Sri Sivasubramaniya Nadar College of Engineering, Kalavakkam – 603 110
(An Autonomous Institution, Affiliated to Anna University, Chennai)
Department of Information Technology

Mini project Report

Degree	B.Tech	**Branch**	All Branches
Semester: I	Academic Year: 2024-2025		Regulation: R2024
Subject Code & Name	UGE3188 – Problem Solving and Programming using Python		

(K1: Remembering, K2: Understanding, K3: Applying, K4: Analyzing, K5: Evaluating, K6: Creating)

CO5:	Create simple software development projects in teams using best coding practices and communicate effectively through reflections, reports, and presentations. (K6)

Project Title:

Emojis Shooter using Pygame

Team Members

Student Reg No	Name	Signature
3122245001077	Keerthana S	
3122243001092	Sai Ramya R U	
3122243001035	Harshini Ashok	

Date of Submission: 14/12/2024

1. Introduction

The "Emoji Shooter" is a simple arcade-style game developed using Python and the Pygame library. In this game, players control an emoji shooter and aim to destroy falling targets (emojis or objects) by shooting projectiles. The game involves quick reflexes and precise aiming, making it both entertaining and a great starting point for learning game development concepts like collision detection, sprite movement, and event handling in Pygame.

Problem Statement

The "Emoji Shooter" aims to address this by providing a straightforward, engaging project to introduce these core concepts. The game presents the following challenges:

1. Implementing player-controlled movement and shooting mechanics.
2. Managing multiple objects like falling targets and projectiles simultaneously.
3. Detecting and handling collisions between bullets and targets.
4. Creating a scoring system to track progress and motivate the player.
5. Ensuring smooth gameplay with proper frame updates and animations**.**

Objective and Scope

The objective of the "Emoji Shooter" game is to create an interactive, beginner-friendly project that introduces core concepts of game development using Python and Pygame. The game is designed to

1. Provide a hands-on learning experience for implementing fundamental game mechanics like player movement, shooting, and collision detection.

2. Develop a functional arcade game with a scoring system and simple controls.

3. Encourage problem-solving and creativity by incorporating additional features like difficulty scaling, power-ups, or animated backgrounds.

Scope

1. Functional Scope

Gameplay Mechanics:

Players aim and shoot emojis using the mouse or keyboard.

Emojis appear at regular intervals and move randomly.

Scores are awarded for each successful hit, with bonuses for combos or special emojis.

Game Progression:

Difficulty increases as emojis move faster and appear in larger numbers.

The game ends when the timer reaches zero or a set number of emojis escape unburst.

User Feedback:

Displays real-time score, time remaining, and a final performance summary.

2. Technical Scope

Pygame Integration:

Leverages Pygame for creating the game loop, handling user input, and rendering graphics.

Modular Design:

Organized into modules for emoji management, input handling, scoring, and rendering, ensuring code reusability and scalability.

Cross-Platform Compatibility:

Runs on platforms like Windows, macOS, and Linux, provided Python and Pygame are installed.

Methodology

Here’ s a methodology framework for your Emojis Shooter project:

1. Design and Architecture

The game follows a modular architecture, with distinct components for handling the game's logic, visuals, and user interaction. The design ensures scalability and ease of modification.

Core Components:

- ***Game Loop:***

Manages the flow of the game, including initialization, updates, and rendering.

Runs continuously until the game ends (timer reaches 0 or escape conditions are met).

- ***Event Handling:***

Processes user inputs such as mouse clicks or keyboard presses.

Tracks interactions with emojis for scoring.

- ***Game Objects:***

Represents emojis and their attributes (size, speed, type).

Includes logic for movement and collision detection.

- ***Rendering System***:

Updates the game screen by drawing emojis, the score, and the timer.

Provides visual feedback for actions (e.g., burst effects).

2. Modules and Their Descriptions

- *Module 1: Initialization*

Functionality:

Load Pygame and initialize the game screen.

