PROJECT REPORT :SPACESHIP FIGHTHING USING PYGAME

Fall 2024

CSE208L Object Oriented Programming Lab

Submitted by:

SULEMAN MAJEED(23PWCSE2240)

IMAD AHMED(23PWCSE2297)

MUHAMMAD ABUBAKAR BANGASH (23PWCSE2236)

Class Section: **A**

ASSIGNED BY

ENGR.SUMAYYE SALAHUDDIN

JANUARY 28,2025

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar

SPACESHIP FIGHTHING GAME

OBJECTIVES :

- Provide a fun and engaging gameplay experience.
- Improve players' reflexes and hand-eye coordination.
- Offer increasing difficulty to keep the game interesting.
- Demonstrate the use of Pygame for creating interactive games.
- Encourage users to explore game design and development.

APPLICATIONS:

- The spaceship flying game tests players' reflexes and decision-making as they navigate a dynamic environment.
- It provides an immersive gaming experience with colorful graphics and engaging sound effects.
- The game can serve as a learning tool for beginners exploring Python and Pygame development.
- Players can compete for high scores, making it a competitive and replayable game.
- It demonstrates how interactive elements, like obstacles and rewards, can be implemented in games.

INTRODUCTION:

The spaceship flying game is a fun and fast-paced arcade game where players control a spaceship in an exciting space adventure. The goal is to dodge obstacles, collect power-ups, and score points. Made using Pygame, the game shows simple game programming features like handling sprites, detecting collisions, and making levels more challenging. It’s easy to play, fun to enjoy, and a great learning project for beginners.

SOURCE CODE:

```
import pygame
import os
pygame.font.init()
pygame.mixer.init()

WIDTH, HEIGHT = 900, 500
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
RED = (255, 0, 0)
YELLOW = (255, 255, 0)

FPS = 60
SPACESHIP_WIDTH, SPACESHIP_HEIGHT = 55, 40
VEL = 5
BULLET_VEL = 7
MAX_BULLETS = 3

BORDER = pygame.Rect(WIDTH // 2 - 5, 0, 10, HEIGHT)

YELLOW_HIT = pygame.USEREVENT + 1
RED_HIT = pygame.USEREVENT + 2

WIN = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Game with Levels")

HEALTH_FONT = pygame.font.SysFont('comicsans', 40)
WINNER_FONT = pygame.font.SysFont('comicsans', 100)
LEVEL_FONT = pygame.font.SysFont('comicsans', 60)
```

Library Imports:

- The code imports the pygame library for game development and the os module for file handling.

Initialization:

- Initializes pygame modules for fonts and audio mixer functionality.

Window Settings:

- Defines the window size (900x500 pixels) and sets its background colors (white, black, red, yellow).

Game Constants:

- **FPS** (frames per second) to control the game speed.
- Spaceship dimensions, velocity, and maximum bullets per player.

Game Objects:

- Creates a dividing border in the middle of the game screen.
- Defines custom pygame events for spaceship collisions (YELLOW_HIT and RED_HIT).

Game Window Initialization:

- Sets up the game window and displays a title ("Game with Levels").

Font Settings:

- Defines fonts with different sizes to display player health, the winner, and levels using the "Comic Sans" font.

```
YELLOW_SPACESHIP_IMAGE = pygame.image.load(os.path.join('Assets', 'spaceship_yellow.png'))
YELLOW_SPACESHIP = pygame.transform.rotate(
    pygame.transform.scale(YELLOW_SPACESHIP_IMAGE, (SPACESHIP_WIDTH, SPACESHIP_HEIGHT)), 90)

RED_SPACESHIP_IMAGE = pygame.image.load(os.path.join('Assets', 'spaceship_red.png'))
RED_SPACESHIP = pygame.transform.rotate(
    pygame.transform.scale(RED_SPACESHIP_IMAGE, (SPACESHIP_WIDTH, SPACESHIP_HEIGHT)), 270)

SPACE = pygame.transform.scale(pygame.image.load(os.path.join('Assets', 'space.png')), (WIDTH, HEIGHT))

class Spaceship:
    def __init__(self, x, y, image, color):
        self.rect = pygame.Rect(x, y, SPACESHIP_WIDTH, SPACESHIP_HEIGHT)
        self.image = image
        self.color = color
        self.health = 10
        self.bullets = []

    def draw(self):
        WIN.blit(self.image, (self.rect.x, self.rect.y))
        for bullet in self.bullets:
            pygame.draw.rect(WIN, self.color, bullet)

    def move(self, keys, controls):
        if keys[controls["left"]] and self.rect.x - VEL > 0:
            self.rect.x -= VEL
        if keys[controls["right"]] and self.rect.x + VEL + self.rect.width < WIDTH:
            self.rect.x += VEL
```

Image Loading and Transformation:

- The code loads images for a yellow spaceship, a red spaceship, and a space background. It then scales these images to appropriate sizes and rotates the spaceship images, likely to orient them correctly.

