A

Mini Project Report

On

Space Invader Game Development Using Python pygame.

Submitted in partial fulfillment of the requirements
of Institute of Distance and Open Learning(IDOL),
University of Mumbai for the degree of
Master of Computer Application
By

Mr. Sujit Prakash Tadadikar 225794

Mr. Pankaj Mahadeo Sakpal 227091

Faculty In charge: Prof. Seema Vishwakarma

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Spacewar! is a space combat video game developed in 1962 by Steve Russell, in collaboration with Martin Graetz and Wayne Wiitanen, and programmed by Russell with assistance from others including Bob Saunders and Steve Piner. It was written for the newly installed DEC PDP-1 at the Massachusetts Institute of Technology. After its initial creation, Spacewar was expanded further by other students and employees of universities in the area, including Dan Edwards and Peter Samson.

It was also spread to many of the few dozen, primarily academic, installations of the PDP-1 computer, making Spacewar the first known video game to be played at multiple computer installations.

The game features two spaceships, "the needle" and "the wedge", engaged in a dogfight while maneuvering in the gravity well of a star. Both ships are controlled by human players. Each ship has limited fuel for maneuvering and a limited number of torpedoes, and the ships follow Newtonian physics, remaining in motion even when the player is not accelerating.

Flying near the star to provide a gravity assist was a common tactic. Ships are destroyed when hit with a torpedo or colliding with the star. At any time, the player can engage a hyperspace feature to move to a new, random location on the screen, though each use has an increasing chance of destroying the ship instead. The game was initially controlled with switches on the PDP-1, though Alan Kotok and Bob Saunders built an early gamepad to reduce the difficulty and awkwardness of controlling the game.

Keyword: python , pygame ,gameloop.

INTRODUCTION

1.1 INTRODUCTION

Space Invaders is a classic arcade video game created by Tomohiro Nishikado in 1978, and was one of the early games that played a major role in popularizing the gaming industry towards mainstream media.

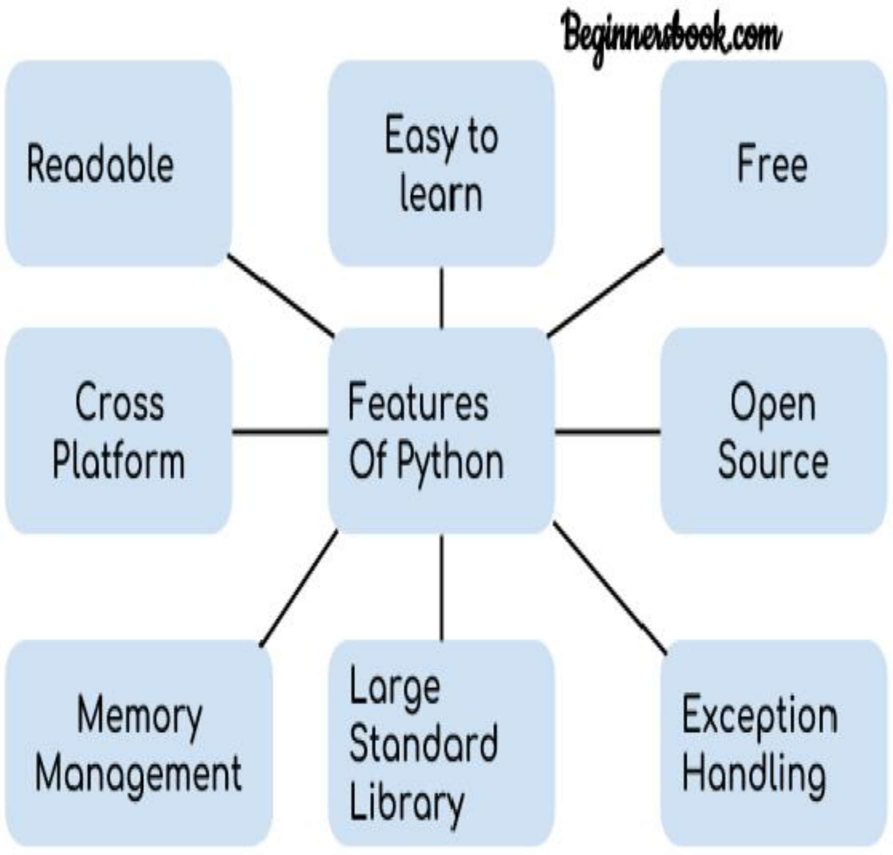
In the game, the **player is a fixed shoote**r, who controls a spaceship by moving it horizontally across the bottom of the screen and firing at descending aliens. The game also includes stationary green bunkers, that partially protect the player. These bunkers can be destroyed either by player or alien missiles. Many versions of the game also feature the aliens moving horizontally back and forth, while they advance to the bottom of the screen.

The player **earns points** by shooting it with the missiles, and the game ends when either alien reaches the bottom, or three lives are lost.

1.1.1 Python Programming language

Python is one of the many open source object oriented programming application software available in the market . Python is developed by **Guido van Rossum**. Guido van Rossum started implementing Python in 1989. Python is a very simple programming language so even if you are new to programming, you can learn python without facing any issues. Some of the many uses of Python are application development, implementation of automation testing process, allows multiple programming build, fully constructed programming library, can be used in all the major operating systems and platforms, database system accessibility, simple and readable code, easy to apply on complex software development processes, aids in test driven software application development approach, machine learning/ data analytics, helps pattern recognitions, supported in multiple tools, permitted by many of the provisioned frameworks, etc.

Some features of Python are-



1.2 Problem Definition

The game features two spaceships, "the needle" and "the wedge", engaged in a dogfight while maneuvering in the gravity well of a star. Both ships are controlled by human players. Each ship has limited fuel for maneuvering and a limited number of torpedoes, and the ships follow Newtonian physics, remaining in motion even when the player is not accelerating**.**

1.3 Objective of Game:

The objective of the game is simply to destroy a formation of advancing enemy spaceships. The player moves on to the next level upon destroying all enemy ships.

