A
MINOR PROJECT REPORT
ON

“Earth Invader Game”

submitted in the partial fulfilment of the requirement for the award of degree of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE & ENGINEERING

SUBMITTED BY:
RISHI KUMAR THAKUR (A40318052)
Batch: 2018-2022
PROJECT GUIDE:
Mr Yatin Chopra (AP, CSE Dept)

Department of Computer Science & Engineering Faculty of Engineering & Technology
P.D.M. University, Bahadurgarh

DECLARATION BY THE CANDIDATE

I hereby declare that the work presented in this report entitled **"Earth Invader Game"**, in fulfilment of the requirement for the award of the degree Bachelor of Technology in Computer Science & Engineering, submitted in CSE Department, PDMU, Bahadurgarh, Haryana is an authentic record of my own work carried out during my degree under the guidance of

Mr Yatin Chopra.

The work reported in this has not been submitted by me for award of any other degree or diploma.

Date: 15th April 2021
Place: New Delhi

Rishi Kumar Thakur
A40318052
B.Tech CSE 6th SEM.

Certificate of Completion

This is to certify that the Project work entitled “Earth Invader Game” submitted by Rishi Kumar Thakur in fulfilment for the requirements of the award of Bachelor of Technology Degree in Computer Science & Engineering at PDMU, Bahadurgarh, Haryana is an authentic work carried out by him under my supervision and guidance. To the best of my knowledge, the matter embodied in the project has not been submitted to any other University / Institute for the award of any Degree.

Mr Yatin Chopra
A.P. in CSE Department
Faculty of Engineering & Technology

ACKNOWLEDGEMENT

I express my sincere gratitude to Mr. Ajay Dureja (Head, Department of CSE) and Mr Yatin Chopra (AP), for his valuable guidance and timely suggestions during the entire duration of my dissertation work, without which this work would not have been possible. I would also like to convey my deep regards to all other faculty members who have bestowed their great effort and guidance at appropriate times without which it would have been very difficult on my part to finish this work. Finally, I would also like to thank my friends for their advice and pointing out my mistakes.

Rishi Kumar Thakur
A40318052
B.Tech CSE 6th SEM

TABLE OF CONTENTS

1. Img- Image
2. Px- Pixels
3. RGB- Red Green Blue
4. Min- Minimum
5. Max- Maximum
6. Attr- Attributes
7. Var- Variable
8. Obj- Object

CHAPTER 1
INTRODUCTION

1.1 BACKGROUND OF THE PROJECT

This project was made for the purpose of implementing the code in the most efficient way possible to create a game which you can play in your free time.

This Project was started in Mid-February this year and is a result of the work done in almost six weeks. This game was slightly tweaked compared to your classic space invader type game to make it much tough as possible.

1.2 AIMS AND OBJECTIVE

The main aim of this project is to use Python and its modules efficiently to create a working game which can give a newbie an idea about the working of code, modules, terminals, implementation, and other important factors of programming.

My main aim was to explain the code and it's working as efficiently as possible so a person who is new to programming can learn to create a game from scratch and to spark his/her interest in Python as well as programming.

