# Galagala: A tribute to Galaga that tests your typing skills

**Group 6C's members**
Student name 1: Le Viet Hai - 1008033
Student name 2: Aryan Mahesh - 1008484
Student name 3: Bhavna Ganesh - 1008447
Student name 4: Su Keming - 1007898
Student name 5: Ie Keith Ferdinand Ijaya - 1008476
Student name 6: Zaw Ye Htet - 1008262

## 1 Introduction

The gaming experience of Galagala revolves around a nostalgic spaceship conflict reminiscent of the classic Galaga game. However, a distinctive feature introduces a compelling twist – whenever an enemy ship collides with the player's spaceship, a prompt appears on the screen, requiring the player to swiftly type the displayed letters to resume the game. Additionally, a formidable boss enemy enters the scene each time an enemy ship breaches the defensive line. Moreover, players need to skillfully evade the fire bullets launched by the boss enemy, which adds challenge to the gameplay. The game is developed using Object-Oriented Programming (OOP) and employing distinct classes for various components of the game. This strategic approach is instrumental in enhancing code organisation, facilitating an efficient structure, and streamlining the debugging process for future developments.

## 2 Structure

Galagala was created in Python using its Turtle module. The user interacts with the Turtle screen using tkinter.

### 2.1 Directory Tree

```
/
├── char
│   ├── boss.gif
│   ├── bullet.gif
│   ├── explosion.gif
│   ├── fire.gif
│   ├── minion.gif
│   └── spaceship.gif
├── sound
│   ├── bgm.wav
│   ├── explode.wav
│   └── shooting.wav
├── Character.py
└── main.py
```

To start the game, **main.py** is run. The file accesses classes in **Character.py** which defines each entity on screen. It uses audio files stored in the **sound** folder to play music and sound effects.

Each entity on screen is a **turtle.Turtle()** object with its shape set to a gif file stored in the **char** folder.

## 2.2 Class Structure

```
Character.py
├── Player
├── Bullet
├── Fire # FX
├── Explosion
├── Explosion1
├── Letter # enemy type
├── Stupid # distraction
├── Enemy # enemy type
└── Boss
```

The **Player()** class creates a **turtle.Turtle()** object that the user will control using the left and right arrow keys. **Player()** contains methods to:

1. move the player left and right,
2. fire a bullet every time the space bar is pressed,
3. detect when the player has collided with an enemy type and
4. reward bullets to the player once an enemy is hit.

```
class Player(turtle.Turtle):
def __init__(self):
    super().__init__()
    self.shape('char/spaceship.gif')
    self.penup()
    self.goto(0, -250)

    self.bullets = []
    self.number_of_bullets = 10
    self.bullets_display = turtle.Turtle()
    self.bullets_display.speed(0)
    self.bullets_display.color("white")
    self.bullets_display.penup()
    self.bullets_display.hideturtle()
    self.bullets_display.goto(290, 260)
    self.bullets_display.clear()
    self.bullets_display.write(f"Bullets: {self.number_of_bullets}", align="right",

    self.move_speed = 10  # Adjust the movement speed
    self.move_left = False
    self.move_right = False

    # Keyboard bindings
    turtle.listen()
    turtle.onkeypress(self.start_move_left, "Left")
    turtle.onkeypress(self.start_move_right, "Right")
    turtle.onkeyrelease(self.stop_move_left, "Left")
    turtle.onkeyrelease(self.stop_move_right, "Right")
    turtle.onkeypress(self.fire_bullet, "space")


    # Start continuous movement check
    self.move()
```

```
[...]
```

The **Bullet()** class is a turtle.Turtle() object that adds a turtle at the position of the player and sets the turtle's shape to *bullet.gif*. **Fire()**, **Explosion()** and **Explosion1()** also function the same way to add visual effects and increase the player's engagement with the experience.

```
class Bullet(turtle.Turtle):
    def __init__(self):
        super().__init__()
        self.shape("char/bullet.gif")
        self.penup()
        self.hideturtle()
```

The **Enemy()** class, once instanced, creates a **turtle.Turtle()** object with the shape set to *minion.gif*. Each **Enemy()** is given an initial speed at which they are moved while the game runs. The class allows the enemies to appear randomly at the top of the screen. If the enemy is hit by the player's bullet, the class contains a method to display the explosion and hide the turtle representing the enemy.