Define game constants (screen dimensions, emoji types, spawn rate, etc.).

Initialize variables like the score and timer.

- *Module 2: Emoji Management*

Functionality:

Create emojis with attributes (type, position, speed).

Update emoji positions based on their speed and direction.

Detect when emojis go off-screen.

- Module 3: Input Handling

Functionality:

Capture player actions using the mouse or keyboard.

Determine if a click collides with an emoji.

Handle multiple bursts for combos.

- Module 4: Scoring System

Functionality:

Increase the score for each successful burst.

Award bonus points for specific emojis or combos.

Track and display the player's score.

- Module 5: Difficulty Scaling

Functionality:

Gradually increase emoji speed and spawn rate as time decreases.

Add more challenging emojis with unique behavior.

- Module 6: End Condition

Functionality:

Terminate the game when the timer reaches 0 or escape conditions are met.

Display the final score and game-over screen.

3. Implementation

Below is a high-level implementation plan:

- Step 1: Setup Environment

Install Pygame: pip install pygame.

Create the basic game window using Pygame.

- Step 2: Develop the Game Loop

Structure the loop to update game states, process inputs, and render visuals.

Ensure smooth gameplay by maintaining a consistent frame rate.

- Step 3: Implement Emoji Logic

Use classes to define emoji attributes and behaviors.

Randomly generate emojis with varying speeds and positions.

- Step 4: Integrate Scoring and Difficulty

Maintain a score counter that updates in real-time

- Step 5: Finalize End Conditions

Display a "Game Over" screen when the timer reaches 0.

Show a summary of the player’s performance.

4.**Source code:**

```
import pygame
import random
import time

# Initialize Pygame
pygame.init()

# Game Constants
WIDTH, HEIGHT = 800, 600
FPS = 60
```