CHAPTER 2

CODING AND IT'S SIGNIFICANCE

```
import math
import random
```

```
import pygame
from pygame import mixer

# Intialize the pygame
pygame.init()

# create the screen
screen = pygame.display.set_mode((800, 600))

# Background
background = pygame.image.load("background.jpg")

# Sound
mixer.music.load("background.wav")
mixer.music.play(-1)

# Caption and Icon
pygame.display.set_caption("Earth Invader")
icon = pygame.image.load("ufo.png")
pygame.display.set_icon(icon)
```

```
# Player
playerImg = pygame.image.load("player.png")
playerX = 370
playerY = 480
playerX_change = 0

# Enemy
enemyImg = []
enemyX = []
enemyY = []
enemyX_change = []
enemyY_change = []
```

```
num_of_enemies = 6

for i in range(num_of_enemies):
    enemyImg.append(pygame.image.load("enemy.png"))
    enemyX.append(random.randint(0, 736))
    enemyY.append(random.randint(50, 150))
    enemyX_change.append(3)
    enemyY_change.append(4)

# Bullet

# Ready - You can't see the bullet on the screen
# Fire - The bullet is currently moving

bulletImg = pygame.image.load("bullet.png")
bulletX = 0
bulletY = 480
bulletX_change = 0
bulletY_change = 10
bullet_state = "ready"

# Score

score_value = 0
font = pygame.font.Font("freesansbold.ttf", 32)

textX = 10
testY = 10

# Game Over
over_font = pygame.font.Font("freesansbold.ttf", 64)



def show_score(x, y):
    score = font.render("Score : " + str(score_value), True, (255, 255,
255))
    screen.blit(score, (x, y))


def game_over_text():
    over_text = over_font.render("GAME OVER", True, (255, 255, 255))
    screen.blit(over_text, (200, 250))
```

```
def player(x, y):
    screen.blit(playerImg, (x, y))


def enemy(x, y, i):
    screen.blit(enemyImg[i], (x, y))


def fire_bullet(x, y):
    global bullet_state
    bullet_state = "fire"
    screen.blit(bulletImg, (x + 16, y + 10))


def isCollision(enemyX, enemyY, bulletX, bulletY):
    distance = math.sqrt(math.pow(enemyX - bulletX, 2) + (math.pow(enemyY -
bulletY, 2)))
    if distance < 27:
        return True
    else:
        return False


# Game Loop
running = True
while running:

    # RGB = Red, Green, Blue
    screen.fill((0, 0, 0))
    # Background Image
    screen.blit(background, (0, 0))
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

        # if keystroke is pressed check whether its right or left
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_LEFT:
                playerX_change = -5
            if event.key == pygame.K_RIGHT:
                playerX_change = 5
            if event.key == pygame.K_SPACE:
```

```
                if bullet_state is "ready":
                    bulletSound = mixer.Sound("laser.wav")
                    bulletSound.play()
                    # Get the current x cordinate of the spaceship
                    bulletX = playerX
                    fire_bullet(bulletX, bulletY)

        if event.type == pygame.KEYUP:
            if event.key == pygame.K_LEFT or event.key == pygame.K_RIGHT:
                playerX_change = 0

    # 5 = 5 + -0.1 -> 5 = 5 - 0.1
    # 5 = 5 + 0.1

    playerX += playerX_change
    if playerX <= 0:
        playerX = 0
    elif playerX >= 736:
        playerX = 736

    # Enemy Movement
    for i in range(num_of_enemies):

        # Game Over
        if enemyY[i] > 440:
            for j in range(num_of_enemies):
                enemyY[j] = 2000
            game_over_text()
            break

        enemyX[i] += enemyX_change[i]
        if enemyX[i] <= 0:
            enemyX_change[i] = 2
            enemyY[i] += enemyY_change[i]
        elif enemyX[i] >= 736:
            enemyX_change[i] = -2
            enemyY[i] += enemyY_change[i]

        # Collision
        collision = isCollision(enemyX[i], enemyY[i], bulletX, bulletY)
        if collision:
            explosionSound = mixer.Sound("explosion.wav")
```

```
                explosionSound.play()
                bulletY = 480
                bullet_state = "ready"
                score_value += 1
                enemyX[i] = random.randint(0, 736)
                enemyY[i] = random.randint(50, 150)

            enemy(enemyX[i], enemyY[i], i)

        # Bullet Movement
        if bulletY <= 0:
            bulletY = 480
            bullet_state = "ready"

        if bullet_state is "fire":
            fire_bullet(bulletX, bulletY)
            bulletY -= bulletY_change

        player(playerX, playerY)
        show_score(textX, testY)
        pygame.display.update()
```

2.1 Code Significance

What is Import and Why we use It

```
1   import pygame
2   import math
3   import random
```

Set of Python modules designed for writing video games.

