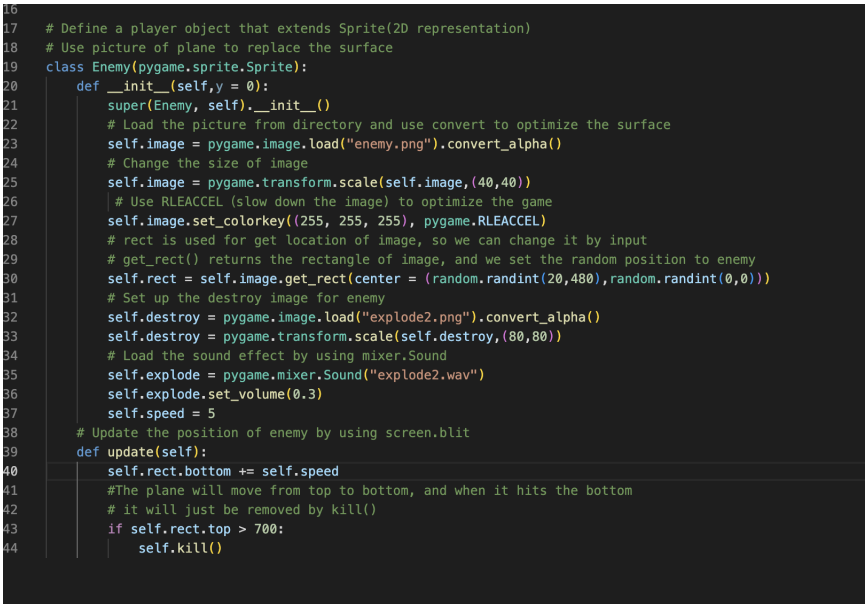
Just like the player class and enemy class, basically the bullet class is extending the sprite class so it can use some bulit-in function directly from the sprite class, and then call the constructor of sprite class in order to initialize the class. For the constructor, there are three parameters including position_x , position_y and direction to determine where the bullet should be placed. Then, the image is loaded by using exactly same function used before, and use pygame.transform.scale[2] to change the size of the bullet. line 16 is used to get the rectangle of that image so we can draw them easily on the screen. In addition, the position_x and position_y are set up to be the centerx and centery of the rectangle, and then setup the direction of the bullet, whether it is going forward or going backward. After the constructor is bulit, the function update is used to update the bullet position according tot he direction. Basically , first if the direction is true, then the bullet is moving upward, and every time the update function is called, the position of rectangle of bullet sprite will be minus speed. Then check if the bullet hit the top of the screen, the sprite will just be killed and removed from the screen in line 32. Overall, the bullet class is created and we can use it in the main game file.

Secondly, the boss class is created as a sprite class to represent the big and more powerful enemy in the game.



```
17  # Define a player object that extends Sprite(2D representation)
18  # Use picture of plane to replace the surface
19  class Boss(pygame.sprite.Sprite):
20      def __init__(self,y = 0):
21          super(Boss, self).__init__()
22          # Load the picture from directory and use convert to optimize the surface
23          self.image = pygame.image.load("boss.png").convert_alpha()
24          # Change the size of image
25          self.image = pygame.transform.scale(self.image,(80,80))
26          # Use RLEACCEL (slow down the image) to optimize the game
27          self.image.set_colorkey((255, 255, 255), pygame.RLEACCEL)
28          # rect is used for get location of image, so we can change it by input
29          # get_rect() returns the rectangle of image, and we set the random position to enemy
30          self.rect = self.image.get_rect(center = (random.randint(20,480),random.randint(0,0)))
31          # Set up the destroy image for enemy
32          self.destroy = pygame.image.load("explode2.png").convert_alpha()
33          self.destroy = pygame.transform.scale(self.destroy,(80,80))
34          # Load the sound effect by using mixer.Sound
35          self.explode = pygame.mixer.Sound("explode2.wav")
36          self.explode.set_volume(0.3)
37          self.speed = 8
38      # Update the position of enemy by using screen.blit
39      def update(self):
40          self.rect.bottom += self.speed
41          #The plane will move from top to bottom, and when it hits the bottom
42          # it will just be removed by kill()
43          if self.rect.top > 700:
44              self.kill()
45
```

Figure 16: The Boss class



```
16
17  # Define a player object that extends Sprite(2D representation)
18  # Use picture of plane to replace the surface
19  class Enemy(pygame.sprite.Sprite):
20      def __init__(self,y = 0):
21          super(Enemy, self).__init__()
22          # Load the picture from directory and use convert to optimize the surface
23          self.image = pygame.image.load("enemy.png").convert_alpha()
24          # Change the size of image
25          self.image = pygame.transform.scale(self.image,(40,40))
26          # Use RLEACCEL (slow down the image) to optimize the game
27          self.image.set_colorkey((255, 255, 255), pygame.RLEACCEL)
28          # rect is used for get location of image, so we can change it by input
29          # get_rect() returns the rectangle of image, and we set the random position to enemy
30          self.rect = self.image.get_rect(center = (random.randint(20,480),random.randint(0,0)))
31          # Set up the destroy image for enemy
32          self.destroy = pygame.image.load("explode2.png").convert_alpha()
33          self.destroy = pygame.transform.scale(self.destroy,(80,80))
34          # Load the sound effect by using mixer.Sound
35          self.explode = pygame.mixer.Sound("explode2.wav")
36          self.explode.set_volume(0.3)
37          self.speed = 5
38      # Update the position of enemy by using screen.blit
39      def update(self):
40          self.rect.bottom += self.speed
41          #The plane will move from top to bottom, and when it hits the bottom
42          # it will just be removed by kill()
43          if self.rect.top > 700:
44              self.kill()
```

Figure 17: The Enemy class

Basically it is totally same with the enemy class I explained in level B above, the only difference is from line 30 to line 36. For both enemy and boss class will have a self.destroy variable that load the image of explosion, and the explode sound is defined by using pygame.mixer.Sound()[2]. So every time if the enemy or boss is attacked by the bullet sprite, the image will be replaced by destroy image and the explosion sound will be play in order to enhance the user experience. line 36 is just set the volume of that explosion sound. In addition all of the enemies now have their own speed variable, so when the update function is called, enemies' position will be updated according to the speed.