Spaceship Class:

- A `Spaceship` class is defined, representing a spaceship object in the game. Each spaceship has attributes like its rectangular area (`rect`), image, color, health, and a list to store its bullets.

Spaceship Initialization:

- The `__init__` method of the `Spaceship` class initializes these attributes when a new spaceship object is created. It takes the initial x and y coordinates, the image, and the color as input.

Drawing:

- The `draw` method handles drawing the spaceship image onto the game window (`WIN`). It also iterates through the spaceship's bullet list and draws each bullet (presumably as a colored rectangle).

Movement:

- The `move` method controls the spaceship's movement. It checks for user input (keyboard presses) for "left" and "right" controls. It then adjusts the spaceship's x-coordinate (`self.rect.x`) to move it, ensuring it stays within the game window's boundaries. The `VEL` variable likely controls the speed of movement.

Classes and Objects

- A class is a blueprint for creating objects.
- An object is an instance of a class.

```
class Spaceship:
    def __init__(self, x, y, image, color):
        self.rect = pygame.Rect(x, y, SPACESHIP_WIDTH, SPACE
        self.image = image
        self.color = color
        self.health = 10
        self.bullets = []
```

Objects created from the class:

```
yellow = YellowSpaceship(100, 300)
red = RedSpaceship(700, 300)
```

Inheritance

- Inheritance allows a new class to derive properties and methods from an existing class.
- The game uses single inheritance, where a subclass inherits from one parent class.

```
class YellowSpaceship(Spaceship):
    def __init__(self, x, y):
        super().__init__(x, y, YELLOW_SPACESHIP, YELLOW)
```

RedSpaceship and YellowSpaceship inherit from Spaceship and reuse its methods and attributes

Encapsulation

- Encapsulation ensures that data is hidden and only accessible through defined methods.
- Example: Spaceship's health is only modified by event handling, preventing direct modification.

```
if event.type == RED_HIT:
    game.red.health -= 1
if event.type == YELLOW_HIT:
    game.yellow.health -= 1
```

Methods in Classes

- Methods define behavior for objects.
- Example: The move method allows spaceship movement.

```
def move(self, keys, controls):
    if keys[controls["left"]] and self.rect.x - VEL > 0:
        self.rect.x -= VEL
    if keys[controls["right"]] and self.rect.x + VEL + self.
        self.rect.x += VEL
```

Game Class and Object Interaction

The Game class manages the entire game flow

```
class Game:
    def __init__(self):
        self.yellow = YellowSpaceship(100, 300)
        self.red = RedSpaceship(700, 300)
        self.level = 1
```

Additional OOP Concepts Used

- Polymorphism: Both YellowSpaceship and RedSpaceship inherit from Spaceship, but they behave differently based on their specific attributes.
- Abstraction: The Spaceship class abstracts the common properties of spaceships so specific spaceship types can focus on their unique attributes.

```
        elif game.yellow.health <= 0:
            winner_text = "Red Wins!"
            game.next_level("Red")

        if winner_text:
            game.draw_winner(winner_text)

        keys_pressed = pygame.key.get_pressed()
        game.yellow.move(keys_pressed, {"left": pygame.K_a, "right": pygame.K_d, "up": pygame.K_w, "down": pygame.K_s
        game.red.move(keys_pressed, {"left": pygame.K_LEFT, "right": pygame.K_RIGHT, "up": pygame.K_UP, "down": pygam

        game.handle_bullets()
        game.draw_window()

if __name__ == "__main__":
    main()
```

Winner Check:

- Determines if a spaceship has won by checking if either spaceship's health is zero or less. If so, it sets the winner text and triggers the next level (likely with the winner's color).

Display Winner:

- If there's a winner, displays the winner text on the screen.

Input:

- Captures which keys are currently pressed by the player.

Movement:

- Moves the yellow and red spaceships based on the pressed keys (using WASD for yellow, arrow keys for red).

Bullets:

- Manages bullet behavior (updates positions, checks collisions, etc.).

