# INFO1111: Computing 1A Professionalism

## 2023 Semester 1

# Self-Learning Report

# Submission number: NO.1

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

| | |
|---|---|
| **Student name** | Dianlei Huang |
| **Student ID** | 520268838 |
| **Topic** | Pygame |
| **Levels already achieved** | None |
| **Levels in this report** | A and B |

# 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**

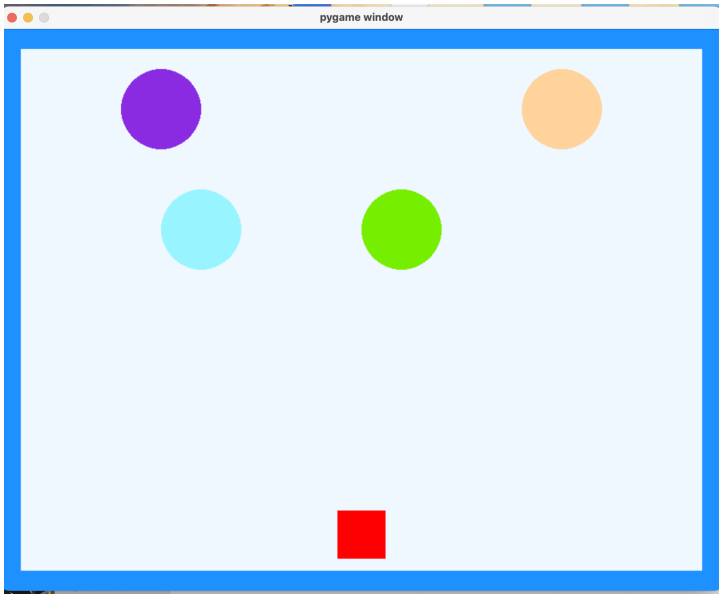


Figure 6: Example1

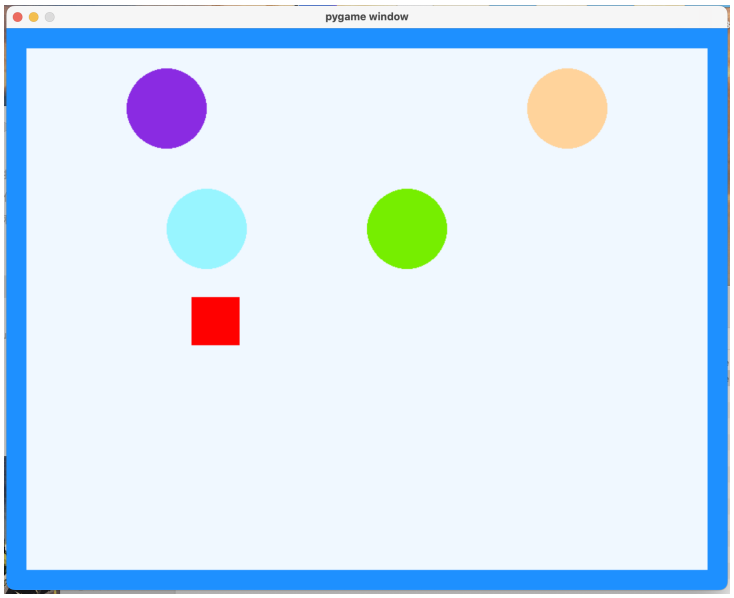


Figure 7: Example2

# 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.

## 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. 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. 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. 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, 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. 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))**. 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)**. 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()** 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 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()** . 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()**. 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. 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(). 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. 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()**.

The whole program finished all the requirements in level B, and the codes can be found in github link. The example picture is given below

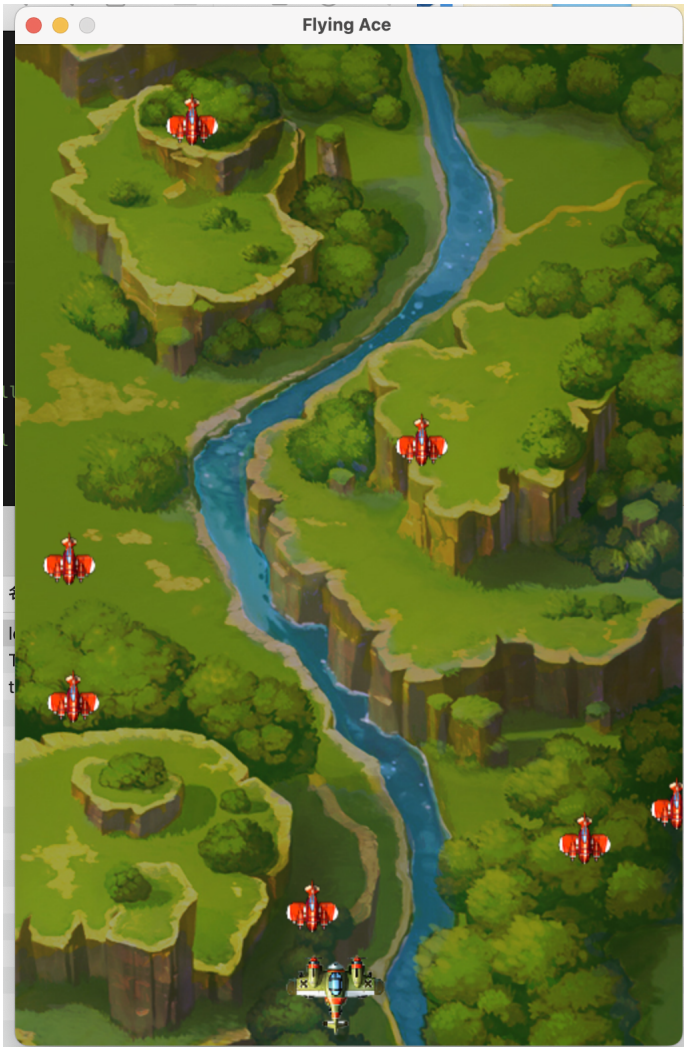


Figure 14: Example1

# 3. Level C: Deeper Understanding

Level C focuses on showing that you have actually understood the tool or technology at a relatively advanced level. You will need to compare it to alternatives, identifying key strengths and weaknesses, and the areas where this tool is most effective.

## 3.1. Strengths

What are the key strengths of the item you have learnt? (50-100 words)

## 3.2. Weaknesses

What are the key weaknesses of the item you have learnt? (50-100 words)

## 3.3. Usefulness

Describe one scenario under which you believe the topic you have learnt could be useful. (50-100 words)

## 3.4. Key Question 1

Note: This question is in the table in the 'Self Learning: List of Topics' page on Canvas. (50-100 words)

## 3.5. Key Question 2

Note: This question is in the table in the 'Self Learning: List of Topics' page on Canvas. (50-100 words)

# 4. Level D: Evolution of skills

## 4.1. Level D Demonstration

This is a short description of the application that you have developed. (50-100 words). **IMPORTANT:** *You might wish to submit this as part of an earlier submission in order to obtain feedback as to whether this is likely to be acceptable for level D.*

## 4.2. Application artifacts

Include here a description of what you actually created (what does it do? How does it work? How did you create it?). Include any code or other related artefacts that you created (these should also be included in your github repository).

If you do include screengrabs to show what you have done then these should be annotated to explain what it is showing and what the application does.

## 4.3. Alternative tools/technologies

Identify 2 alternative tools/technologies that can be used instead of the one you studied for your topic. (e.g. if your topic was Python, then you might identify Java and Golang)

## 4.4. Comparative Analysis

Describe situations in which both your topic and each of the identified alternatives would be preferred over the others (100-200 words).