Thirdly, I will introduce the Player class, some new functions and features has been

added to the player class.

```
16  # Define a player object that extends Sprite(2D representation)
17  # Use picture of plane to replace the surface
18  class Plane(pygame.sprite.Sprite):
19      def __init__(self):
20          super(Plane, self).__init__()
21          # Load the picture from directory and use convert to optimize the surface
22          self.image = pygame.image.load("plane.png").convert_alpha()
23          self.image = pygame.transform.scale(self.image,(80,80))
24          # Use RLEACCEL (slow down the image) to optimize the game
25          self.image.set_colorkey((255, 255, 255), pygame.RLEACCEL)
26          # rect is used for get location of image, so we can change it by input
27          # get_rect() returns the rectangle of image, and we set the original position
28          self.rect = self.image.get_rect(left = 200, bottom = 750)
29          # Set up the destroy image for plane
30          self.destroy = pygame.image.load("explode1.png").convert_alpha()
31          self.destroy = pygame.transform.scale(self.destroy,(100,100))
32          # Load the sound effect by using mixer.Sound
33          self.explode = pygame.mixer.Sound("explode.wav")
34          self.explode.set_volume(0.4)
35      # Update the position of sprite by detecting the user input
36      # move_ip stands for move in place, with x and y axis as parameter
37      def update(self, pressed_key):
38          if pressed_key[K_UP]:
39              self.rect.move_ip(0, -10)
40          if pressed_key[K_DOWN]:
41              self.rect.move_ip(0, 10)
42          if pressed_key[K_LEFT]:
43              self.rect.move_ip(-10, 0)
44          if pressed_key[K_RIGHT]:
45              self.rect.move_ip(10, 0)
46
```

Figure 18: The Player class

```
46
47          # Keep our plane in the screen
48          # If it hit left corner, we keep the value of
49          # the X-coordinate of the left side of the rectangle be 0
50          if self.rect.left < 0:
51              self.rect.left = 0
52          # If it hit right corner, we keep the value of
53          # the X-coordinate of the right side of the rectangle be width
54          if self.rect.right > 500:
55              self.rect.right = 500
56          # If it hit top corner, we keep the value of
57          # the Y-coordinate of the top side of the rectangle be 0
58          if self.rect.top <= 0:
59              self.rect.top = 0
60          # If it hit bottom corner, we keep the value of
61          # the Y-coordinate of the bottom side of the rectangle be height
62          if self.rect.bottom >= 750:
63              self.rect.bottom = 750
64
65      # Reset the position of the plane
66      def reset(self):
67          self.rect = self.image.get_rect(left = 200, bottom = 750)
68
69
70
```

Figure 19: The Player class

For the player class, line 30 to line 34 are added to the player's constructor. It does the same function just like the enemy and boss class. So the destroy image is loaded by picture in the local directory, and this image has been re scaled at line 31, then load the explosion sound by using mixer.Sound.[2]. Furthermore, a new function has been defined in the player class, which is the reset function. This function is used to reset the position of a player, so at line 67, the rectangle of player has be set to the original place by just changing the position of get_rect. Every time the player's plane dies, it will be resurrected at the initial position.

For now, every sprite class has been developed and these sprite will be used in the main program, so now I will introduce the main program.

```
1  # Import the pygame and some key user input so we can directly use it
2  import pygame
3  from Player import Plane
4  from Enemy import Enemy
5  from Bullet import Bullet
6  from Boss import Boss
7
8  from pygame.locals import (
9      K_UP,
10     K_DOWN,
11     K_LEFT,
12     K_RIGHT,
13     K_ESCAPE,
14     KEYDOWN,
15     QUIT,
16 )
17
18 # Initialize the pygame environment
19 pygame.init()
20 pygame.mixer.init()
21
22
23 #Set up the screen and oringinal axis
24 width = 500
25 height = 750
26 x = (width - 60)/2
27 y = 600
28 val = 0.5
29
30 #Set up the font and size for score by using bulit-in function
31 # Load the font from local directory
32 score_font = pygame.font.Font("bold.ttf",30)
33 score_color = (255,255,255)
34
```

Figure 20: Setup of main program

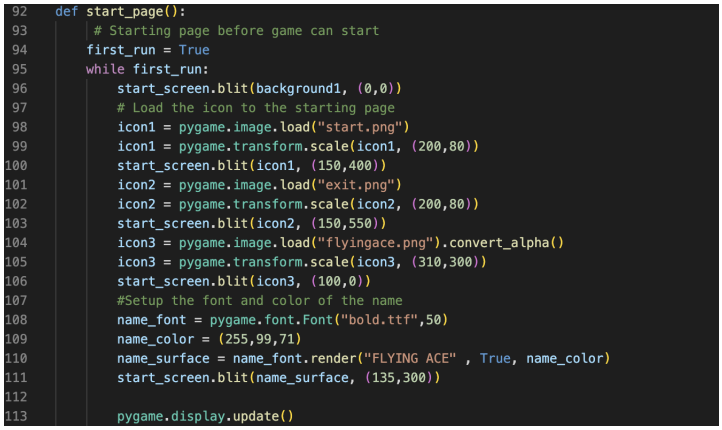
Before we can start the main game loop , we need to first import the pygame and all other sprite classes at line2 - 7, and then initialize the pygame environment and sound environment by using pygame.init() and pygame.mixer.init().[2] . Then we can setup the width, height variable and etc. Because we need to show the player's current score in real-time, we need to use a bulit in function called pygame.font.Font[2]. This function take two parameters , one is the font.ttf file from local directory, and another one is the size of that font. By doing that, we can set up the font and size for the score, and then setup the score_color to be white.

```
29
30 #Set up the font and size for score by using bulit-in function
31 # Load the font from local directory
32 score_font = pygame.font.Font("bold.ttf",30)
33 score_color = (255,255,255)
34
35 # Setup the game start screen
36 start_screen  = pygame.display.set_mode([width,height])
37 #Load the background of gamestart
38 background1 = pygame.image.load("gamestart.png")
39 # Set up the end game page
40 end_screen  = pygame.display.set_mode([width,height])
41 background2 = pygame.image.load("overover.jpg")
42 # Load the sound effect for gameover
43 gameover = pygame.mixer.Sound("gameover.wav")
44 gameover.set_volume(0.5)
```