Each consecutive level gets tougher and tougher with enemies shooting at the player more frequently. The number of points obtained per enemy destroyed increases with every level. The player should aim to maximize his or her score.

1.4 Scope of Project:

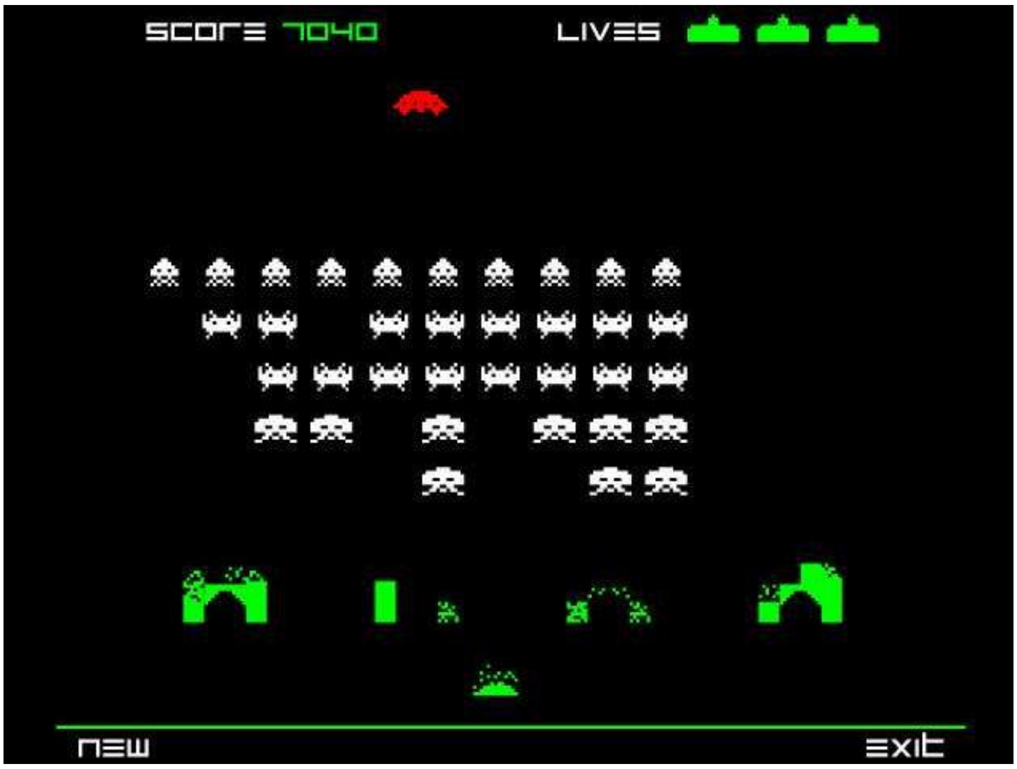
The game is based on the popular 2D arcade game, Space Invaders, developed in the 70's by Tomohiro Nishikado. There have been several implementations of different version of the game over time. However, there have been few 3D implementations of this game. The implementation discussed in this report aims to produce a 3D version of the game, with a novel user interface that brings more realism to the player.

Chapter 2

SYSTEM STUDY

2.1 Existing System

The versions of Space Invaders out there are mostly designed the way shown in Figure 1.



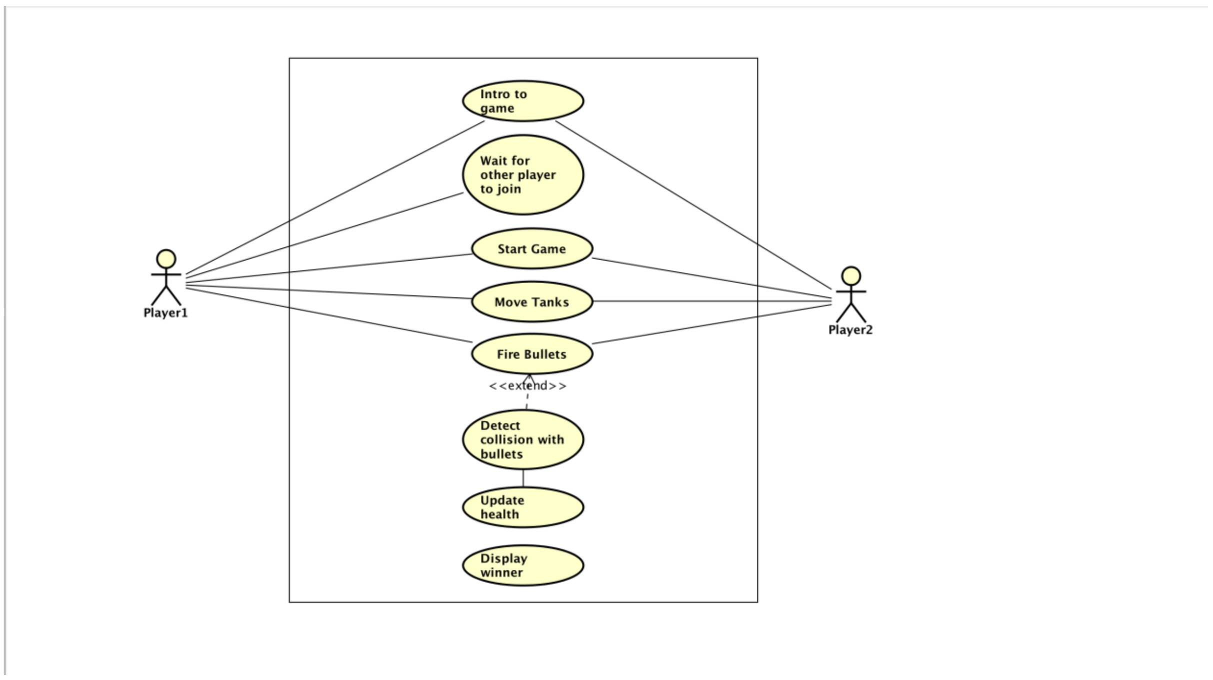
2.2 Disadvantages of Existing System:

The 2D Version Game has tiny spirits and doesn’t have audio support.