IMPORT USES

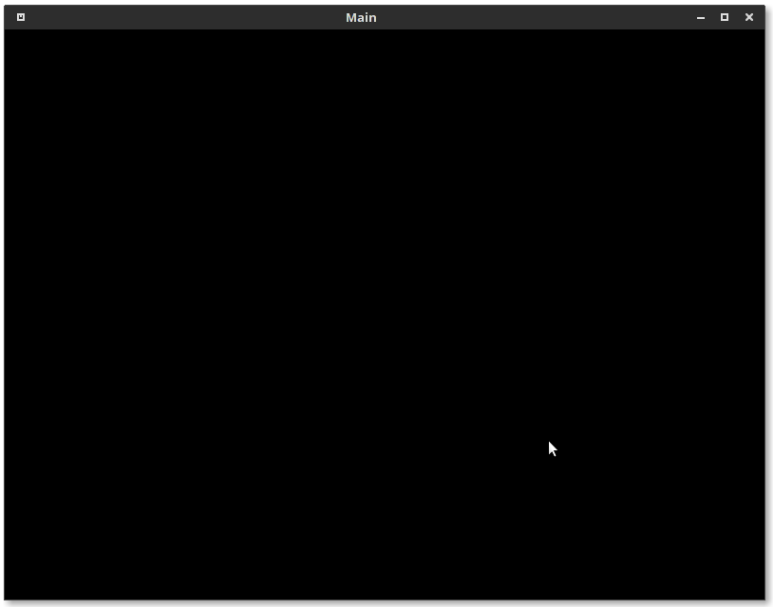
Importing refers to allowing a Python file or a Python module to access the script from another Python file or module. You can only use functions and properties your program can access. For instance, if you want to use mathematical functionalities, you must import the math package first.

The __init__ method is similar to constructors in C++ and Java . Constructors are used to initialize the object's state.

```
5   # INtializing pygame
6   pygame.init()
```

Creating Screen

```
8   #  Creating screen
9   screen = pygame.display.set_mode((800, 600))
```



THIS WILL CREATE A BLANK SCREEN FOR OUR PYGAME WINDOW AND THE PARAMTER WILL CONTAIN HEIGHT AND WIDTH INFO

GAMELOOP

```
102   # Game Loop
103   running = True
104   while running:
```

Anything happening inside the game and will end the loop whenever the QUIT event will take place.

```
110       for event in pygame.event.get():
111           if event.type == pygame.QUIT:
112               running = False
```

Changing the Title and Icon of our window

```
20  # Caption and Icon
21  pygame.display.set_caption("Earth Invader")
22  icon = pygame.image.load('ufo.png')
23  pygame.display.set_icon(icon)
```

Filling the Screen with color

```
106     # RGB = Red, Green, Blue
107     screen.fill((0, 0, 0))
```

RGB describes a colour as a tuple of three components. Each component can take a value between 0 and 255, where the tuple (0, 0, 0) represents black and (255, 255, 255) represents white.

Here are a few more examples of colors in RGB:

Color	RGB value
Red	255, 0, 0
Orange	255, 128, 0
Pink	255, 153, 255

Defining our player and Displaying it onto screen

```
25  # Player
26  playerImg = pygame.image.load('player.png')
27  playerX = 370
28  playerY = 480
```

Our player will be loaded with

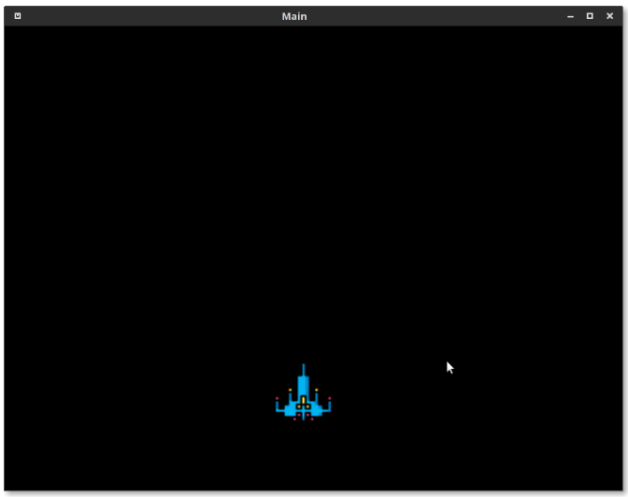
playerX = X-axis

playerY = Y-axis

```
80  def player(x, y):
81      screen.blit(playerImg, (x, y))
```

blit() accepts either a Surface or a string as its image parameter. If image is a str then the named image will be loaded from the images/ directory.