Figure 21: More setup

Then, we need to do some more setup, the start screen and end screen are defined by using pygame.display.set_mode which just take width and height as the parameter of a screen. Two background image are loaded at line 38 and 41 so we can directly use it in the program after. Then the gameover sound effect is loaded at line 43 py using the bulit-in function pygame.mixer.Sound[2]

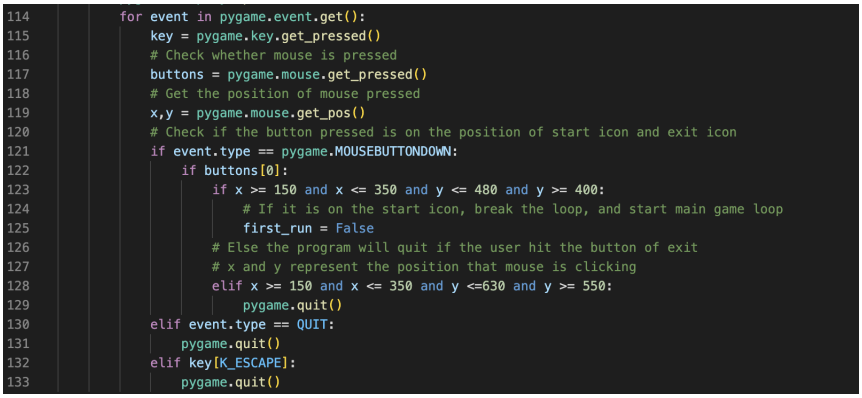
Now I will introduce one of the new features which are start-screen and end-screen. During level B, there is no start screen for user to choose whether start the game or quit the game, so for level D this feature is implemented.



```
92  def start_page():
93      # Starting page before game can start
94      first_run = True
95      while first_run:
96          start_screen.blit(background1, (0,0))
97          # Load the icon to the starting page
98          icon1 = pygame.image.load("start.png")
99          icon1 = pygame.transform.scale(icon1, (200,80))
100         start_screen.blit(icon1, (150,400))
101         icon2 = pygame.image.load("exit.png")
102         icon2 = pygame.transform.scale(icon2, (200,80))
103         start_screen.blit(icon2, (150,550))
104         icon3 = pygame.image.load("flyingace.png").convert_alpha()
105         icon3 = pygame.transform.scale(icon3, (310,300))
106         start_screen.blit(icon3, (100,0))
107         #Setup the font and color of the name
108         name_font = pygame.font.Font("bold.ttf",50)
109         name_color = (255,99,71)
110         name_surface = name_font.render("FLYING ACE" , True, name_color)
111         start_screen.blit(name_surface, (135,300))
112
113         pygame.display.update()
```

Figure 22: start screen function

we can define a function called start_page, so we can call it before the game start. Firstly, the variable first_run is fined to be True, just like other event loop, the background image will be drew on the screen by using blit function[2], basically every time we want to draw something on the screen, we need to use screen.blit. Then some icons have been loaded and re scaled from line 98 to 106, and all the icons will be draw on the screen. Two important icon is the icon1 and icon2 which are start button and exit button that user can press. Because we know the exact location and scale of those two icon, so we can use them later for the mouse detection. at line 108-109, the name of the game is using the same font we created before but different size. After that a surface of name is created by using render function[2]. This method allows us to generate a surface with the desired input string using specific fonts and colors, and we can draw this surface on the screen. After all the setup, we always use pygame.display.update() to update the screen.



```
114         for event in pygame.event.get():
115             key = pygame.key.get_pressed()
116             # Check whether mouse is pressed
117             buttons = pygame.mouse.get_pressed()
118             # Get the position of mouse pressed
119             x,y = pygame.mouse.get_pos()
120             # Check if the button pressed is on the position of start icon and exit icon
121             if event.type == pygame.MOUSEBUTTONDOWN:
122                 if buttons[0]:
123                     if x >= 150 and x <= 350 and y <= 480 and y >= 400:
124                         # If it is on the start icon, break the loop, and start main game loop
125                         first_run = False
126                     # Else the program will quit if the user hit the button of exit
127                     # x and y represent the position that mouse is clicking
128                     elif x >= 150 and x <= 350 and y <=630 and y >= 550:
129                         pygame.quit()
130             elif event.type == QUIT:
131                 pygame.quit()
132             elif key[K_ESCAPE]:
133                 pygame.quit()
```

Figure 23: event control loop for start screen

Every while loop need a event control loop inside the while loop so we know when we should end the while loop for start-screen. Basically, the line 114 is used to detect all the event in pygame, then the key variable is considered as the keyboard that we pressed and buttons is considered as the mouse we detect, x and y variable are used to get the position of the mouse. From line 121-129, we check whether the mouse is pointing on the start icon or exit icon. First, we check on line 121 if the type of this event is a mouse click. Then, on line 122, we check if it is the left mouse button that was clicked. In other words, when the left mouse button is pressed, we will proceed to lines 123 and 128 to determine whether

the position of the mouse click belongs to the range of the start icon and exit icon that we previously drew. If the mouse click the start icon, then the first_run variable will be false and this while loop in the function will end , the main program can start after the function call, else if user hit the button of exit, the whole program will quit. Line130 - 133 are used to check if user click keyboard escape or click the close button on the left top of the screen, the program will also quit.

For the end screen page , the function is basically same with the start screen page.

```
46  # Design a function for end page if a plane dead over 3 times
47  def end_page(score):
48      gameover.play(0)
49      last_run = True
50      while last_run:
51          end_screen.blit(background2, (0,0))
52          # Load the icon to the ending page
53          icon1 = pygame.image.load("gameover.png")
54          icon1 = pygame.transform.scale(icon1, (250,100))
55          icon2 = pygame.image.load("tryagain.png")
56          icon2 = pygame.transform.scale(icon2, (200,80))
57          icon3 = pygame.image.load("exit.png")
58          icon3 = pygame.transform.scale(icon3, (200,80))
59          end_screen.blit(icon1, (125,200))
60          end_screen.blit(icon2, (150,400))
61          end_screen.blit(icon3, (150,550))
62           #Setup the font and color of the score
63          score_font = pygame.font.Font("bold.ttf",40)
64          score_color = (255,255,255)
65          score_surface = score_font.render("Your Final Score: %s" % str(score), True, score_color)
66          end_screen.blit(score_surface, (70,100))
67          pygame.display.update()
```