2.3 Proposed System:

- All the models are rendered in 3D space.
- The game play is from the perspective of the player, rather than from a 3rd person perspective.
- The game makes use of an infrared head-tracker and audio as input.
- We believe that changing the perspective and adding another dimension to the game would improve the realism of the game. By tracking the player’s real position adds anew level of involvement to the game play experience.

2.4 Use Case:


Intro to game
Wait for other player to join
Start Game
Move Tanks
Fire Bullets
<<extend>>
Detect collision with bullets
Update health
Display winner
Player1
Player2

Chapter 3

ANALYSIS & DESIGN

- **SOFTWARE AND HARDWARE REQUIREMENTS**

3.1.1Software Requirements

Operating System: Windows 10

Front End: Pycharm IDE

Module: pygame

3.1.2 Hardware Requirements

Minimum: Core 2 duo 2.4 GHz or Higher

RAM: 8 GB DDR4 and AboveMonitor: TFT

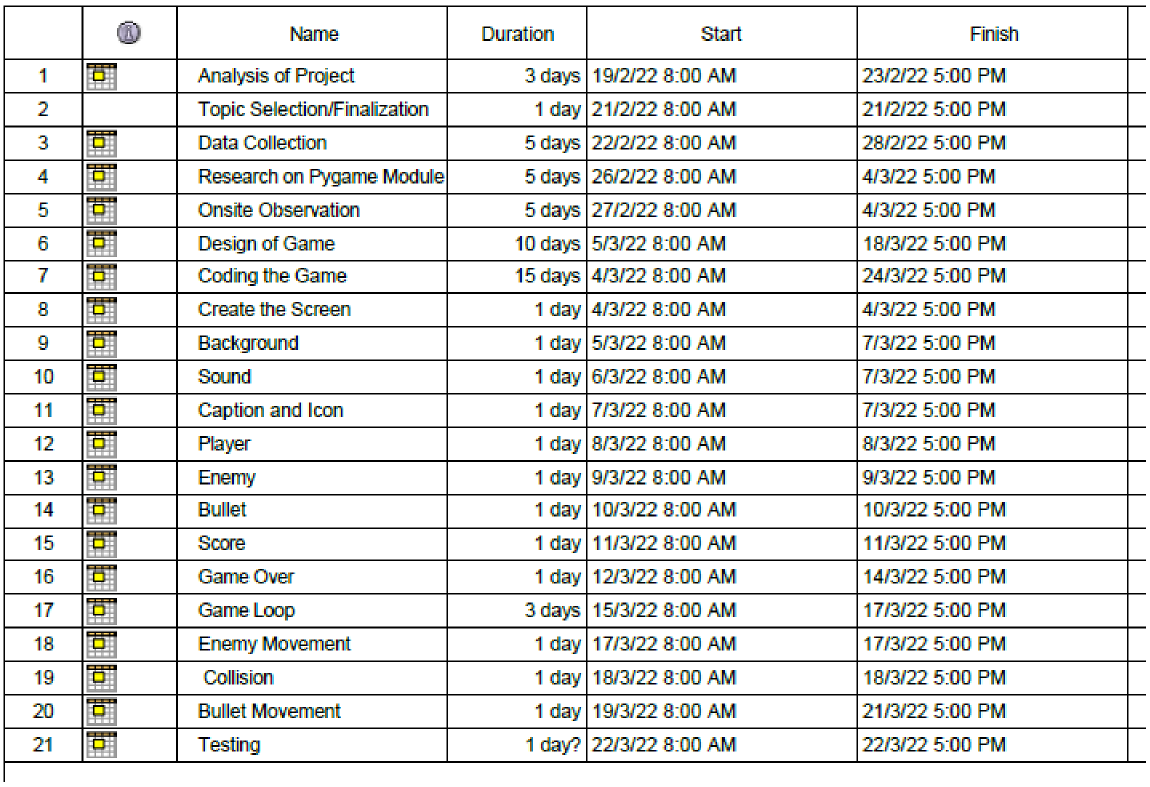
/ VGA / LCDHard Disk: 80GB and Above

Mouse: PS2 / USB Keyboard: PS2 /

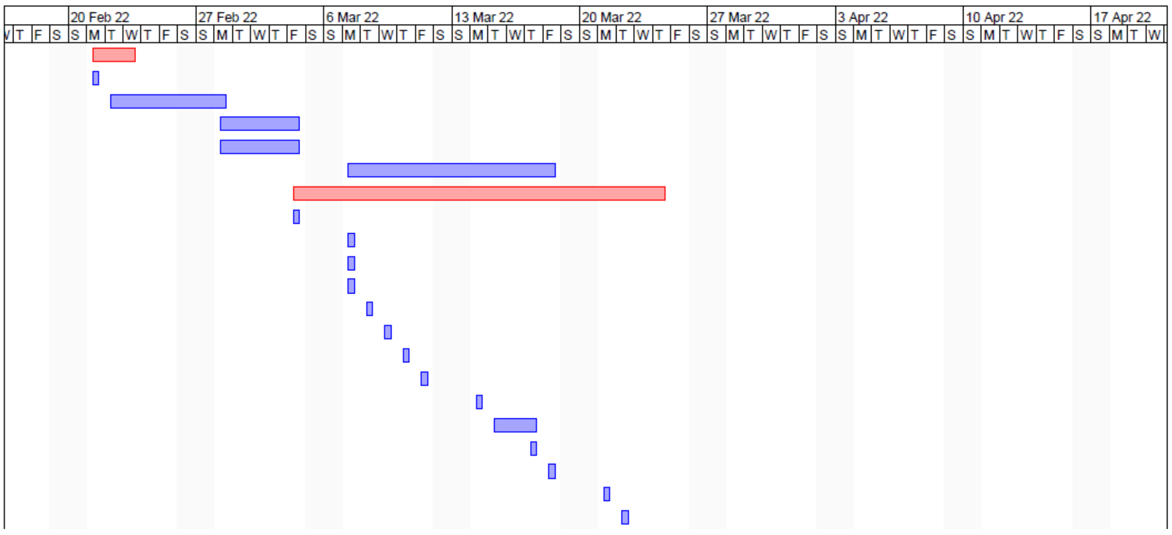
USB

GPU: 2GB AMD

3.2 GANTT CHART

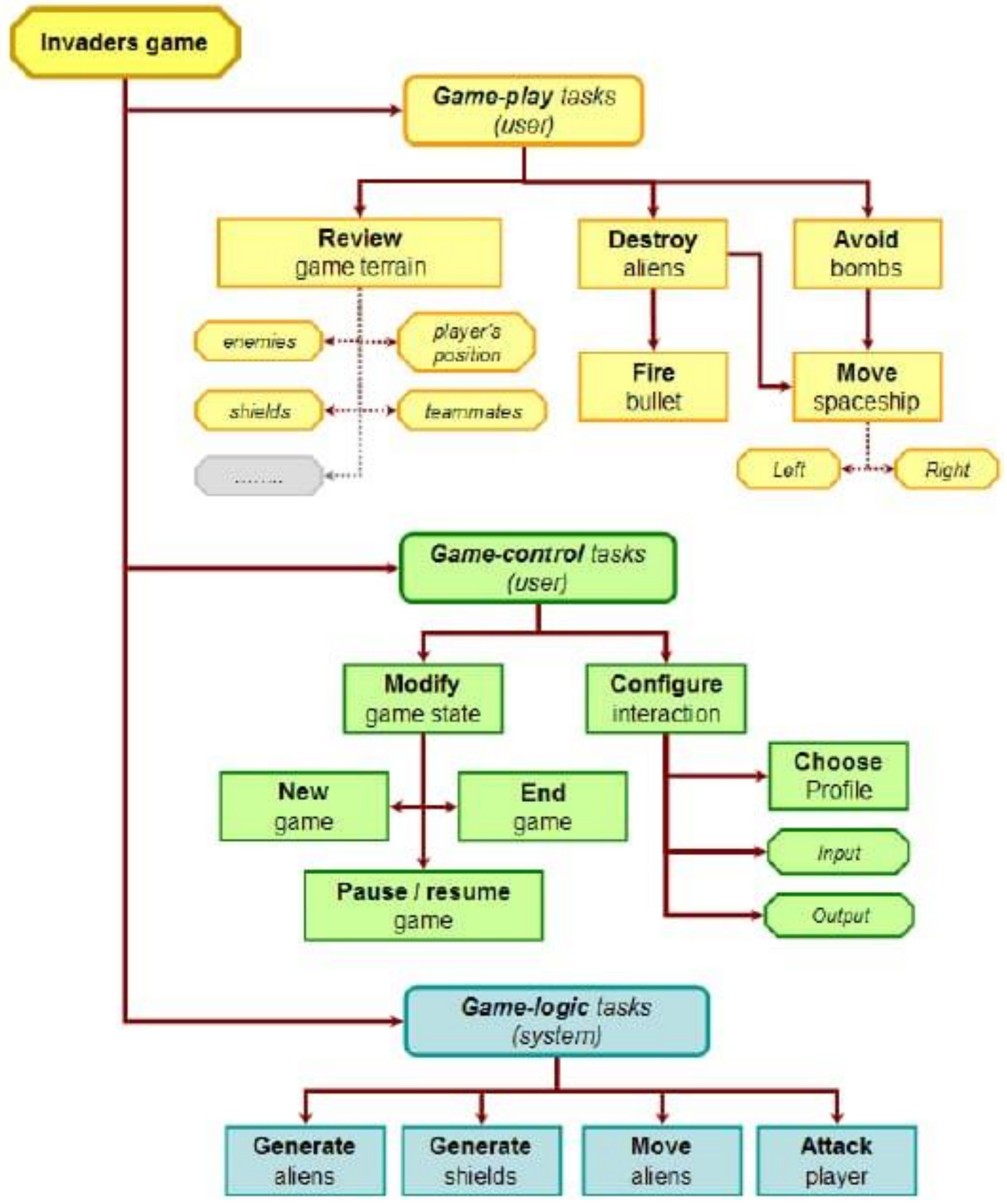


	ⓘ	Name	Duration	Start	Finish