Once the player is hit by an enemy type, a "boss fight" (Fig 1) is triggered where a series of lowercase letters descend from the top of the screen. The player is required to hit all on-screen letters on their keyboard before any of the letters reach the bottom of the screen. The **Letter()** class defines a **turtle.Turtle()** object with a random lowercase letter of the alphabet. Each **Letter()** is given a random speed and position at the top of the screen. If the player hits the key that corresponds to the **Letter()** object's string, the class contains the **die()** method to change the object's status. Failing this boss fight will end the game (Figure 5).

```
class Letter(turtle.Turtle):
    def __init__(self, s):
        super().__init__()
        self.penup()
        self.color("white")
        self.hideturtle()
        self.max = 1
        self.speed = self.max
        self.goto(random.randint(-290, 290), random.randint(100,300))

        self.alive = True

        self.string = s
        turtle.listen()
        turtle.onkeypress(self.die, self.string)

    def die(self):
        self.alive = False
```

To increase the difficulty of the game, the class **Stupid()** (Figure 2) was written to generate turtles that travel across the screen along with the enemies but do not collide with the player.

```
class Stupid(turtle.Turtle):
    def __init__(self, shapedir):
        super().__init__()
        turtle.addshape(shapedir)
        self.shape(shapedir)
        self.penup()
```