Figure 24: end screen function

```
67          pygame.display.update()
68          for event in pygame.event.get():
69              key = pygame.key.get_pressed()
70              # Check whether mouse is pressed
71              buttons = pygame.mouse.get_pressed()
72              # Get the position of mouse pressed
73              x,y = pygame.mouse.get_pos()
74              # Check if the button pressed is on the position of start icon and exit icon
75              if event.type == pygame.MOUSEBUTTONDOWN:
76                  if buttons[0]:
77                      if x >= 150 and x <= 350 and y <= 480 and y >= 400:
78                          # If it is on the restart icon, call the main function to restart the game
79                          main1(False,0)
80                      # if not juse quit the game
81                      elif x >= 150 and x <= 350 and y <=630 and y >= 550:
82                          pygame.quit()
83              # The game will also be quit if user enter esc or press close button
84              elif event.type == QUIT:
85                  last_run = False
86                  pygame.quit()
87              elif key[K_ESCAPE]:
88                  last_run = False
89                  pygame.quit()
90      pygame.quit()
91
```

Figure 25: event control loop for end screen

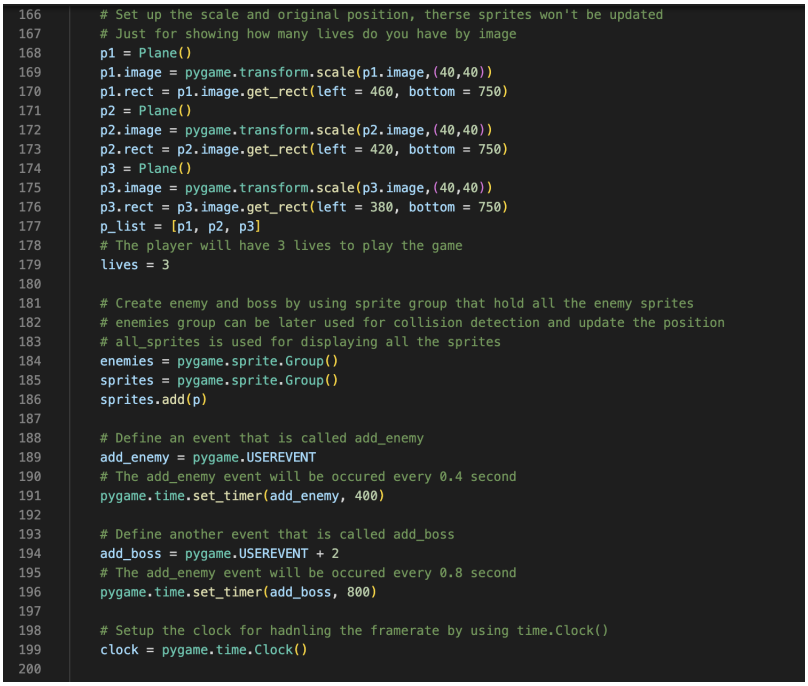
Some differences: if the end_page function called, the gameover sound effect will play one time, and line 63 - 66 will draw the currently score on the screen, so in order to call the function of end_page, you need to pass a parameter called score. Instead of start button, there will be a restart button, and once the user click the restart button, the main game program will be called to restart the game, two parameters used to call the main function will be explained later. Except for the loaded images being different, the other parts are roughly the same as the previous function.

Now, I will introduce the main function which include the main game loop. Firstly, all the variables and class need to be set up:

```
135  def main1(hard_level,score):
136      # Set up the screen and set up the name for this game
137      screen = pygame.display.set_mode([width,height])
138      pygame.display.set_caption("Flying Ace ")
139
140      # Set up the game icon by load the image and use set_icon
141      icon = pygame.image.load("plane.png")
142      pygame.display.set_icon(icon)
143
144      # Set up the backgroun image
145      background = pygame.image.load("back2.jpg")
146
147      # Load a background sound clip and begin playing it.
148      # The sound never end by setting the named parameter loops=-1.
149      pygame.mixer.music.load("bkgmusic.wav")
150      pygame.mixer.music.play(-1)
151
152      # Load the sound effect for explosion by using mixer.Sound
153      explode = pygame.mixer.Sound("explode.wav")
154      explode.set_volume(0.5)
155
156      # Load the sound effect for shooting
157      shoot = pygame.mixer.Sound("shoot.wav")
158      shoot.set_volume(0.5)
159
160      # Load the picture of pause
161      pause_icon = pygame.image.load("pause.png")
162      pause_icon = pygame.transform.scale(pause_icon,(40,40))
163
164      # Creating the object from Player and other 3 plane for counting lives
165      p = Plane()
```

Figure 26: Main loop setup

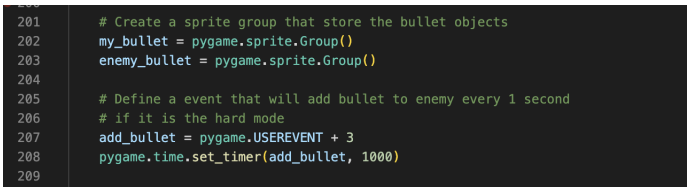
The parameters need to be pass through the main1 function are Boolean value hard_level and the current score, so if the hard level is true, the main function1 will start the game in hard level, and use the player's existing score. line136 - 150 are the setup for screen, game icon, background image and background music, this part is explained in level B. For level D, the sound effect is added when a plane fire the bullet or die. Line 153 use variable explode to store the pygame.mixer.Sound("explode.wav")[2], and line 157 use the same function to store the sound for shoot. Line 161 add a pause icon at the top right corner of the screen, if user press that button the game will pause. After doing all the setup for screen and sound, we need to generate all the sprite instances.



```
166     # Set up the scale and original position, therse sprites won't be updated
167     # Just for showing how many lives do you have by image
168     p1 = Plane()
169     p1.image = pygame.transform.scale(p1.image,(40,40))
170     p1.rect = p1.image.get_rect(left = 460, bottom = 750)
171     p2 = Plane()
172     p2.image = pygame.transform.scale(p2.image,(40,40))
173     p2.rect = p2.image.get_rect(left = 420, bottom = 750)
174     p3 = Plane()
175     p3.image = pygame.transform.scale(p3.image,(40,40))
176     p3.rect = p3.image.get_rect(left = 380, bottom = 750)
177     p_list = [p1, p2, p3]
178     # The player will have 3 lives to play the game
179     lives = 3
180
181     # Create enemy and boss by using sprite group that hold all the enemy sprites
182     # enemies group can be later used for collision detection and update the position
183     # all_sprites is used for displaying all the sprites
184     enemies = pygame.sprite.Group()
185     sprites = pygame.sprite.Group()
186     sprites.add(p)
187
188     # Define an event that is called add_enemy
189     add_enemy = pygame.USEREVENT
190     # The add_enemy event will be occured every 0.4 second
191     pygame.time.set_timer(add_enemy, 400)
192
193     # Define another event that is called add_boss
194     add_boss = pygame.USEREVENT + 2
195     # The add_enemy event will be occured every 0.8 second
196     pygame.time.set_timer(add_boss, 800)
197
198     # Setup the clock for hadnling the framerate by using time.Clock()
199     clock = pygame.time.Clock()
200
```