1	📅	Analysis of Project	3 days	19/2/22 8:00 AM	23/2/22 5:00 PM
2		Topic Selection/Finalization	1 day	21/2/22 8:00 AM	21/2/22 5:00 PM
3	📅	Data Collection	5 days	22/2/22 8:00 AM	28/2/22 5:00 PM
4	📅	Research on Pygame Module	5 days	26/2/22 8:00 AM	4/3/22 5:00 PM
5	📅	Onsite Observation	5 days	27/2/22 8:00 AM	4/3/22 5:00 PM
6	📅	Design of Game	10 days	5/3/22 8:00 AM	18/3/22 5:00 PM
7	📅	Coding the Game	15 days	4/3/22 8:00 AM	24/3/22 5:00 PM
8	📅	Create the Screen	1 day	4/3/22 8:00 AM	4/3/22 5:00 PM
9	📅	Background	1 day	5/3/22 8:00 AM	7/3/22 5:00 PM
10	📅	Sound	1 day	6/3/22 8:00 AM	7/3/22 5:00 PM
11	📅	Caption and Icon	1 day	7/3/22 8:00 AM	7/3/22 5:00 PM
12	📅	Player	1 day	8/3/22 8:00 AM	8/3/22 5:00 PM
13	📅	Enemy	1 day	9/3/22 8:00 AM	9/3/22 5:00 PM
14	📅	Bullet	1 day	10/3/22 8:00 AM	10/3/22 5:00 PM
15	📅	Score	1 day	11/3/22 8:00 AM	11/3/22 5:00 PM
16	📅	Game Over	1 day	12/3/22 8:00 AM	14/3/22 5:00 PM
17	📅	Game Loop	3 days	15/3/22 8:00 AM	17/3/22 5:00 PM
18	📅	Enemy Movement	1 day	17/3/22 8:00 AM	17/3/22 5:00 PM
19	📅	Collision	1 day	18/3/22 8:00 AM	18/3/22 5:00 PM
20	📅	Bullet Movement	1 day	19/3/22 8:00 AM	21/3/22 5:00 PM
21	📅	Testing	1 day?	22/3/22 8:00 AM	22/3/22 5:00 PM