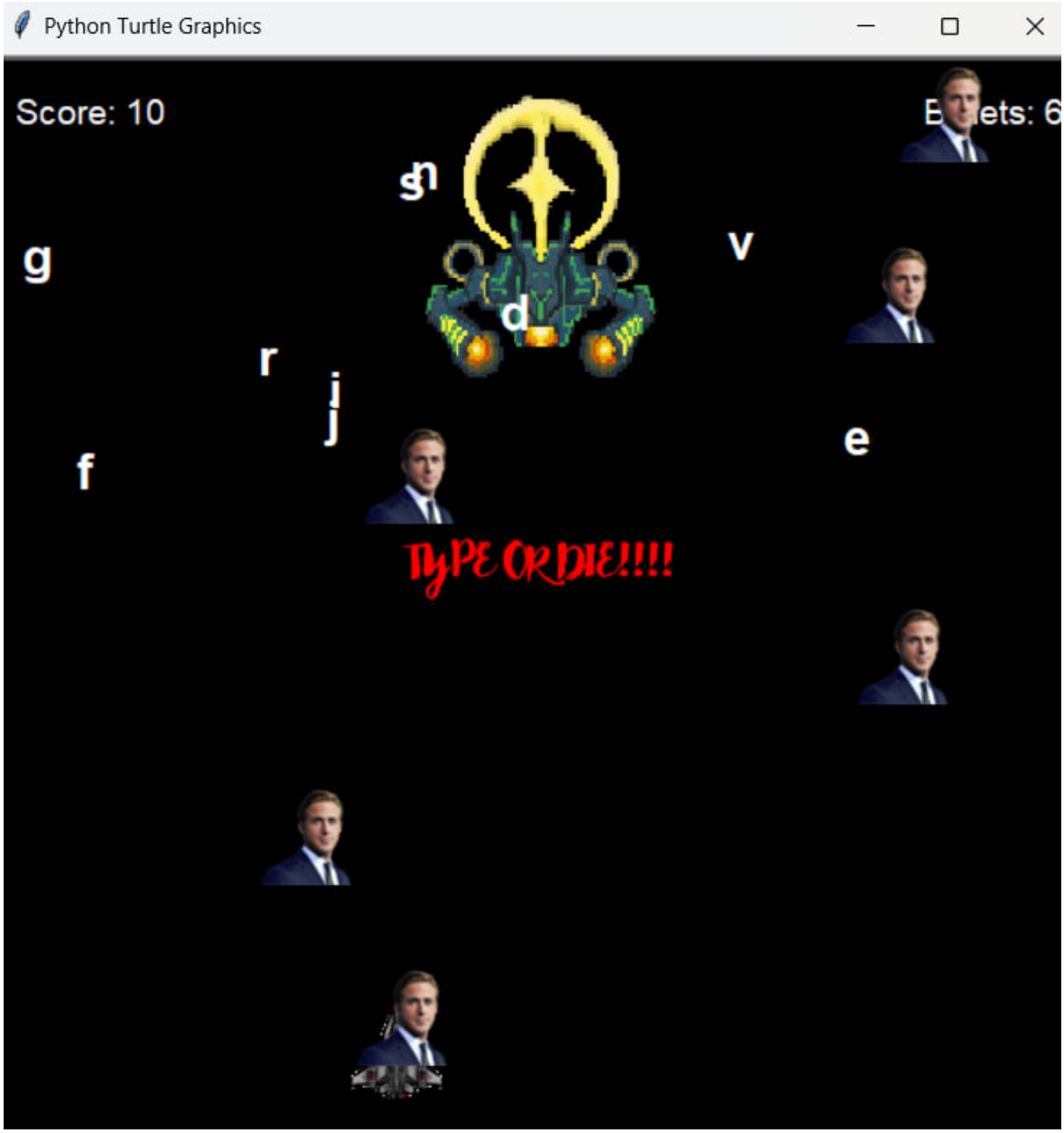


Figure 1: The Screen During A Boss Fight

```
        self.speed = 4
        self.goto(random.randint(-290, 290), 280)
```

Finally, once a certain number of points are scored by the player, a **Boss()** object is instanced. This object fires bullets at the player and increases the speed at which enemies move in the game. To defeat the boss, the player has to dodge incoming enemies and fire while firing bullets at the objects repeatedly. Each **Boss()** object has the attribute *hp* which reduces every time the player's bullet collides with the boss. The class also checks every frame of the game during runtime for such a collision event. Once *hp* reaches zero, the **die()** method is called to display a large explosion and hide the object.

```
class Boss(turtle.Turtle):
def __init__(self):
    super().__init__()
    self.shape("char/boss.gif")
    self.penup()
    self.goto(0, 200)
    self.hp = 100  # Set HP attribute for the boss

    self.fire_bullets = []
    self.explosion = Explosion1()


def fire_player(self, player, number):
    for i in range(number):
        fire = Fire()
        fire.goto(self.xcor(), self.ycor())
        fire.showturtle()
        self.fire_bullets.append(fire)

def reduce_hp(self):
    self.hp -= 10  # Reduce HP by 10 upon collision with a bullet


def die(self):
    if self.hp <= 0:
        self.hideturtle()
        self.explosion.goto(self.xcor(), self.ycor())
        self.explosion.showturtle()
        turtle.ontimer(lambda: self.explosion.hideturtle(), 500)
        self.clear()
        print("Boss defeated")
        return True
    else:
        return False

def check_collision(self, bullet):
    if self.distance(bullet) < 50:
        self.reduce_hp()
        return True
    return False
```