Drawing:

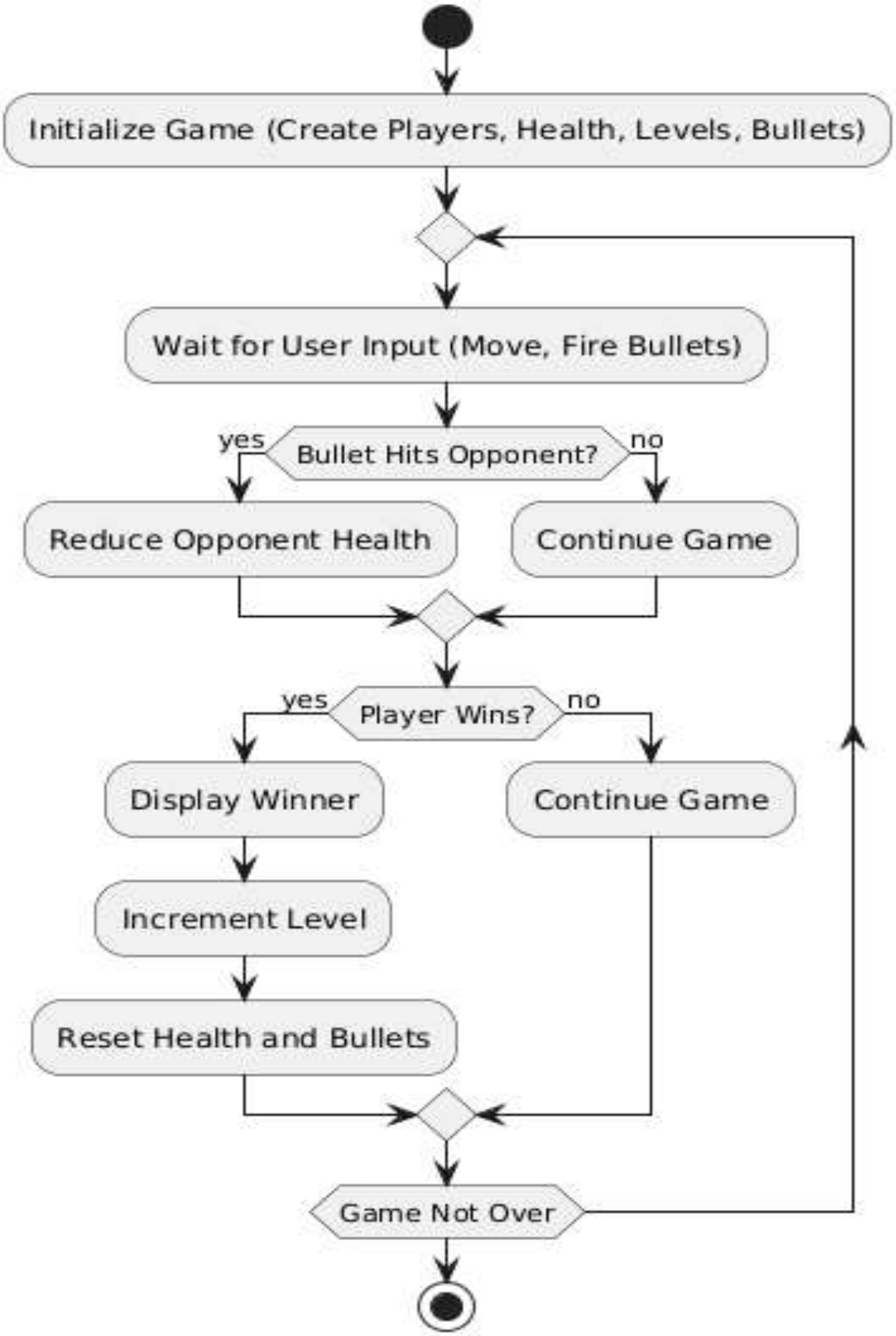
- Redraws the entire game window to reflect changes.

Game Loop:

- The `main()` function (called at the end) contains the core game loop, repeatedly executing these steps to run the game.

OUTPUT:

FLOWCHART:


Initialize Game (Create Players, Health, Levels, Bullets)
Wait for User Input (Move, Fire Bullets)
Bullet Hits Opponent?
yes
no
Reduce Opponent Health
Continue Game
Player Wins?
yes
no
Display Winner
Continue Game
Increment Level
Reset Health and Bullets
Game Not Over

SOURCES:

- Youtube
- chatgpt