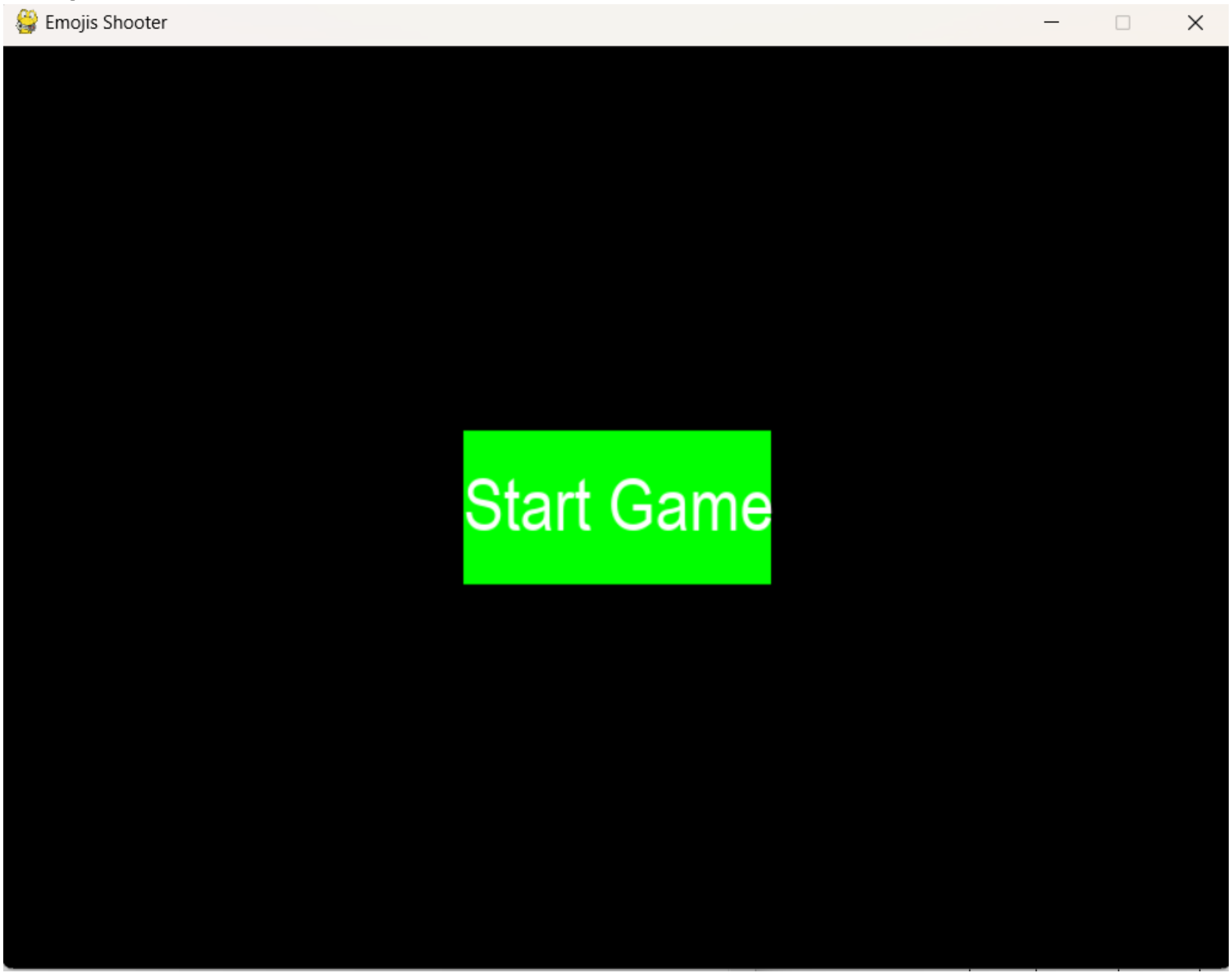
```
EMOJI_SIZE = 50
BULLET_SIZE = 10
TIME_LIMIT = 60  # Game timer in seconds
SPAWN_INTERVAL = 2000  # Emoji spawn every 2 seconds (2000 milliseconds)

# Colors
WHITE = (255, 255, 255)
RED = (255, 0, 0)
BLACK = (0, 0, 0)
LIGHT_BLUE = (173, 216, 230)  # Light blue color
GREEN = (0, 255, 0)  # Green for the start button
YELLOW = (255, 255, 0)  # Yellow for the restart button
GOLD = (255, 223, 0)

# Load sound effects
start_game_sound = pygame.mixer.Sound("button click.mp3")
game_over_sound = pygame.mixer.Sound("game over.mp3")

# Background music
pygame.mixer.music.load("background.mp3")
pygame.mixer.music.set_volume(0.3)  # Adjust the volume of the background music

# Load emoji images
emoji_images = [
    pygame.image.load("angry.png"),
    pygame.image.load("cool.png"),
    pygame.image.load("smiling face.png"),
    pygame.image.load("shocked.png"),
    pygame.image.load("crying.png")
]

# Bullet Class
class Bullet:
    def bullet_description(self, x, y):
        self.x = x
        self.y = y
        self.width = BULLET_SIZE
        self.height = BULLET_SIZE
        self.color = RED

    def move(self):
        self.y -= 10  # Move up

    def draw(self, screen):
        pygame.draw.circle(screen, self.color, (self.x, self.y), self.width)

# Emoji Class
```