Figure 27: Main function sprites setup

To begin with, one player sprite instance is created by using variable p, and three other plane sprites are created just for showing how many lives the user currently have. From line 168 - 177, three player objects are created and re scaled, the position of these three objects are unchanged, they will be placed at the right bottom of the screen, so if you are playing the game. The images of these three planes will show the character's life. Each time the character dies, one plane will be popped out of this list, so a small plane will disappear from the lower right corner of the screen. This feature is a newly added UI design of the game". Line 179 is just saying player will have 3 lives to play the game. then create enemy and boss by using the sprite group that hold all the enemy sprites. pygame.sprite.Group() bulit in function is very useful to detect the collision between different sprite group()[1]. Variable sprites is used for displaying all the sprites in the game including enemies, bullet and plane. In pygame, developer can define their own event by using USEREVENT[2], so add_enemy and add_boss event are created to determine when the game will add enemy sprites. Line 191 and line 196 are used to set a timer for these event, the add_enemy event will be occurred every 0.4 second and add_boss event will be occurred every 0.8 second. line 199 use bulit in function to setup a clock for handling the frame rate.



```
201     # Create a sprite group that store the bullet objects
202     my_bullet = pygame.sprite.Group()
203     enemy_bullet = pygame.sprite.Group()
204
205     # Define a event that will add bullet to enemy every 1 second
206     # if it is the hard mode
207     add_bullet = pygame.USEREVENT + 3
208     pygame.time.set_timer(add_bullet, 1000)
209
```

Figure 28: Main function sprites setup

Variable my_bullet is used to collect the bullet sprites for player and enemy_bullet is used to collect the bullet for enemies. Another event add_bullet is set up, but this event will only occurred when the game is hard level.

By finishing all the setup, the main game loop can start:

```
213     # The main event loop
214     run = True
215     pause = False
216     flag = False
217     while run:
218         # When user click the window close buttob, the run will become false
219         # the program loop will end
220         # Also, when a key escape is pressed, the program will also quit
221         for event in pygame.event.get():
222             key =  pygame.key.get_pressed()
223             #Check the button
224             buttons = pygame.mouse.get_pressed()
225             # Get the position of mouse pressed
226             x,y = pygame.mouse.get_pos()
227             if key[K_ESCAPE]:
228                 run = False
229             elif event.type == QUIT:
230                 run = False
231             # Every 0.5 second, the add_enemy event will occur, and add one enemy
232             elif event.type == add_enemy:
233                 e = Enemy()
234                 enemies.add(e)
235                 sprites.add(e)
236             # Every 0.5 second, the add_enemy event will occur, and add one boss to the sprite group
237             elif event.type == add_boss:
238                 boss = Boss()
239                 sprites.add(boss)
240                 enemies.add(boss)
```

Figure 29: event control loop for main game loop

```
241             # Using key_space to shoot the bullet
242             elif key[pygame.K_SPACE]:
243                 # The position of bullet will be originally placed at top of the plane
244                 # Create a new bullet
245                 b = Bullet(p.rect.centerx,p.rect.top, True)
246                 # Add one bullet to the sprite group
247                 my_bullet.add(b)
248                 sprites.add(b)
249                 # Play sound effect
250                 shoot.play(0)
251               # if the user press the top right button or press the keyboard 1
252             # The pause will become True and the pause loop will start
253             elif event.type == pygame.MOUSEBUTTONDOWN:
254                 if buttons[0]:
255                     if x >= 460 and x <= 500 and y <= 40 and y >= 0:
256                         pause = True
257             elif key[pygame.K_1]:
258                 pause = True
259             elif hard_level:
260                 if event.type == add_bullet:
261                     # For all the enemy, they can attack at the hard_level
262                     # Every 2 second the enemy will fire a bullet
263                     for i in enemies:
264                         i.speed += 2
265                         b = Bullet(i.rect.centerx, i.rect.bottom, False)
266                         b.speed = i.speed + 1
267                         b.image = pygame.image.load("enemy_bullet.png").convert_alpha()
268                         b.image = pygame.transform.scale(b.image,(30,50))
269                         enemy_bullet.add(b)
270                         sprites.add(b)
271
```

Figure 30: event control loop for main game loop

Three variables are created, run is used to determine when to stop the game loop, pause is used to determine when the pause loop will begin, and flag is used to determine when the hard_level is started. In the while run loop, we need to first create a event control loop to handle all the event in the game, many of the functions are similar to the event control loop in function start and end screen. Other than that, the program will detect the event we created by USEREVENT, Line 232- 235 are saying that once the add_enemy event occur, one small enemy will be generated, and it will be added to both enemies sprite group and all_sprites group, and add_boss does the exactly same function. From line 242, it will detect if the user press keyboard_space, one bullet will be generated and the position will be originally placed at the top of the plane, because

the last parameter we used to create Bullet object is True, this bullet will go from bottom to top. After doing that, the bullet will be added to sprite group, and sound effect will be played by using shoot.play()[2]. In addition, the mouse click will be detected to see whether a user press the pause button or press the keyboard , either of choices will make the variable 'pause' will become true, and then the following 'while pause' loop will run, so the player can pause the game anytime, anywhere. The last event we need to handle is whether it is hard level, so if it is the hard_level, the add_bullet event will occur to add one bullet object for all the enemy in enemies sprite group, and all of the enemies will speed up(line 264), the bullet will be placed at the bottom of the every enemy, and the speed of the bullet will be enemy's speed + 1. The images of bullets fired by the player and the enemy are different, which is good for distinguishing enemy attacks. Therefore, the image of each bullet is reloaded as another image, and finally, all bullets will be added to the 'enemy_sprite' and 'sprites' group.