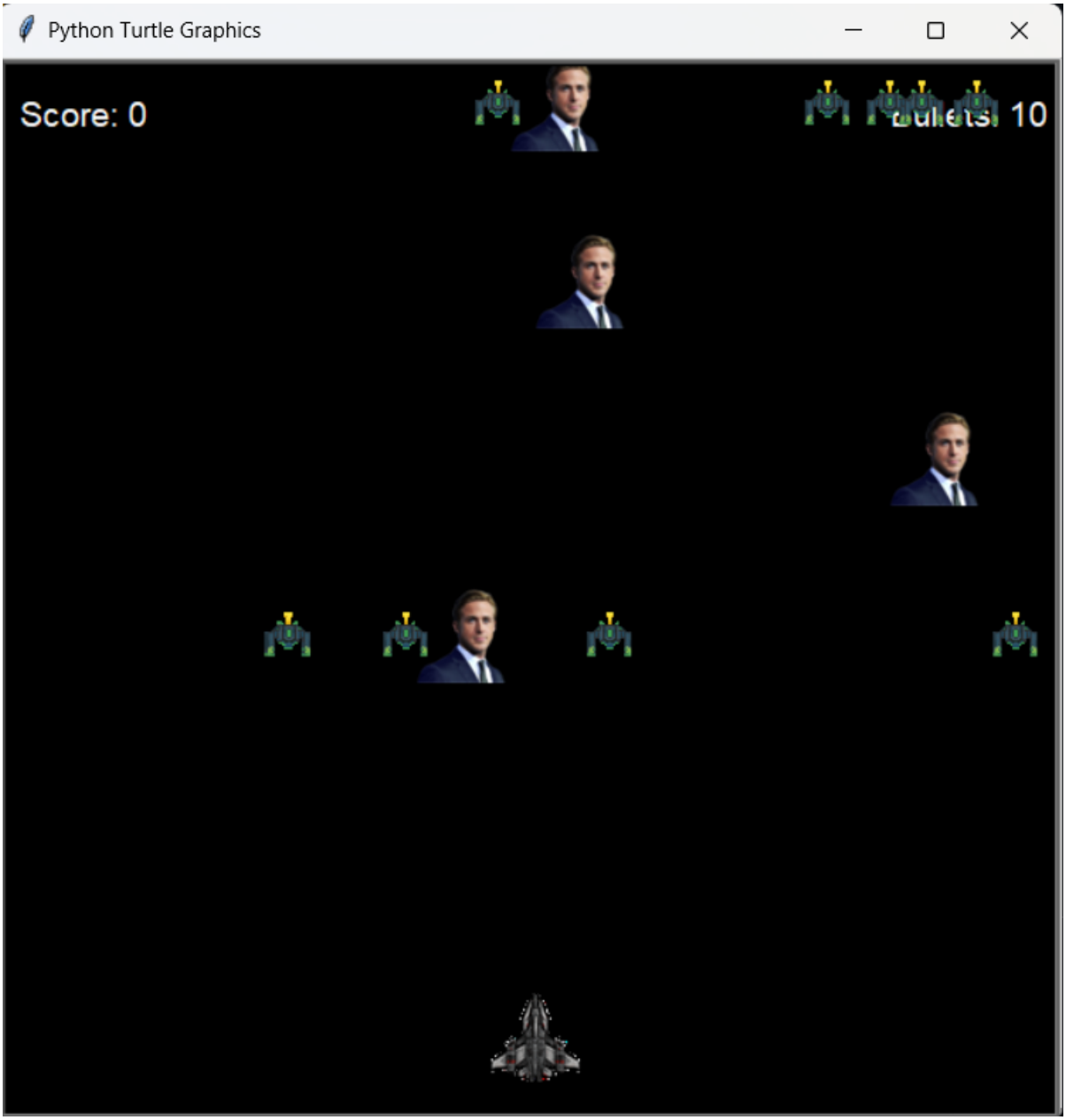


Figure 2: The "Stupids" have their shape set to a random image in the directory

# 3 Game Logic

The file **main.py** contains the logic of the game. All attributes of the game are stored in the **Game()** class.

```
class Game:
def __init__(self):
    self.wn = turtle.Screen()
    # self.wn.title("Galaga")
    self.wn.bgcolor("black")
    self.wn.setup(width=600, height=600)
    self.wn.tracer(0)
    self.wn.frames = 0

    self.player = Player()
    self.mess = Text()
    self.score = 0

    self.number_of_enemies = 5
    self.enemies = []
    self.letters = []
    self.stupids = []

    self.boss_appeared = False
    self.type_boss = False
    self.type_fight = False

    self.game_running = False

    self.number_of_words = 10
    self.boss_attack_speed = 5
```