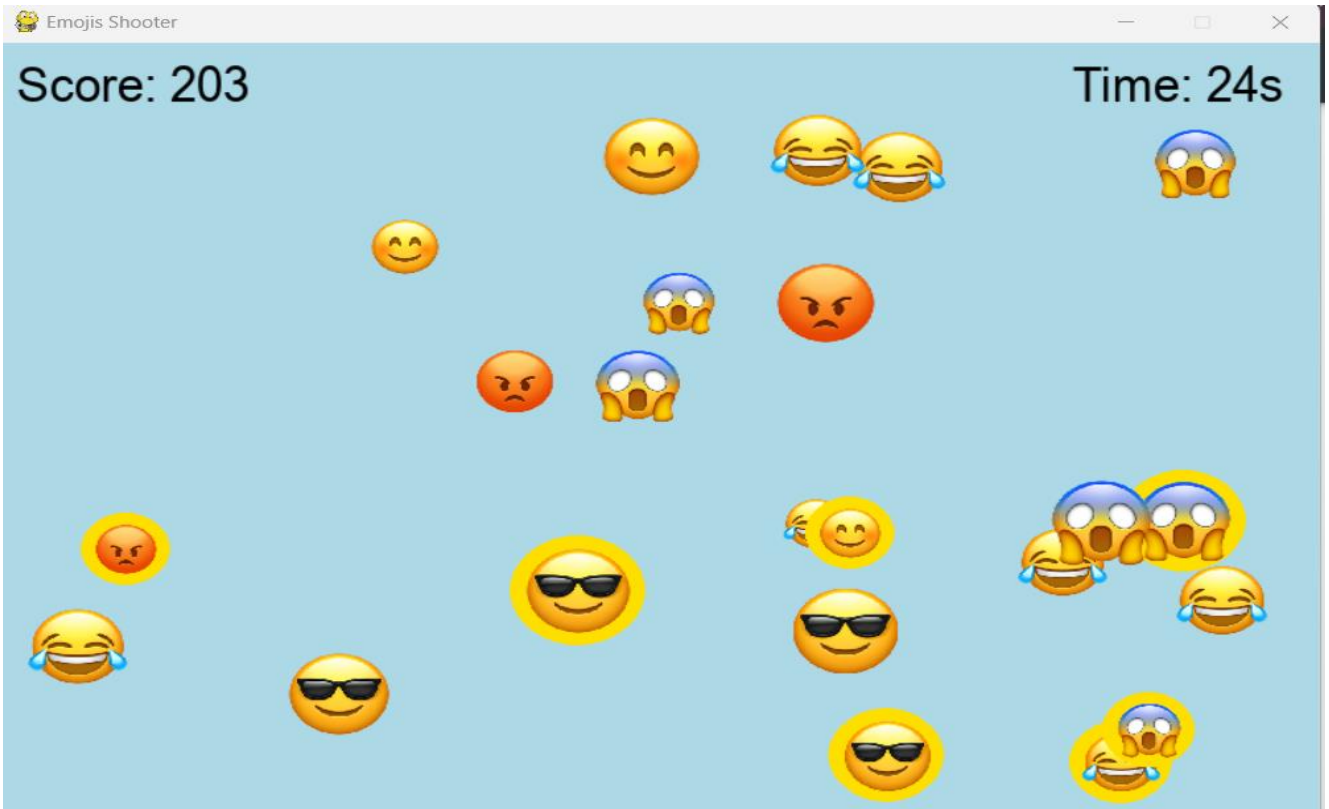
```
class Emoji:
    def emoji_description(self, emoji_images, is_special=False):
        self.image = random.choice(emoji_images)
        self.x = random.randint(0, WIDTH - EMOJI_SIZE)
        self.y = random.randint(50, HEIGHT - EMOJI_SIZE)
        self.size = random.randint(40, 70)
        self.is_special = is_special
        self.border_width = 7
        self.special_color = GOLD
        self.speed_x = random.uniform(1.5, 3.0)
        self.speed_y = random.uniform(1.5, 3.0)

    def move(self, speed_factor):
        self.x += self.speed_x * speed_factor
        self.y += self.speed_y * speed_factor
        if self.x < 0 or self.x + self.size > WIDTH:
            self.speed_x = -self.speed_x

        if self.y < 0 or self.y + self.size > HEIGHT:
            self.speed_y = -self.speed_y

    def draw(self, screen):
        if self.is_special:
            pygame.draw.circle(screen, self.special_color, (self.x + self.size // 2, self.y + self.size // 2),
self.size // 2 + self.border_width)
        screen.blit(pygame.transform.scale(self.image, (self.size, self.size)), (self.x, self.y))

    def get_points(self):
        if self.is_special:
            return 10
        return 1

# Draw Start Button
def draw_start_button(screen):
    font = pygame.font.SysFont('Arial', 48)
    text = font.render('Start Game', True, WHITE)
    button_rect = pygame.Rect(WIDTH // 2 - 100, HEIGHT // 2 - 50, 200, 100)  # Button position and
size
    pygame.draw.rect(screen, GREEN, button_rect)  # Draw button background
    screen.blit(text, (WIDTH // 2 - 100, HEIGHT // 2 - 30))  # Draw button text
    return button_rect

# Draw Restart Button
def draw_restart_button(screen):
    restart_button = pygame.Rect(WIDTH // 2 - 100, HEIGHT // 2 + 50, 200, 50)  # Positioned below