The pause loop:

```
272         # If the user choose pause the game, this loop will run instead of main loop
273         while pause:
274             # Load the pause screen
275             screen.blit(background2, (0,0))
276             # Load three icon to the pause screen, and one is resume button,
277             # Another one is exit button
278             icon1 = pygame.image.load("resume.png")
279             icon1 = pygame.transform.scale(icon1, (200,80))
280             icon2 = pygame.image.load("exit.png")
281             icon2 = pygame.transform.scale(icon2, (200,80))
282             icon3 = pygame.image.load("break.png")
283             icon3 = pygame.transform.scale(icon3, (300,300))
284             screen.blit(icon3, (100,50))
285             screen.blit(icon2, (150,550))
286             screen.blit(icon1, (150,400))
287             pygame.display.update()
288             for event in pygame.event.get():
289                 key = pygame.key.get_pressed()
290                 # Check whether mouse is pressed
291                 buttons = pygame.mouse.get_pressed()
292                 # Get the position of mouse pressed
293                 x,y = pygame.mouse.get_pos()
294                 # Check if the button pressed is on the position of resume icon and exit icon
295                 if event.type == pygame.MOUSEBUTTONDOWN:
296                     if buttons[0]:
297                         if x >= 150 and x <= 350 and y <= 480 and y >= 400:
298                             # If it is on the resume icon, break the loop, and back to main game loop
299                             pause = False
300                         # Else the program will quit if the user hit the button of exit
301                         # x and y represent the position that mouse is clicking
302                         elif x >= 150 and x <= 350 and y <=630 and y >= 550:
303                             pygame.quit()
304                 elif event.type == QUIT:
305                     pygame.quit()
306                 elif key[K_ESCAPE]:
307                     pygame.quit()
```

Figure 31: Pause loop

After the main event control loop, a pause loop will be placed after the main event control loop, if the user choose pause the game, this loop will run instead of main loop. From line275 - 286 are the same thing I did in start screen page, and end screen page, basically when the user press pause button, this page will show up with a resume button and exit button. The only difference is images that loaded are different, and also from line 295 - 307 are the event control loop for pause screen page. If the user use right mouse button to click the resume icon, the pause variable will become false, so this loop will end(line297-299), else if user click the exit button, the whole program will quit. Also, user can always use keyboard escape to quit the game.

Hard_level screen:

```
        while flag:
            background3 = pygame.image.load("hard_level.png")
            background3 = pygame.transform.scale(background3,(width,height))
            screen.blit(background3, (0,0))
            iconx = pygame.image.load("congra.png")
            iconx = pygame.transform.scale(iconx, (400,300))
            icony = pygame.image.load("doyou.png")
            icony = pygame.transform.scale(icony, (400,300))
            continue_icon = pygame.image.load("continue.png")
            continue_icon = pygame.transform.scale(continue_icon, (200,80))
            exit_icon = pygame.image.load("exit.png")
            exit_icon = pygame.transform.scale(exit_icon, (200,80))
            screen.blit(iconx, (50,0))
            screen.blit(icony, (50,100))
            screen.blit(continue_icon,(150,400))
            screen.blit(exit_icon, (150,550))
            pygame.display.update()
            for event in pygame.event.get():
                key = pygame.key.get_pressed()
                # Check whether mouse is pressed
                buttons = pygame.mouse.get_pressed()
                # Get the position of mouse pressed
                x,y = pygame.mouse.get_pos()
                # Check if the button pressed is on the position of resume icon and exit icon
                if event.type == pygame.MOUSEBUTTONDOWN:
                    if buttons[0]:
                        if x >= 150 and x <= 350 and y <= 480 and y >= 400:
                            # If it is on the continue icon, break the loop, and continue main game loop
                            main1(hard_level,5000)
                            flag = False
                        # Else the program will quit if the user hit the button of exit
                        # x and y represent the position that mouse is clicking
                        elif x >= 150 and x <= 350 and y <=630 and y >= 550:
                            pygame.quit()
                elif event.type == QUIT:
                    pygame.quit()
```

Figure 32: flag loop

When this loop begin, it means the flag is true, which also means the hard_level is true, therefore a new screen will be occurred to ask whether user want to go to the hard endless level. So if user use left button to click the continue button, the main1 function will be recalled but with different parameter , one is the True for hard_hard level and the score will now be set up to 5000, and flag will become false in order to break the loop. If the value of the 'hard level' is True, then the previous main event control loop will execute the setting of the 'hard level', which will speed up the enemies' movement and enable them to fire bullets.

Draw all the sprites on the screen and update their location:

```
            # Detect the keys that are pressed
            keys = pygame.key.get_pressed()
            # Put player at the specific postion we want and
            # Update the postion according to the user input
            p.update(keys)

            # Update all the enemy positon in that sprite group
            enemies.update()

            #Update all the bullet position in the sprite group
            my_bullet.update()

            #Update all the bullet position of enemy
            enemy_bullet.update()

            # Fit the background
            screen.blit(background,(0,0))

            # Draw all the sprites in that sprite group on the screen
            for i in sprites:
                screen.blit(i.image, i.rect)

            # Draw the lives you currently have on the screen
            for players in p_list:
                screen.blit(players.image, players.rect)

            # Draw the pause button on the righttop of the screen
            screen.blit(pause_icon,(460,0))
```

Figure 33: Drawing all sprites

After all the loop detection, in the main game loop, the press key well be detected, and player will move according to the key(line348, 351). From line 353 and 360 are basically update all the sprites we created before, after calling the update function, their positions will be updated. Then fit the background by using screen.blit. After that, we need to draw all the sprites in 'sprites' group, so we can simply use a for loop to draw every single sprite on the screen(line 366-367). In addition, draw the players in p_list to show the user the lives on the screen. Line 373 is drawing the pause icon on the right top of the screen by using x and y axis.