Our current window after compiling



Movement Mechanic

```
181        player(playerX, playerY)
```

To change the location of the image which in this case is our player

```
114        # if keystroke is pressed check whether its right or left
115        if event.type == pygame.KEYDOWN:
116            if event.key == pygame.K_LEFT:
117                playerX_change = -5
118            if event.key == pygame.K_RIGHT:
119                playerX_change = 5
```

We can now move our player Right and Left with the Keystroke of Arrow button onto Keyboard.

Keydown will record the key pressed

```
132        # 5 = 5 + -0.1 -> 5 = 5 - 0.1
133        # 5 = 5 + 0.1
134
135        playerX += playerX_change
```

Adding Boundaries for Our Game

We have to take in consideration of our player size also which in this case is 64 pixel

By using If condition, We are going to make sure that If our player touches the End-Coordinates, It will re-appear within the boundaries

```
136         if playerX <= 0:
137             playerX = 0
138         elif playerX >= 736:
139             playerX = 736
```

Our current window after compiling

Creating The Enemy

Just like Player, We will Import Enemy

```
40      enemyImg.append(pygame.image.load('enemy.png'))
41      enemyX = 370
42      enemyY = 480
43      enemyX_change = 0
44      enemyY_change = 0
```

```
84  def enemy(x, y, i):
85      screen.blit(enemyImg[i], (x, y))
```

Why Import Random

```
enemyImg.append(pygame.image.load('enemy.png'))
enemyX.append(random.randint(0, 736))
enemyY.append(random.randint(50, 150))
```

Our Enemy will be Re-spawning now within the Range of our game

IMPORT RANDOM

The random module is a built-in module to generate the pseudo-random variables.

The random.randint() method returns a random integer between the specified integers.

Movement Mechanic for Our Enemy

We want that enemy should move from

left to right i.e X-axis and then move from up to down i.e Y-axis

```
151             enemyX[i] += enemyX_change[i]
152             if enemyX[i] <= 0:
153                 enemyX_change[i] = 2
154                 enemyY[i] += enemyY_change[i]
155             elif enemyX[i] >= 736:
156                 enemyX_change[i] = -2
157                 enemyY[i] += enemyY_change[i]
```

Movement from left to Right = X-axis

Movement from Up to Down = Y-axis

Enemy will move down only after hitting both the boundaries of our window

Bullets And It’s Mechanic

STUFF TO CONSIDER

1. It has to be shot from top of our spaceship.
2. It’s position should decrease from our player position to 0 eventually.
3. It should keep moving in the same direction even if our spaceship is moving.

Loading the Bullet

```
48  # Ready - You can't see the bullet on the screen
49  # Fire - The bullet is currently moving
50
51  bulletImg = pygame.image.load('bullet.png')
```

Defining the Bullet

```
88  def fire_bullet(x, y):
89      global bullet_state
90      bullet_state = "fire"
91      screen.blit(bulletImg, (x + 16, y + 10))
```

Firing the bullet

```
120             if event.key == pygame.K_SPACE:
121                 if bullet_state is "ready":
122                     # Get the current x coordinate of the spaceship
123                     bulletX = playerX
124                     fire_bullet(bulletX, bulletY)
```

```
177         if bullet_state is "fire":
178             fire_bullet(bulletX, bulletY)
179             bulletY -= bulletY_change
```

Currently Our ammo will move along with our spaceship

Modifying the Bullet Mechanic

```
173         if bulletY <= 0:
174             bulletY = 480
175             bullet_state = "ready"
```

To make our Bullet unlimited

```
120             if event.key == pygame.K_SPACE:
121                 if bullet_state is "ready":
122                     bulletSound = mixer.Sound("laser.wav")
123                     bulletSound.play()
124                     # Get the current x cordinate of the spaceship
125                     bulletX = playerX
126                     fire_bullet(bulletX, bulletY)
```

It won’t move along with our player now

It also won’t appear again even after pressing space

Collision Detection

To collide our player with the Enemy

Importing Math for Mathematical Function

```python
import math
```

Collision Formula

The distance between the points (x_1, y_1) and (x_2, y_2) is given by the following formula:

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

```python
def isCollision(enemyX, enemyY, bulletX, bulletY):
    distance = math.sqrt(math.pow(enemyX - bulletX, 2) + (math.pow(enemyY - bulletY, 2)))
    if distance < 27:
        return True
    else:
        return False
```

```python
        # Collision
        collision = isCollision(enemyX[i], enemyY[i], bulletX, bulletY)
        if collision:
            explosionSound = mixer.Sound("explosion.wav")
            explosionSound.play()
            bulletY = 480
            bullet_state = "ready"
            score_value += 1
            enemyX[i] = random.randint(0, 736)
            enemyY[i] = random.randint(50, 150)
```

Some things to consider here are

1) Score will increase by one when we hit our enemy
2) Our enemy will re-spawn in random position

Creating Multiple Enemies

```
31  # Enemy
32  enemyImg = []
33  enemyX = []
34  enemyY = []
35  enemyX_change = []
36  enemyY_change = []
37  num_of_enemies = 6
38
39  for i in range(num_of_enemies):
40      enemyImg.append(pygame.image.load('enemy.png'))
41      enemyX.append(random.randint(0, 736))
42      enemyY.append(random.randint(50, 150))
43      enemyX_change.append(3)
44      enemyY_change.append(4)
```

Collision will be now shifted inside the while loop to make it running

We have to add i in range(no_of_enemy) i.e

enemyX[i] += enemyX_change[i]

Enemy now (6 in this case)

Adding Text and Displaying Score

```
60    score_value = 0
61    font = pygame.font.Font('freesansbold.ttf', 32)
62
63    textX = 10
64    testY = 10
```

```
70  def show_score(x, y):
71      score = font.render("Score : " + str(score_value), True, (255, 255, 255))
72      screen.blit(score, (x, y))
```

With Reference to Collision part we know the score will increase by one when collision takes place

str(score_value) is Type Casting from Int to String

(255,255,255) for white display, Refer to RGB section for more info

Score Section

Adding BGM/Sound

```
5   from pygame import mixer
```

```
16  # Sound
17  mixer.music.load("background.wav")
18  mixer.music.play(-1)
```

To Handle BGM throughout the Game

```
121                 if bullet_state is "ready":
122                     bulletSound = mixer.Sound("laser.wav")
123                     bulletSound.play()
```

To play the Sound when the bullet state is ‘Ready’

```
161             if collision:
162                 explosionSound = mixer.Sound("explosion.wav")
163                 explosionSound.play()
```

To play the Sound when the collision takes place

Game Over

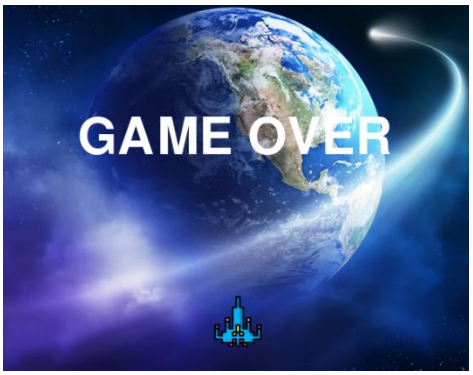
Game Should End when the Enemy hit our player

```
144             # Game Over
145             if enemyY[i] > 440:
146                 for j in range(num_of_enemies):
147                     enemyY[j] = 2000
148                 game_over_text()
149                 break
```

Moving All the Enemy out of our game window when the Game gets Over

```
66  # Game Over
67  over_font = pygame.font.Font('freesansbold.ttf', 64)
```