the message
    pygame.draw.rect(screen, GREEN, restart_button)  # Draw the button
    restart_text = pygame.font.SysFont('Arial', 30).render("Restart", True, WHITE)
    screen.blit(restart_text, (restart_button.centerx - restart_text.get_width() // 2,
restart_button.centery - restart_text.get_height() // 2))
    return restart_button

# Main Game Loop
def game_loop():
    global game_running, score, emojis, bullets, start_time, last_spawn_time, game_over  # Declare
needed variables as global
    running = True
    clock = pygame.time.Clock()

    # Game Variables
    emojis = []
    for _ in range(5):
        is_special = random.random() < 0.2
        emoji = Emoji()  # Create an Emoji object
        emoji.emoji_description(emoji_images, is_special)  # Initialize it
        emojis.append(emoji)

    bullets = []  # List of bullets
    score = 0
    start_time = pygame.time.get_ticks()  # Start time in milliseconds
    last_spawn_time = start_time  # Time when the last emoji was spawned

    # Create Game Window
    screen = pygame.display.set_mode((WIDTH, HEIGHT))
    pygame.display.set_caption('Emojis Shooter')

    # Play background music
    pygame.mixer.music.play(-1, 0.0)  # Loop indefinitely (-1) from the start

    # Game loop
    while running:
        current_time = pygame.time.get_ticks()

        # Fill screen with light blue background
        screen.fill(LIGHT_BLUE)

        # Handle Events
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
            if event.type == pygame.MOUSEBUTTONDOWN and not game_over:
                # Check if the start button is clicked
                mouse_x, mouse_y = pygame.mouse.get_pos()
                if start_button.collidepoint(mouse_x, mouse_y):
```