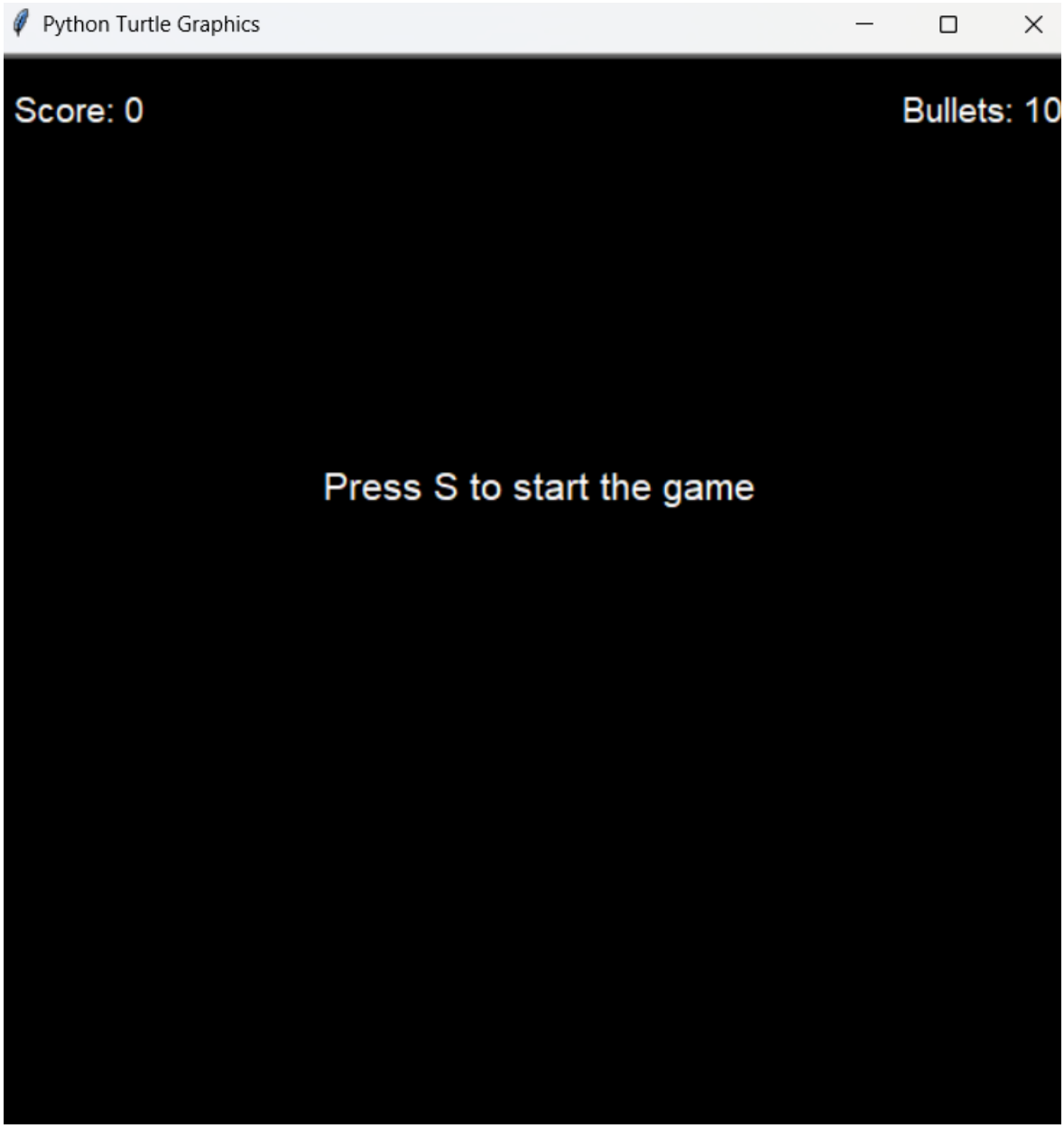


Figure 3: Start Screen

The following functions are used to control the game and the entities that appear during runtime.

```
def create_enemy(self, n): [...]
def create_letter(self, n): [...]
def menu(self): [...]
def start_game(self): [...]
def restart_game(self): [...]
def continue_game(self): [...]
```

The function **game_loop()** updates the variables before each frame of the game is displayed. It controls the movement of the player, the enemies and the bullets. It also initiates the boss fight if the player is hit and creates new waves of enemies every five seconds. Failing the boss fight will end the game (Figure 5).

```
def game_loop(self):
    try:
        if self.game_running:
            self.wn.update()


            # Move bullets
            for bullet in self.player.bullets: [...]


            if self.type_fight: # Boss Fight
                if self.type_boss == False:

                if len(self.letters) == 0:
                    self.type_fight = False
                    self.type_boss = False

                for letter in self.letters:
                    if letter.alive:
                        if y < -300:
                            self.restart_game()
                    else:
                        letter.clear()
                        self.letters.remove(letter)

            else:
                # Create a new enemy every 5 seconds
                if len(self.enemies) == 0 or self.wn.frames % 300 == 0:
                    self.create_enemy(self.number_of_enemies)
```

Every time the player hits an enemy (Figure 4), their score is increased by 10 points. Once the score reaches 100 points, a "mini boss" is created using the **Boss()** class.

```
if self.score > 0 and self.score % 100 == 0 and not self.boss_appeared:
    self.boss = Boss()
if self.boss_appeared:
    if len(self.boss.fire_bullets) == 0 or self.wn.frames % 200 == 0:
        self.boss.fire_player(self.player, 4)

    for bullet in self.player.bullets:
        if bullet.isvisible() and bullet.distance(self.boss) < 40:
            self.boss.reduce_hp()
```

```
            for fire in self.boss.fire_bullets:
                if self.player.is_attacked(fire):
                    self.type_fight = True

            if self.boss.die():
                self.score += 50
                self.boss_attack_speed += 2
                self.player.reward_bullet(10)
```

Finally, the enemy positions are updated every frame according to their speed.

```
        for enemy in self.enemies:
            y = enemy.ycor()
            y -= Enemy.speed
            x = enemy.xcor()
            x += random.randint(1,2)
            if x > 290:
                x = -290
            enemy.setx(x)
            enemy.sety(y)
```

For every frame, the program checks if the player has collided with an enemy. If attacked, the game will prompt a boss fight. The program also removes any bullet entities that end up outside the screen.

```
    if self.player.is_attacked(enemy):
            self.type_fight = True

        for bullet in self.player.bullets:
            if bullet.isvisible() and bullet.distance(enemy) < 20:
                bullet.hideturtle()
                enemy.disappear()
                self.player.reward_bullet(3)
                self.score += 10

            if bullet.ycor() > 300:
                if bullet in self.player.bullets:
                    self.player.bullets.remove(bullet)

    if self.player.isvisible():
        self.wn.ontimer(self.game_loop, 10)
        self.wn.frames += 1

    except turtle.Terminator:
        pass
```