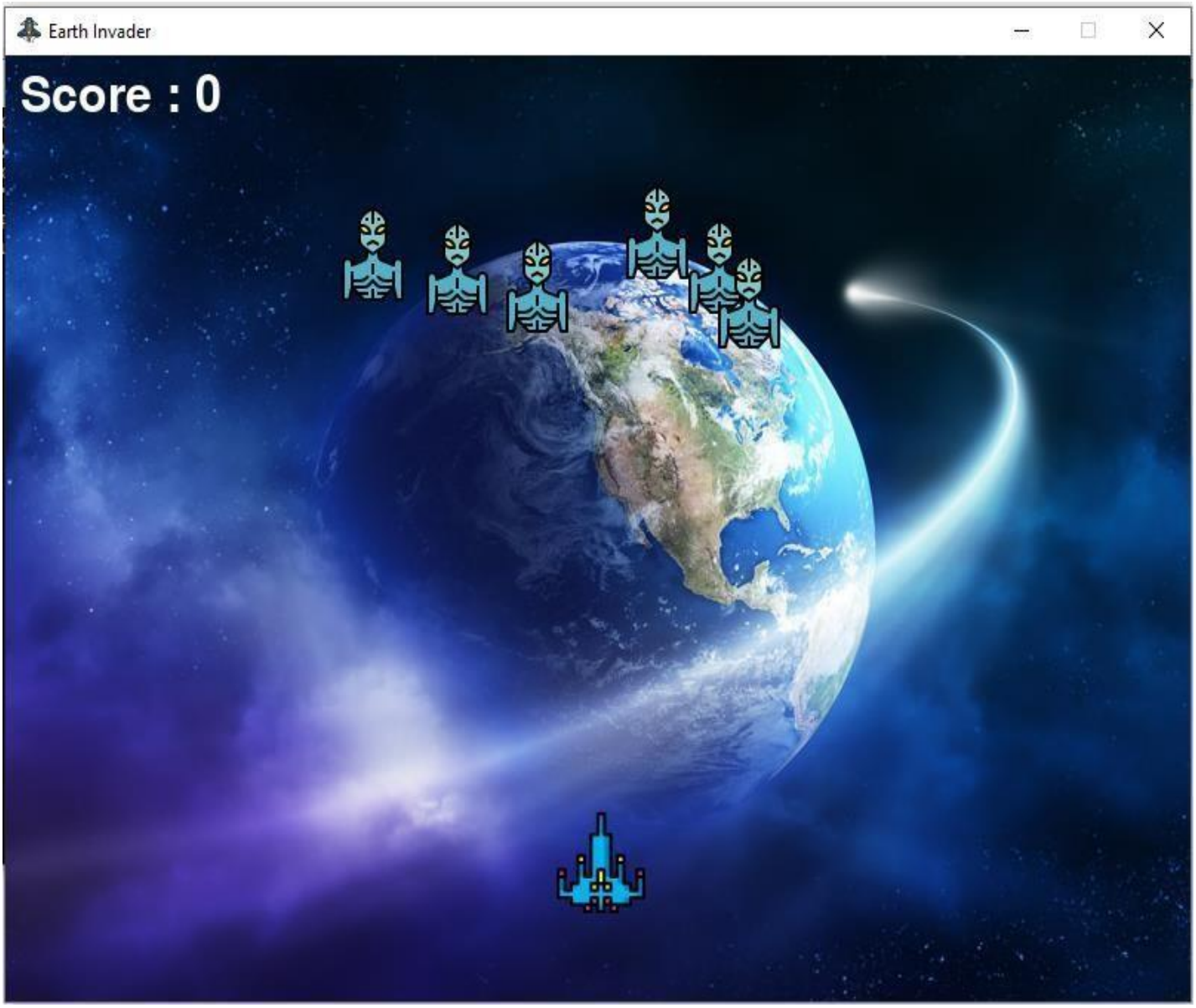
```
75  def game_over_text():
76      over_text = over_font.render("GAME OVER", True, (255, 255, 255))
77      screen.blit(over_text, (200, 250))
```