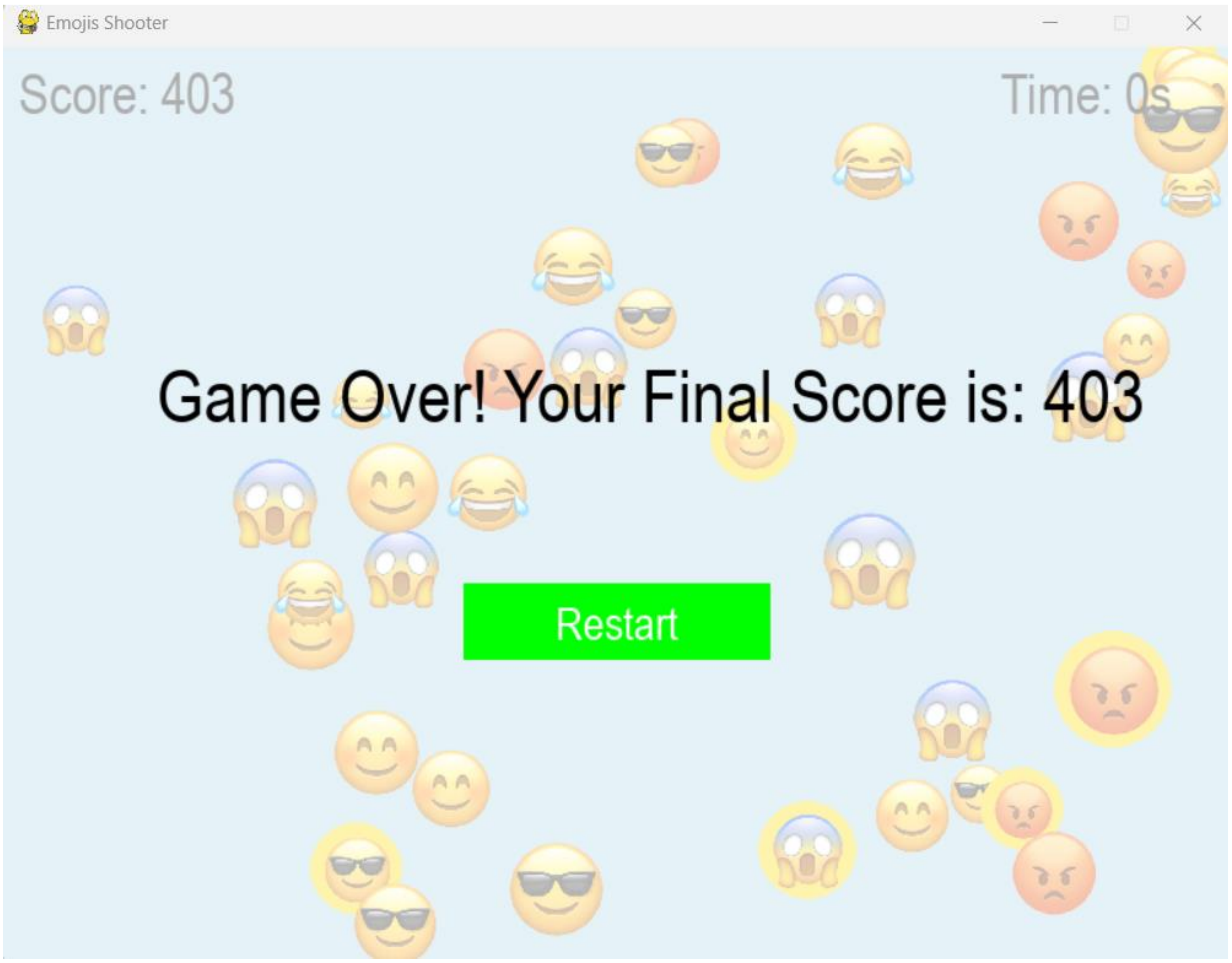
```
            game_running = True # Set game_running to True once the button is clicked
            start_game_sound.play()
        if game_running:
            bullet = Bullet()
            bullet.bullet_description(mouse_x,mouse_y)
            bullets.append(bullet)

    if event.type == pygame.MOUSEBUTTONDOWN and game_over:
        # Handle restart button click
        mouse_x, mouse_y = pygame.mouse.get_pos()
        if restart_button.collidepoint(mouse_x, mouse_y):
            restart_game()  # Restart the game if the restart button is clicked
            running = False  # Stop the current game loop

# If game is not started yet, show start screen with button
if not game_running:
    draw_start_button(screen)
else:
    if current_time - last_spawn_time >= SPAWN_INTERVAL:
        is_special = random.random() < 0.2  # 20% chance for a new emoji to be special
        emoji = Emoji()  # Create an Emoji object
        emoji.emoji_description(emoji_images, is_special)  # Initialize it
        emojis.append(emoji)
        last_spawn_time = current_time

    elapsed_time = (current_time - start_time) / 1000  # Convert to seconds

    # Increase emoji speed over time
    speed_factor = 1 + (elapsed_time / 100)  # Speed increases as the game progresses

    # Move and Draw Emojis
    for emoji in emojis:
        emoji.move(speed_factor)
        emoji.draw(screen)

    # Move and Draw Bullets
    for bullet in bullets:
        bullet.move()
        bullet.draw(screen)

    # Create lists to store objects that need to be removed
    bullets_to_remove = []
    emojis_to_remove = []

    # Check for collisions (bursting emojis)
    for bullet in bullets[:]:
        for emoji in emojis[:]:
            if emoji.x < bullet.x < emoji.x + emoji.size and emoji.y < bullet.y < emoji.y + emoji.size:
                bullets_to_remove.append(bullet)  # Mark bullet for removal
                emojis_to_remove.append(emoji)  # Mark emoji for removal
                score += emoji.get_points() # Increase score when an emoji is burst
                emoji = Emoji()  # Create an Emoji object
                emoji.emoji_description(emoji_images, is_special)  # Initialize it
                emojis.append(emoji)

        # Remove bullets and emojis outside the loop to avoid modifying the list during iteration
        for bullet in bullets_to_remove:
            if bullet in bullets:
                bullets.remove(bullet)

        for emoji in emojis_to_remove:
            if emoji in emojis:
                emojis.remove(emoji)

        # Display the score and timer
        elapsed_time = (current_time - start_time) / 1000  # Convert to seconds
        time_left = max(TIME_LIMIT - int(elapsed_time), 0)
        score_text = pygame.font.SysFont('Arial', 36).render(f"Score: {score}", True, BLACK)