Collision detection: Collision detection is very important in every game we created.

```
376         # Check if enemy colide with player by using bulit in function
377         # .spritecollideany()accepts a Sprite and a Group as parameters
378         # and it will automatically check if the rectangle of player
379         # intersects with any rectangle in enemies group
380         if pygame.sprite.spritecollideany(p,enemies):
381             # if that is the case, the player sprite will be killed , the game is over
382             # Check if player still have lives, if not end the game and kill the sprite
383             if lives == 0:
384                 # Draw the explosion image at the position of plane
385                 screen.blit(p.destroy, p.rect)
386                 # Play the sound of explosion
387                 explode.play(0)
388                 p.kill()
389                 #Display the gameover page
390                 end_page(score)
391             # if the player still have lives, reset the plane
392             else:
393                 screen.blit(p.destroy, p.rect)
394                 # minus one lives
395                 lives -= 1
396                 # Reset the position of plane
397                 p.reset()
398                 explode.play(0)
399                 # remove one plane that represent you have one plane less
400                 p_list.pop()
401
```

Figure 34: Collision detection for player and enemies

This part is used to check if enemy collide with player by using the bulit-in function pygame.sprite.spritecolidany(), this method take two parameters, first is an sprite object and second one is sprite group. It is checking if the rectangle of player object intersects with any rectangle in enemies sprite group.[9]. Then it check whether player still have lives, if it has, lives will minus by 1, and draw the destroy image on the screen(line 393), and the sound will be played. if the player has no lives, the player object will be killed, and then call the end_page function with current score to show the end page.

```
402
403         # Set up a variable called dict_colide that will
404         # check if there is any colision between two group my_bullet and enemies
405         # The bulit-in function groupcolide will return a dictionary with elements colide in both group
406         dict_colide = pygame.sprite.groupcollide(enemies,my_bullet,False,False)
407         # Loop through the diectionary and kill all the sprites in that dictionary
408         for key,value in dict_colide.items():
409             for elements in value:
410                 # kill the bullet sprite
411                 elements.kill()
412                 sprites.remove(elements)
413                 # Draw the destroy picture at the postion of enemy
414                 screen.blit(key.destroy, key.rect)
415                 # Sound effect when an enemy is destroyed
416                 key.explode.play(0)
417                 # kill the ememy sprite
418                 key.kill()
419                 sprites.remove(key)
420             # Add score if a player kill the enemies successfully
421             score += 100
422
```

Figure 35: Collision detection for bullet and enemies

For this part, we are checking if any of the sprites in my_bullet collide with any of the

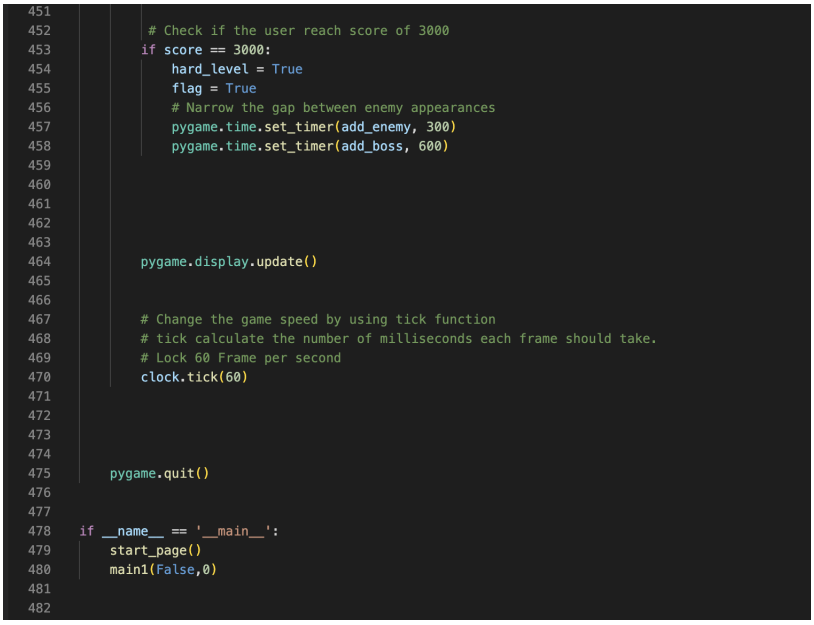
sprites in enemies. By using the built-in function groupcolide, this will return a dictionary with all the elements colide in both group.[9] All we need to do is loop through the key and value in that dictionary we get, and kill the collided bullets and enemies, and remove them from the corresponding sprite group. Line 406 is the dictionary variable we created to receive all the objects that collides in both enemies and my_bullet group.

```
422
423         # Check if the plane is hit by enemy's bullet
424         if pygame.sprite.spritecollideany(p,enemy_bullet):
425             # if that is the case, the player sprite will be killed , the game is over
426             # Check if player still have lives, if not end the game and kill the sprite
427             if lives == 0:
428                 # Draw the explosion image at the position of plane
429                 screen.blit(p.destroy, p.rect)
430                 # Play the sound of explosion
431                 explode.play(0)
432                 p.kill()
433                 #Display the gameover page
434                 end_page(score)
435             # if the player still have lives, reset the plane
436             else:
437                 screen.blit(p.destroy, p.rect)
438                 # minus one lives
439                 lives -= 1
440                 # Reset the position of plane
441                 p.reset()
442                 explode.play(0)
443                 # remove one plane that represent you have one plane less
444                 p_list.pop()
445
446         # Setup the score surface by using render function
447         # render takes three arguments which is text, whether antalias and the color of score
448         score_surface = score_font.render("Score: %s" % str(score), True, score_color)
449         # Draw the score surface on the screen
450         screen.blit(score_surface,(10, 5))
451
```

Figure 36: Collision detection for enemy bullet and player

Last thing but not least, we still need to check whether the plane hit by enemy's bullet, the function is exactly same with collision between player and enemy, the only difference is sprite group now change to enemy_bullet. After that, set up the score surface by using render function, and this function takes three arguments which is text, whether antalias and the color of text[2]. The score is drew on the screen (Line 448, 450).