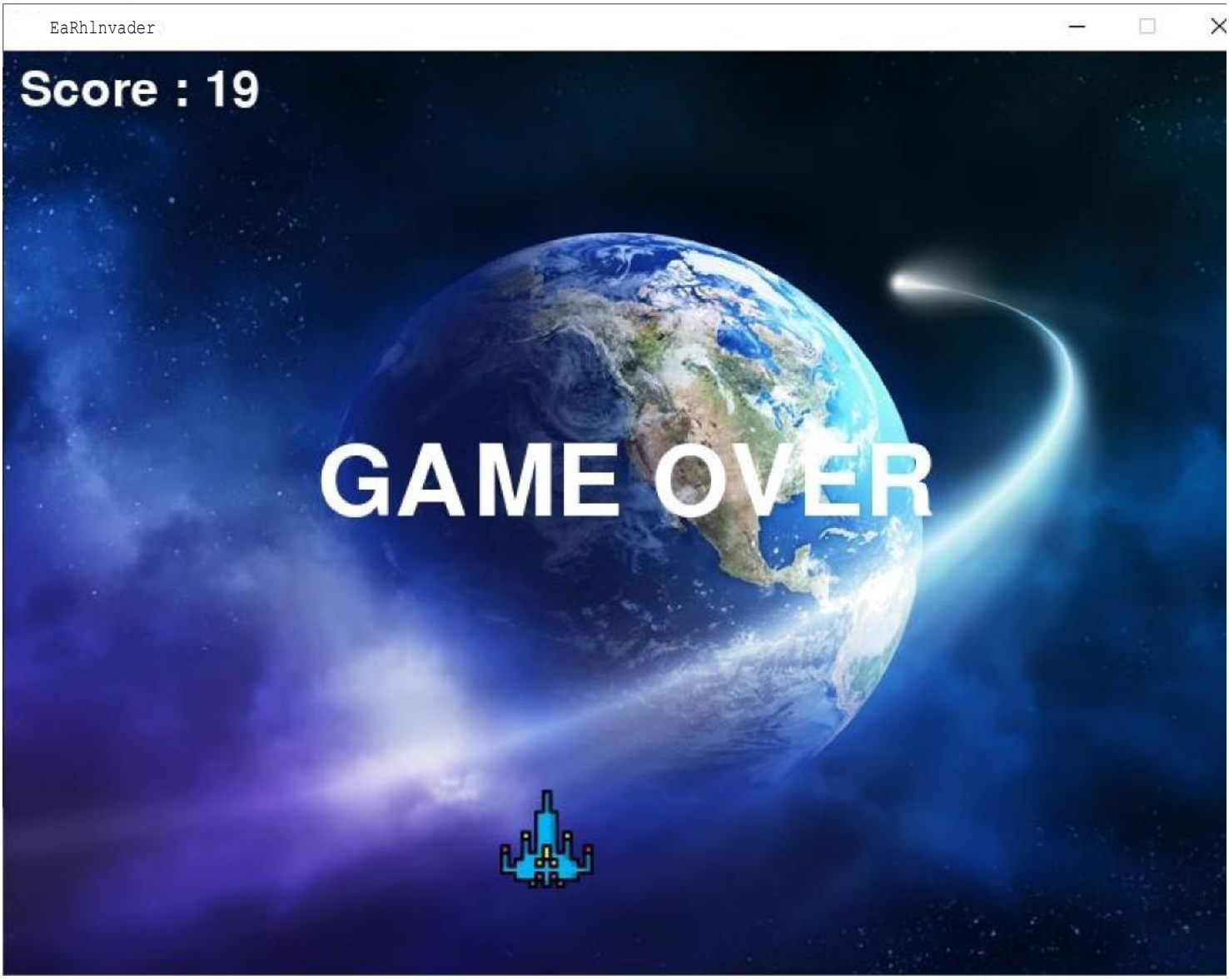
CHAPTER 3

TESTING THE CODE

3.1 TESTING AND RESULT



Earth Invader X


EaRhlnvader
Score : 19
GAME OVER

CHAPTER 5

DISCUSSION

4.1 LIMITATIONS

Some limitations of this project are: -

- The first and foremost limitation of this game is that It is very basic game but good enough for a complete beginner.
- The game runs on Python terminal so anyone who wants to play the game needs to have Python installed on their system.
- The main aim of the project was to help a newbie learn how to build a game from scratch but the person needs to be at least skilled enough to redirect the main file into an editor terminal to learn about code used.

4.2 CONCLUSION

This project is successfully implemented with the help of Python and some of it's modules. The project will give a beginner a summarised idea of working of a code and how each line of code works to create a functioning game.

4.3 FUTURE SCOPE

We can add new features as and when we require. Reusability is possible as and when required in this game. There is flexibility in all the codes.

This game also has the scope of enhancements like: -

- Addition of target limit along with score panel to stop the loop.
- The target can also help to create levels so the players can face much harder enemy in the next level.
- As the game was build for a beginner, several new features can be thought and implemented by them while they build the project.

4.4 BIBLIOGRAPHY

1. **https://www.freecodecamp.org/**
2. **https://www.youtube.com/channel/UCirPbvoHzD78Lnyll6YYUpg**
3. **https://www.pygame.org/**
4. **https://www.flaticon.com/**