        time_text = pygame.font.SysFont('Arial', 36).render(f"Time: {time_left}s", True, BLACK)
        screen.blit(score_text, (10, 10))
        screen.blit(time_text, (WIDTH - 150, 10))

        # End Game if Time is Up
        if time_left == 0:
            game_over = True  # Set game_over to True when the time is over
            pygame.mixer.music.stop()  # Stop background music


            overlay = pygame.Surface((WIDTH, HEIGHT))
            overlay.set_alpha(175)  # Set transparency level (0-255)
            overlay.fill((255, 255, 255))  # Fill with white color

            # Blit the overlay to the screen (to cover everything with transparency)
            screen.blit(overlay, (0, 0))

            game_over_text = pygame.font.SysFont('Arial', 48).render(f"Game Over! Your Final Score
is: {score}", True, BLACK)
            screen.blit(game_over_text, (WIDTH // 2 - 300, HEIGHT // 2-100))

            # Draw Restart Button
            restart_button = draw_restart_button(screen)

            # Play the game over sound
            game_over_sound.play()
        # Update the Display
        pygame.display.flip()

        # Limit the frame rate
        clock.tick(FPS)

    pygame.quit()

# Restart the game
def restart_game():
    global game_running, score, emojis, bullets, start_time, last_spawn_time, game_over  # Declare
needed variables as global
    game_running = False  # Reset the game running flag
    score = 0  # Reset score
    emojis = [Emoji() for _ in range(5)]  # Restart with some emojis
    bullets = []  # Clear bullets
    start_time = pygame.time.get_ticks()  # Reset start time
    last_spawn_time = start_time  # Reset spawn time
    game_over = False  # Reset game over state
    start_screen()  # Go back to the start screen

# Show the start screen and wait for user click
def start_screen():
    global start_button, game_running  # Declare game_running as global
    running = True
    screen = pygame.display.set_mode((WIDTH, HEIGHT))
    pygame.display.set_caption('Emoji Shooter')

    # Create start button
    start_button = draw_start_button(screen)

    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
            if event.type == pygame.MOUSEBUTTONDOWN:
                mouse_x, mouse_y = pygame.mouse.get_pos()
                if start_button.collidepoint(mouse_x, mouse_y):
                    game_running = True  # Set game_running to True once the button is clicked
                    start_game_sound.play()  # Play the start game sound
                    game_loop()  # Start the game loop
                    running = False

        pygame.display.flip()

# Initialize game_running as False before the start screen is shown
```

```
game_running = False
game_over = False
start_screen()
```