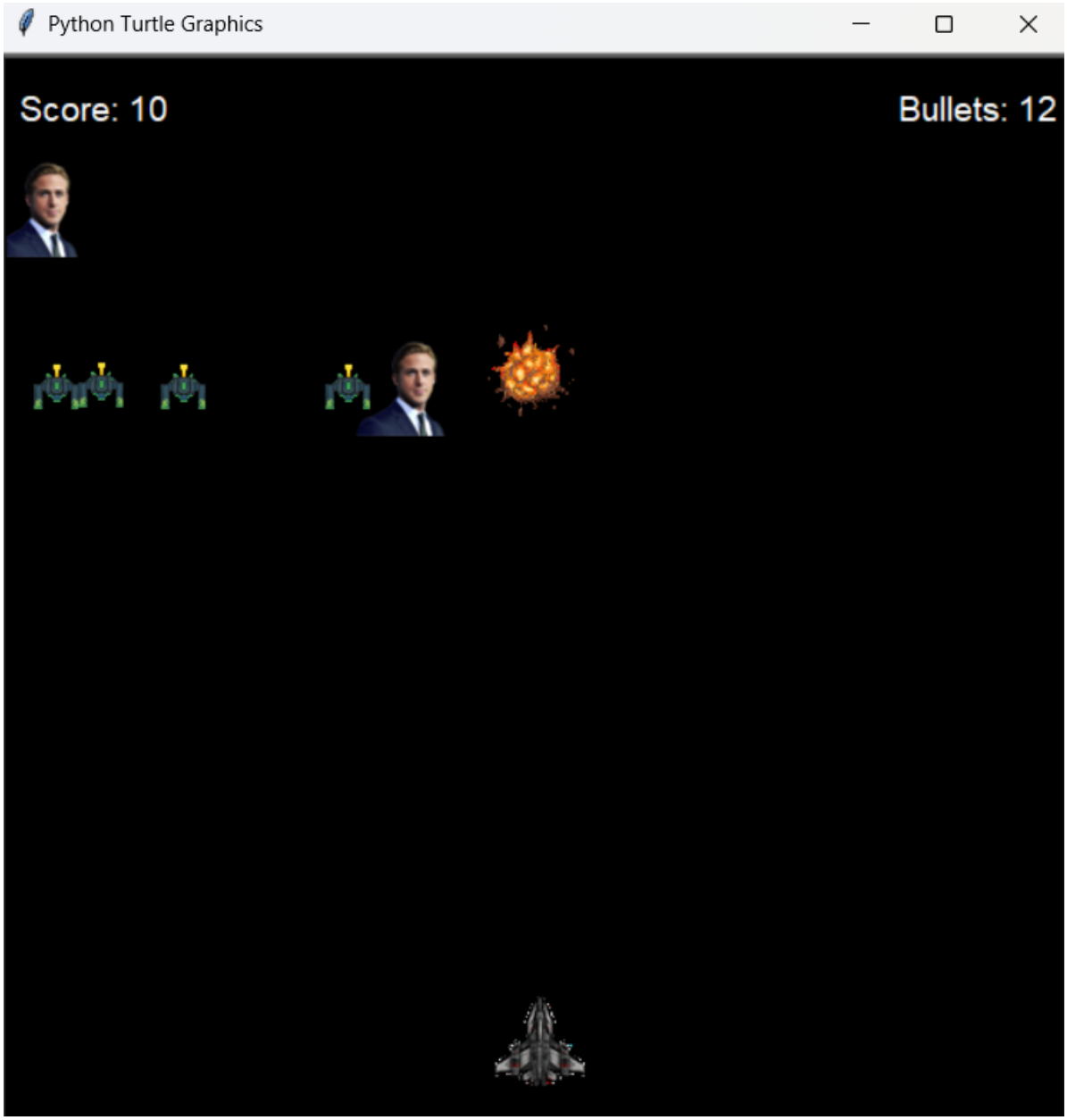


Figure 4: Enemy is hit by Player's Bullet

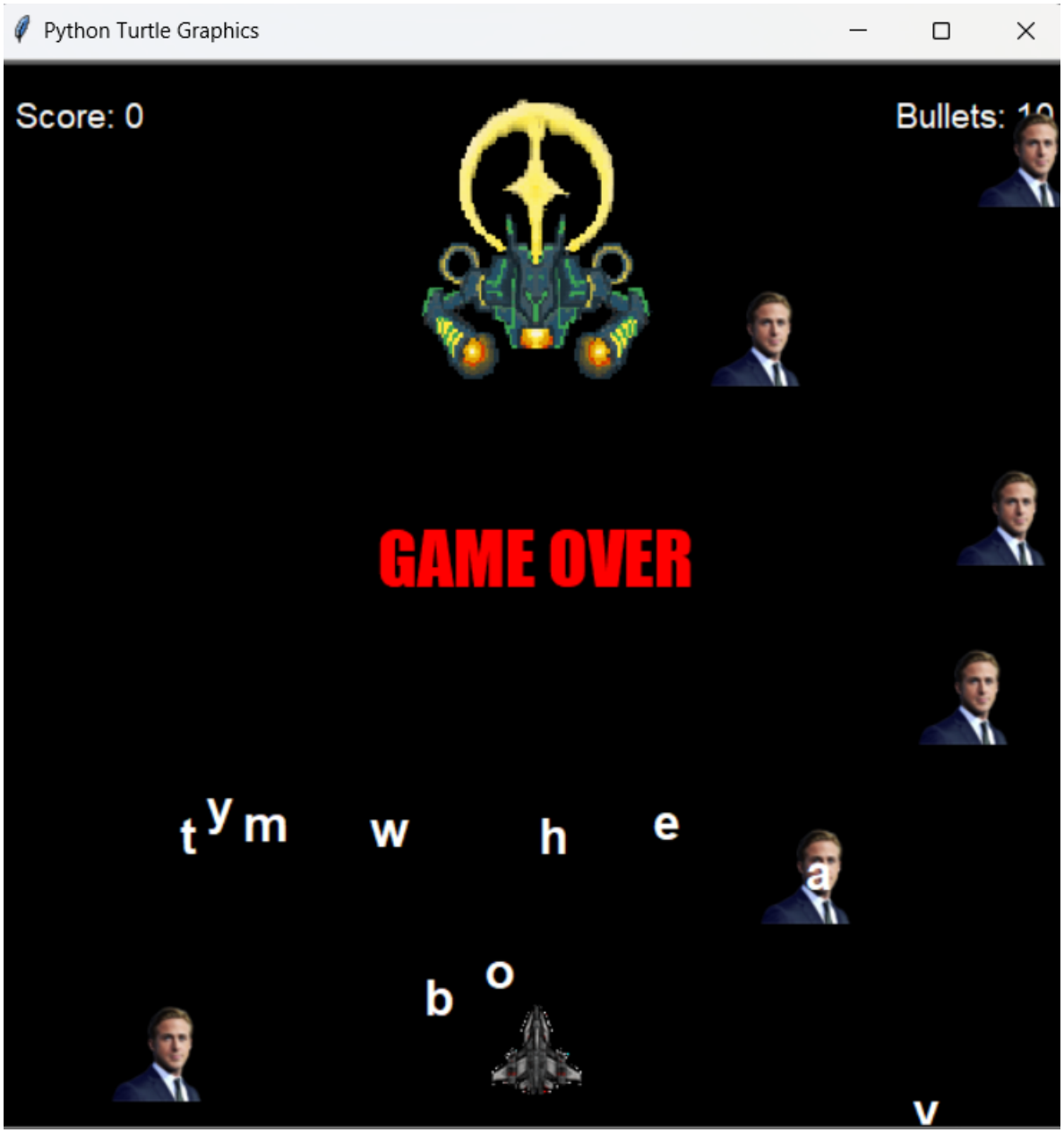


Figure 5: Game Over

# 4 References and Materials

## 4.1 Modules Used

1. Turtle Documentation
2. Tkinter
3. Turtle Graphics

## 4.2 Guides/Inspiration

1. Galaga (1981)
2. Typing Game with Python and Turtle
3. MonkeyType

## 4.3 Materials

1. Galagala Github Repository
2. Video Demonstration