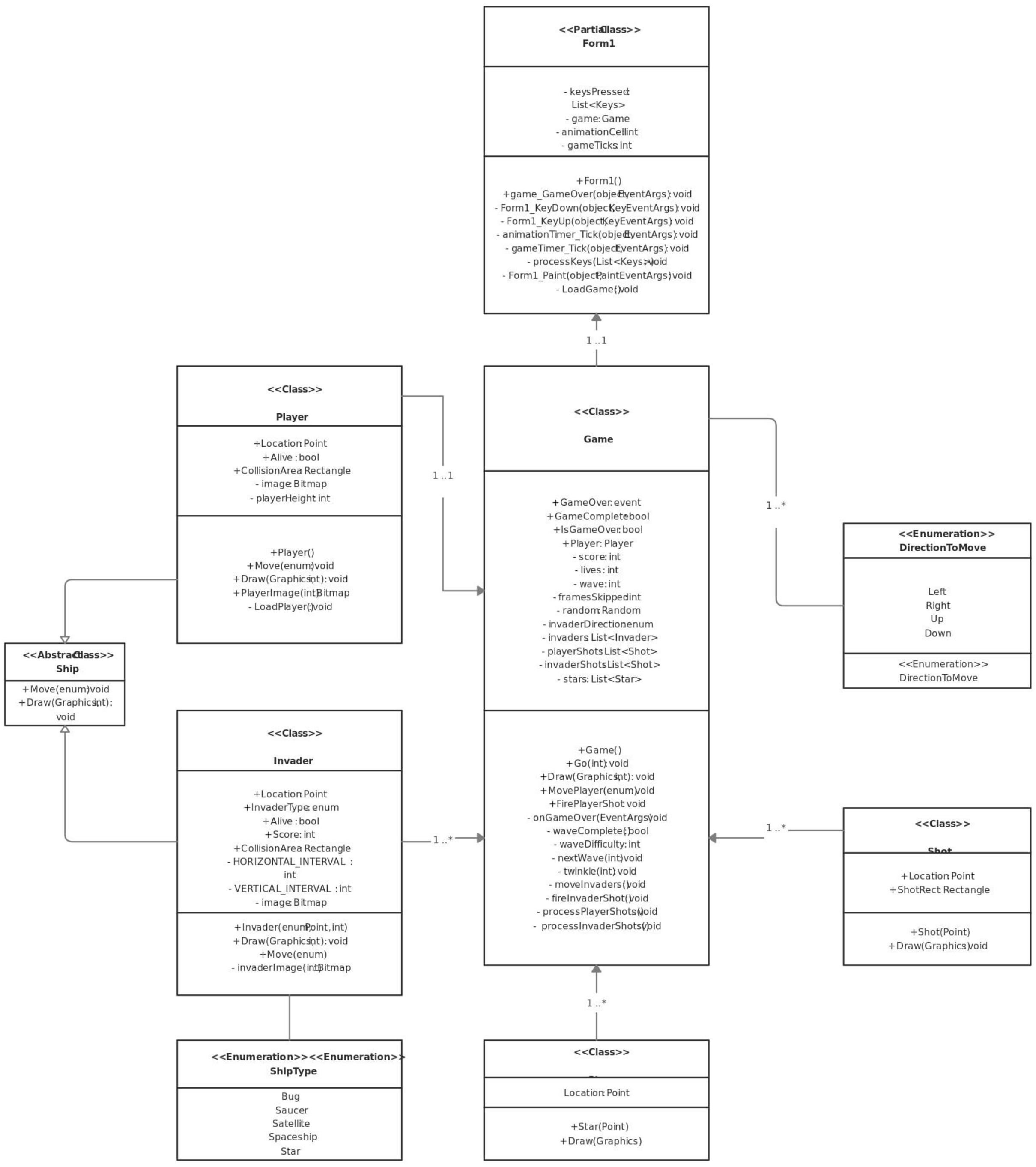

20 Feb 22
27 Feb 22
6 Mar 22
13 Mar 22
20 Mar 22
27 Mar 22
3 Apr 22
10 Apr 22
17 Apr 22

3.3 DIAGRAMS:

3.3.1 FLOWCHART:



3.3.2 UML DIAGRAM:



3.4 MODULE DESIGN AND ORGANIZATION:

❖ SOURCE CODE:

```
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.png')

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

# Caption and Icon
pygame.display.set_caption("Space 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(4)
    enemyY_change.append(40)

# 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] = 4
        enemyY[i] += enemyY_change[i]
    elif enemyX[i] >= 736:
        enemyX_change[i] = -4
        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()
```

Chapter 4

4.1 Explanation of key Functions:

Approach:

Import the required module.

Initialize the pygame.