output:

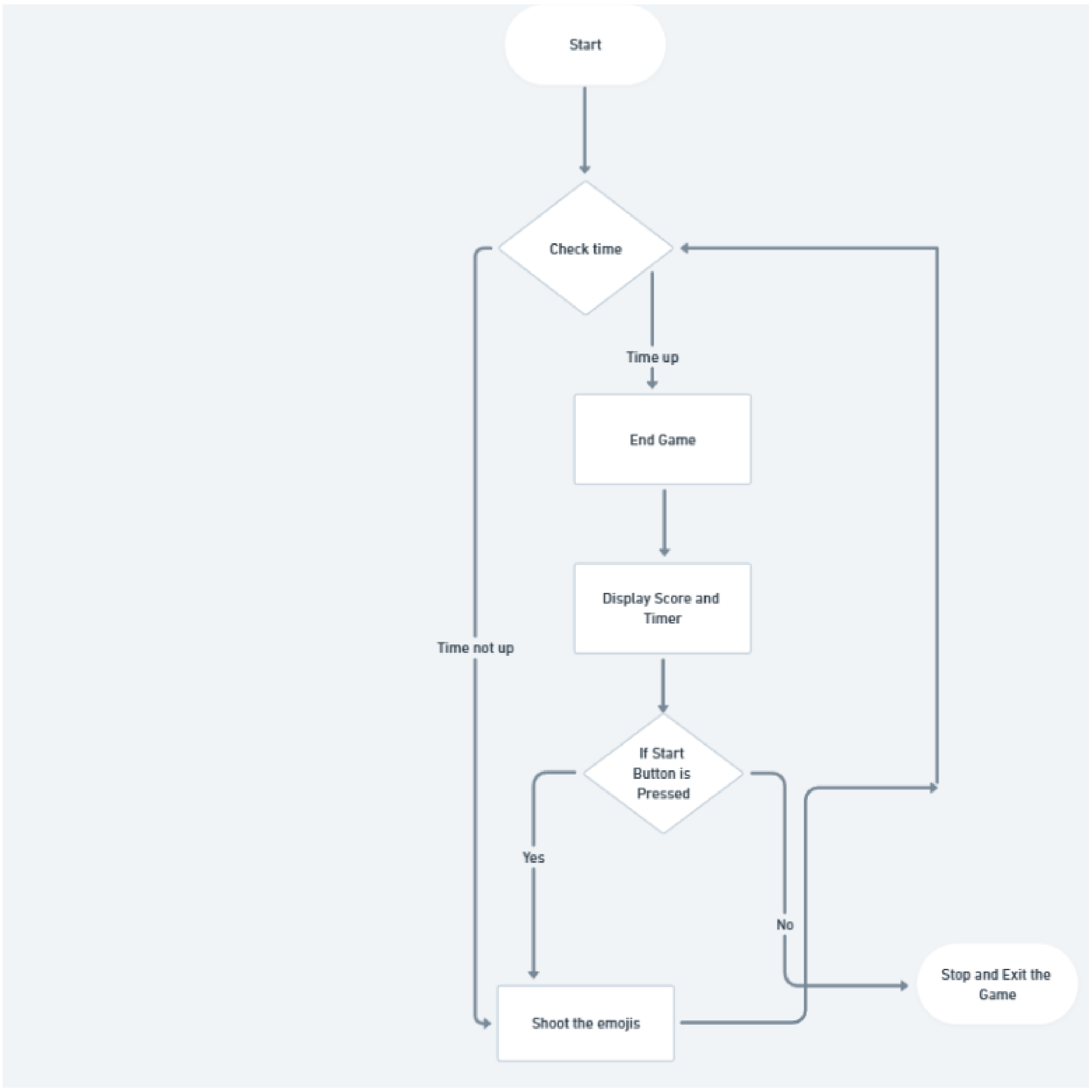






Game Over! Your Final Score is: 403

Flowchart:



5.Conclusion

The Emojis Shooter project offers an engaging game that enhances reflexes and precision while teaching Python and game development. Built with Pygame, it provides an interactive learning experience.

Its modular design allows easy expansion and customization, making it an excellent foundation for further game development exploration.