Check if the player has reached 3000 points:



```
451
452             # Check if the user reach score of 3000
453             if score == 3000:
454                 hard_level = True
455                 flag = True
456                 # Narrow the gap between enemy appearances
457                 pygame.time.set_timer(add_enemy, 300)
458                 pygame.time.set_timer(add_boss, 600)
459
460
461
462
463
464         pygame.display.update()
465
466
467         # Change the game speed by using tick function
468         # tick calculate the number of milliseconds each frame should take.
469         # Lock 60 Frame per second
470         clock.tick(60)
471
472
473
474
475     pygame.quit()
476
477
478 if __name__ == '__main__':
479     start_page()
480     main1(False,0)
481
482
```

Figure 37: Check point

Finally, check if the user reach the score of 3000, if that is the case, both high_level and flag will be true, so for the next loop, the game will enter the hard_level page that asks user if they want to continue play in hard_mode. Furthermore, I narrow the gap between enemy appearances. After finish all the game setting, use pygam.display.upadte() to update the screen, and change the game speed by using tick function, tick calculate the number of milliseconds each frame should take, I lock the game to 60FPS, so it will run the same speed in any hardware.[2].(line 470) For line 478 - line 480, before call the main method, always call start_page first. In conclusion, this game has completed all of the new features mentioned above. The codes can be found in github link, and all references can be found in bibliography which contains both python and pygame documentation and some online tutorials. Here are some game example picture:

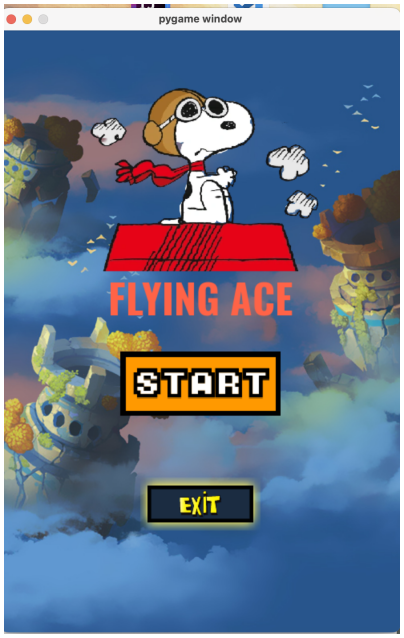


Figure 38: Example1

Figure 39: Example2

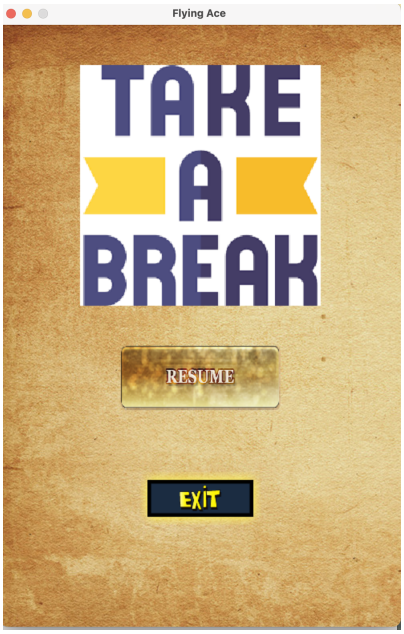


Figure 40: Example3

Figure 41: Example4

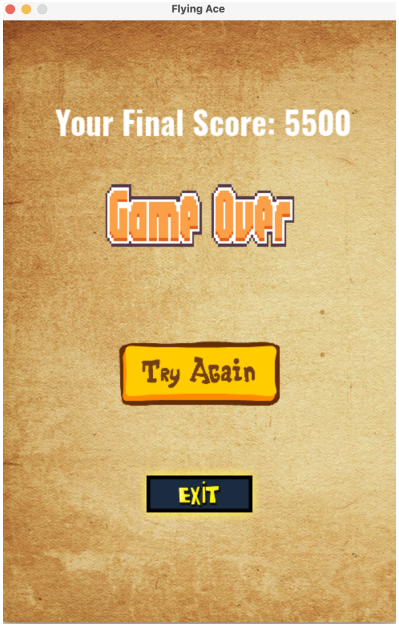


Figure 42: Example5

4.3. Alternative tools/technologies

Instead of using pygame to create this 2D aircraft battle, there are 2 alternative tools that can be used.

1. **Java**
2. **Unity (supported by C# coding)**

4.4. Comparative Analysis

When deciding whether to use Pygame, Java, or Unity for game development, it is essential to consider factors such as the scale of the project, the target platform, performance requirements.

For pygame:

Firstly, compare to other two language java and C#, pygame is ideal for beginners in game development, as Python's syntax is easy to learn and understand.

Secondly, pygame has a huge library and many useful built-in functions, so it is particularly useful when you are developing simple 2D games, prototypes, or educational game in a short amount of time.

For java:

Java is generally faster and more efficient than Python because it is a compiled language[10], so when creating some complex 2D games which need to deal with extensive resources, java can process information faster and make your game performance better.

For Unity:

Firstly, most of commercial games such as mobile game and online game are developed by Unity engine, so it has became one of the most favorite game tool. If you want to design a complex 3D game or highly polished 2D games with complex mechanics, Unity engine is the best choice.[11]

Secondly, Unity has advantages on built-in physics, animation, and real physical engine, so it will be very beneficial if you want to create a complex game in a game development company.

Bibliography

[1] Jon Fincher , “Pygame: A primer on game programming in python,” 2019, see https://realpython.com/pygame-a-primer/.

[2] Pygam docs, “pygame documentation,” see https://www.pygame.org/docs/.

[3] Tech with Tim, “Pygame in 90 minutes - for beginners,” 2021, see https://www.youtube.com/watch?v=jO6qQDNa2UY.

[4] Python, “Python documentation,” see https://docs.python.org/3/.

[5] Austin Rogers, “A comprehensive introduction to pygame,” 2016, see https://gamedevacademy.org/a-comprehensive-introduction-to-pygame/.

[6] Annoy, “Pygame pros and cons,” 2022, see "https://www.tutorialandexample.com/pygame-pros-and-cons".

[7] Matt Layman, “Teaching a kid to code with pygame zero,” see "https://www.tutorialandexample.com/pygame-pros-and-cons".

[8] 3D Graphics, “3d graphics with pygame,” 2011, see "https://www.petercollingridge.co.uk/tutorials/3d/pygame/".

[9] Coders Legacy, “Pygame sprite collision detection,” see "https://coderslegacy.com/python/pygame-sprite-collision-detection/".

[10] snapLogic, “Python vs. java performance,” see "https://www.snaplogic.com/glossary/python-vs-java-performance".

[11] Yan Telles, “The pros and cons of the unity game engine,” 2022, see "https://www.itechart.com/blog/unity-pros-and-cons/".