Create three functions:

isCollision(): Which tells us whether the collision has occurred or not?

game_over(): Which returns True or False on the basis of which the code decided if the game has ended.

show_score(x, y): This shows the score on the screen

Create an infinite loop to execute the code continuously.

isCollision():

It’s very simple actually. Before explaining this, we want you to take a look at the collision portion of the code inside the game loop first below:

The value returned by the “isCollision()” function is being stored inside the “collision” variable. The value returned by the function is either True or False based on the criteria set for collision inside the function. Let’s take a look at what is inside the isCollision() function:

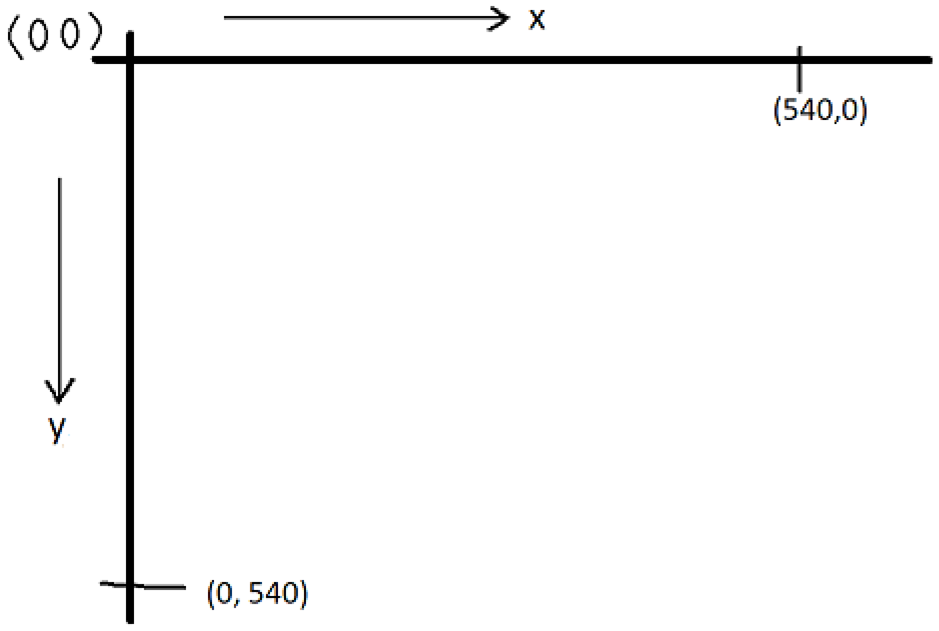
The criteria for collision set inside the function is the simplest thing as the distance between the bullet and the invader (our enemy). As you can see the formula used for calculating distance is something that every student study in their high school mathematics class. It’s the formula of the distance between two points having coordinates (x1, y1) and (x2, y2) which are being passed as parameters of the isCollision() function.

Here, we have set the criteria that if the value of the distance is less than or equal to 50, then it means collision has occurred. The value is chosen on the basis of the height and width of the png image used for the bullet and the invader. The value can be tweaked as per your own requirements by using the trial and error method.

So, whenever the position of the bullet and the invader changes then the isCollision() function checks if a collision has occurred or not. That is the reason why it is being called inside the game loop.

game_over():
Which returns True or False on the basis of which the code decided if the game has ended.
For understanding the game_over() function, let's take a look at the below snippet of code which is present inside the game loop:

Before getting into the explanation of code, it is recommended to know about the coordinate system followed in pygame. Take a look at the image below:



So, the criteria for game over is also collision. When the y-coordinate of the invader is greater than the spaceship i.e., 450 (y-coordinate of the spaceship), and the distance between the invader and the spaceship is less than 80 then a collision occurs and the game_over() function is called followed by the explosion sound.

The following lines will make it more clear:

The above is checked for all the invaders present in the game which is ensured by a for-loop at the starting.

show_score(x, y):
It shows the score on the screen.

The only thing show_score() function is doing is showing the score on the screen in a proper font selected by the user.

Every time a collision between the bullet and the invaders is happening a variable "score_val" is being incremented. This variable is then being displayed on the screen by the show_score() function as can be seen in the above code snippet.

4.2 Method of Implementation:

Output Screens:

4.2.1 Play Screen

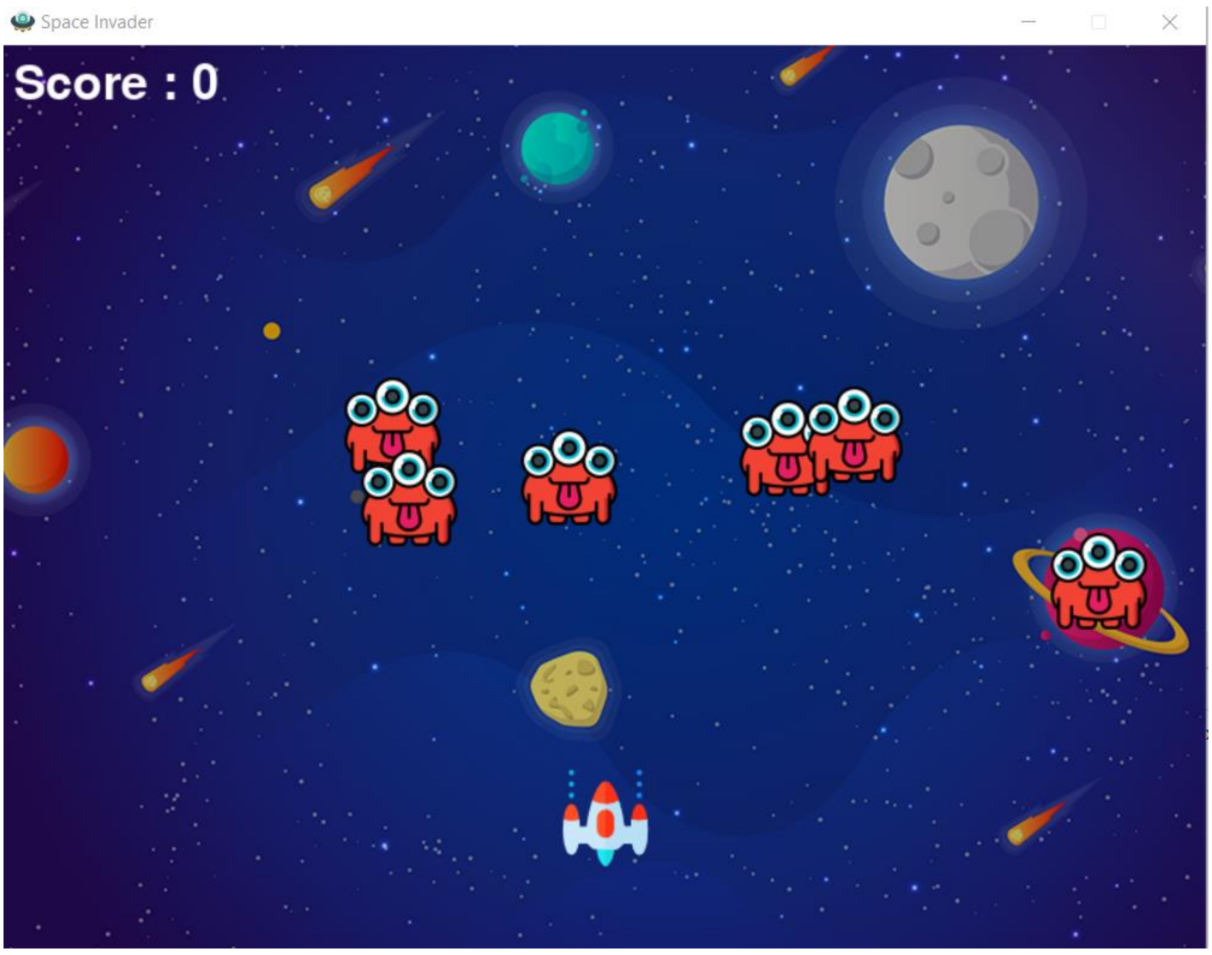
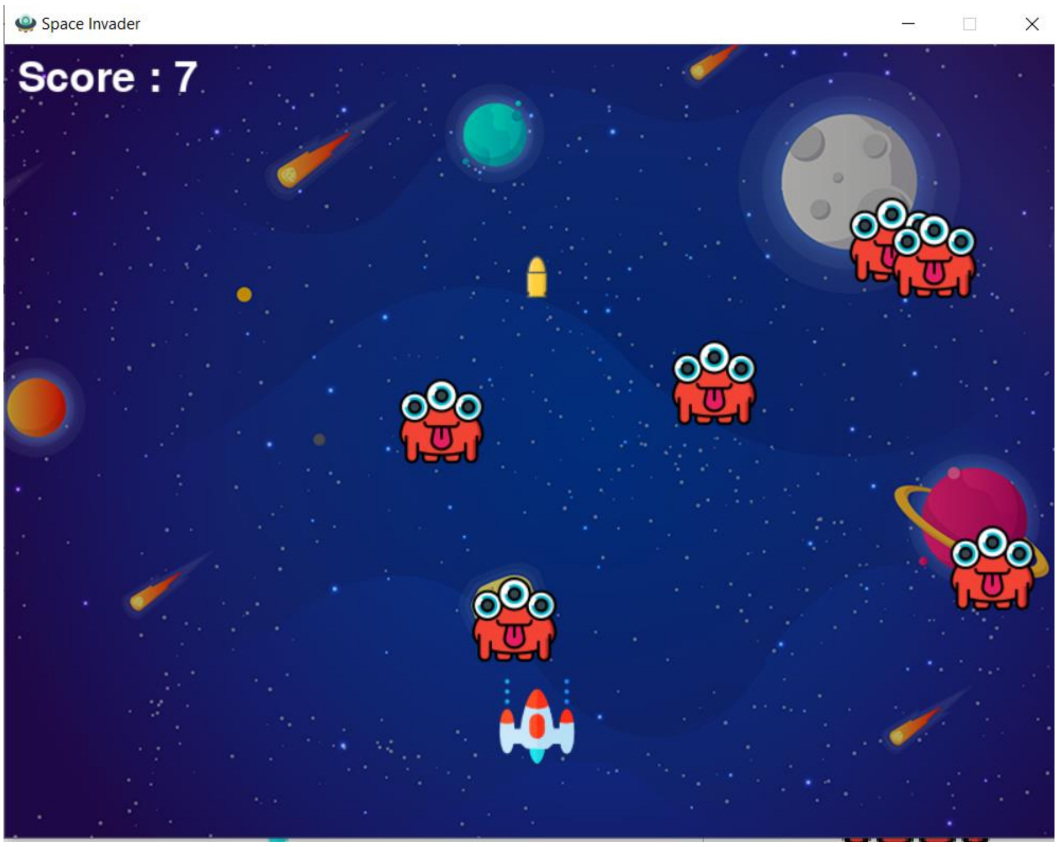
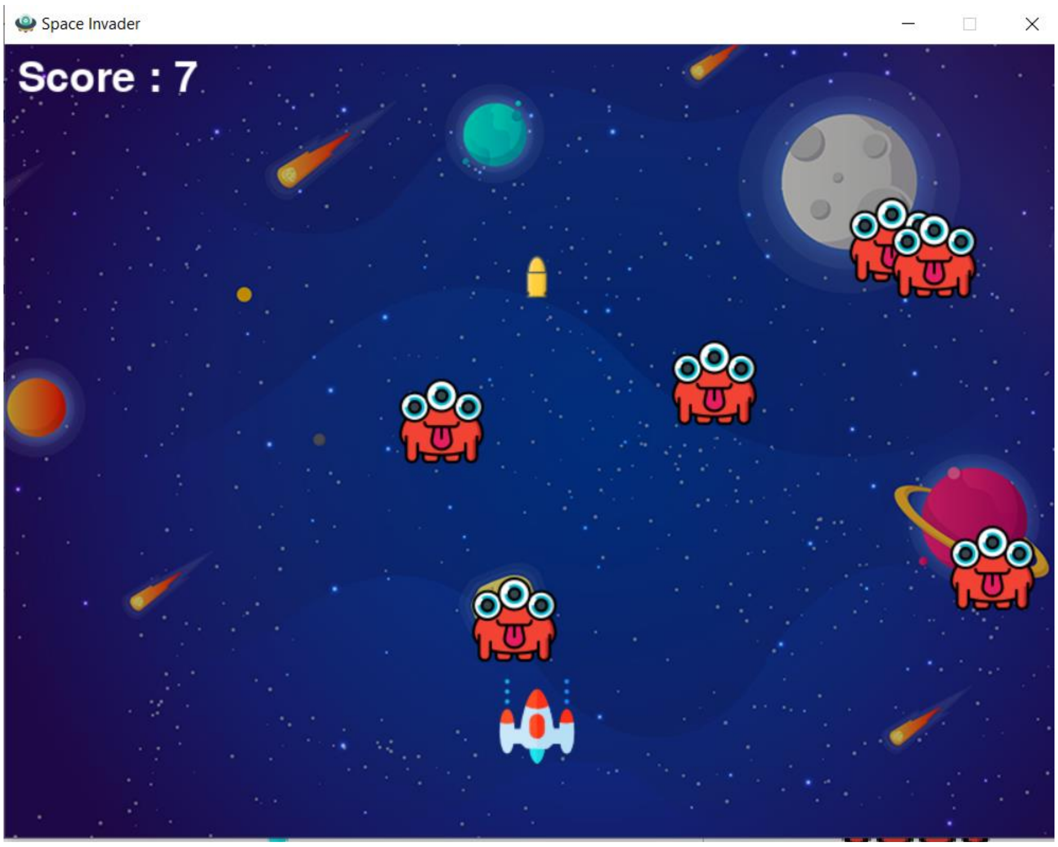


Fig.4.2.1 Play Screen

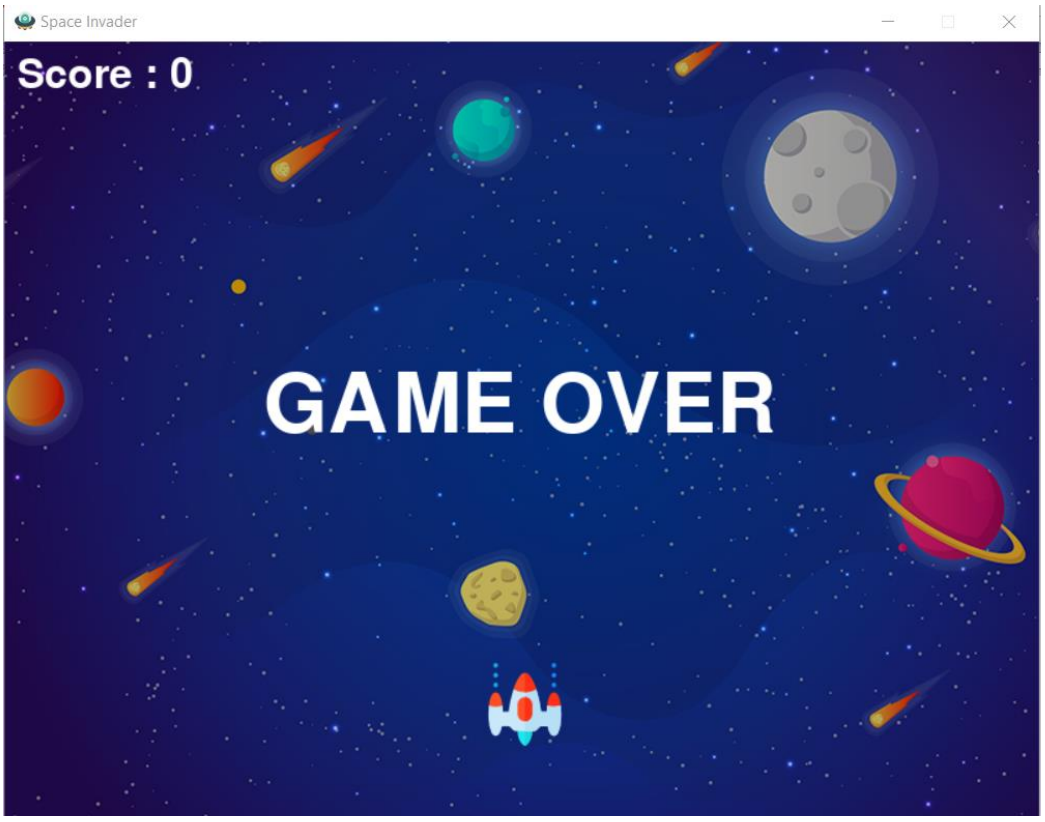
4.2.2 Bullet Fire



4.2.3 Live Score Board



4.2.6 Game Over



Chapter 5

➢ CONCLUSION

It has been a great pleasure for us to work on this exciting and challenging project. This project proved good for us as it provided practical knowledge of not only programming in Python and pygame based application and to some extent understanding game development of python.

IGN gave the game a mediocre score of 5.4 out of 10, quipping that the game is as addictive as it was shallow. Despite criticizing its playability and its "no skill" requirement, IGN noted that the gameplay made it "an addictive short- term distraction" for the casual skill and score-obsessed players. The game's difficulty has been a source of ire for many users, with one user stating that it took him half an hour to achieve a score of five points.

CHAPTER 6

REFERENCES

Bibliography:

- “Python for everybody”, Charles R.Severance
- “Think Python”, Allen B.Downey
- ”Python programming”, Mark lutz

Websites:

1. www.slideshare.net
2. https://digitalbreed.com
3. www.studentsproject.in
4. https://en.wikipedia.org
5. www.youtube.com/micro demy
6. www.code.org
7. www.scratch.mit.edu/projects/18262469/
8. www.google.com
9. www.tutorialspoint.com
10. www.scribd.com