Final Project
Algorithm and Programming

Project Name: “Rock Falling Game”

Student Name: Nicholas Nixon Iswanto

Student ID: 2802546664

Class: L1CC

Table of Contents

A. Description

I. Introduction

II. The Function of the Program

B. Design

I. Sections of the Program of the Requirements

II. Function of Each Part of the Program

C. Implementation

I. Class Diagram

II. Extensibility

III. Explanation of the Functions Made and Used

D. Lessons Learned

I. Error Handling and Debugging

E. Evaluation

I. Does the Program Work Properly?

II. Future Improvements That Can Be Done

F. Evidence of Working Program

I. Testing and Pictures

A. Description

I. Introduction

The Rock Falling Game is a 2D arcade-style game developed in Python using the Pygame library. The player's goal is to avoid falling rocks while earning points for survival. It integrates elements such as images, sounds, and animations to provide an engaging experience.

II. The Function of the Program

The program offers interactive gameplay where the player navigates left and right to dodge falling obstacles. The difficulty dynamically increases as the player's score rises, adding to the challenge and excitement. The game also maintains a leaderboard of high scores for competitive play.

B. Design

I. Sections of the Program and Requirements

Sections:

1. **Main Menu**:
    - Displays game title, instructions, and options to start or quit the game.
2. **Gameplay**:
    - Includes player movement, obstacle generation, collision detection, and score tracking.
3. **Game Over Screen**:
    - Shows final score, leaderboard, and options to replay or exit.
4. **Resources**:
    - Images and sounds.

Requirements:

1. **Import pygame:**
    - Used for getting the game to run and design properly.
2. **Import random:**
    - Used to randomize the size and location rocks.
3. **Import sys:**
    - Used for handling game closing.
4. **Import os:**
    - Used for handling the saving and loading of high score.

II. Function of Each Part of the Program

1. **Player Class**:
    - Manages player position and movement.
    - Ensures the player stays within screen boundaries.
2. **Rock Class**:
    - Generates and animates falling obstacles.
    - Handles dynamic scaling and random positioning.
3. **Game Class**:
    - Combines all components, manages game logic, and oversees the main game loop.
    - Handles score updates, high score tracking, and transitions between screens.

C. Implementation

I. Class Diagram

```
# Player Class
class Player:
    def __init__(self, x, y, speed):
        self.x = x  # Player's horizontal position
        self.y = y  # Player's vertical position
        self.width = 50  # Player's width
        self.height = 50  # Player's height
        self.speed = speed  # Speed of movement

    def move(self, keys):
        # Move left if left arrow or 'A' is pressed, ensuring not to go off-screen
        if (keys[pygame.K_LEFT] or keys[pygame.K_a]) and self.x > 0:
            self.x -= self.speed
        # Move right if right arrow or 'D' is pressed, ensuring not to go off-screen
        if (keys[pygame.K_RIGHT] or keys[pygame.K_d]) and self.x < WIDTH - self.width:
            self.x += self.speed

    def draw(self):
        # Draw the player image at the current position
        screen.blit(player_image, (self.x, self.y))
```

```
    def get_rect(self):
        # Get the rectangle representation of the player for collision detection
        return pygame.Rect(self.x, self.y, self.width, self.height)

# Rock Class
class Rock:
    def __init__(self, speed):
        self.x = random.randint(0, WIDTH - 50)  # Random horizontal start position
        self.y = -50  # Start just above the screen
        self.width = random.randint(10, 100)  # Random width for variation
        self.height = random.randint(10, 100)  # Random height for variation
        self.speed = speed  # Falling speed

    def move(self):
        # Move the rock downward by its speed
        self.y += self.speed

    def draw(self):
        # Dynamically scale the rock image based on its size
        scaled_rock_image = pygame.transform.scale(rock_image, (self.width, self.height))
        screen.blit(scaled_rock_image, (self.x, self.y))

    def get_rect(self):
        # Get the rectangle representation of the rock for collision detection
        return pygame.Rect(self.x, self.y, self.width, self.height)

# Main Game Class
class Game:
    def __init__(self):
        self.player = Player(WIDTH // 2, HEIGHT - 60, 7)  # Create the player instance
        self.rocks = []  # List to hold falling rocks
        self.score = 0  # Initialize score
        self.high_scores = []  # Leaderboard of high scores
        self.running = True  # Flag to control game loop
        self.load_high_scores()  # Load saved high scores

    def load_high_scores(self):
        # Load high scores from a file, if it exists
        if os.path.exists("high_scores.txt"):
            with open("high_scores.txt", "r") as file:
                self.high_scores = [int(line.strip()) for line in file.readlines()]
        else:
            self.high_scores = []
        self.high_scores.sort(reverse=True)
```

```
    def save_high_scores(self):
        # Save top 5 high scores to a file
        with open("high_scores.txt", "w") as file:
            for score in self.high_scores[:5]:
                file.write(f"{score}\n")

    def create_rock(self):
        # Increase rock spawn rate and speed based on score
        if random.randint(1, max(20 - self.score // 100, 5)) == 1: # Increase rock spawn rate as score increases
            speed = 5 + self.score // 100  # Increase speed as score increases
            self.rocks.append(Rock(speed))

    def update_rocks(self):
        # Move each rock and remove rocks that fall below the screen
        for rock in self.rocks:
            rock.move()
        self.rocks = [rock for rock in self.rocks if rock.y < HEIGHT]

    def detect_collision(self):
        # Check for collisions between the player and any rock
        player_rect = self.player.get_rect()
        for rock in self.rocks:
            if player_rect.colliderect(rock.get_rect()):
                collision_sound.play()  # Play collision sound
                self.running = False  # End the game

    def update_score(self):
        # Increment the score over time
        self.score += 1

    def update_leaderboard(self):
        # Update high scores with the current score
        self.high_scores.append(self.score)
        self.high_scores.sort(reverse=True)
        self.high_scores = self.high_scores[:5]  # Keep only the top 5 scores
        self.save_high_scores()

    def draw_text(self, text, x, y, color=WHITE):
        # Render and display text on the screen
        label = font.render(text, True, color)
        screen.blit(label, (x, y))

    def draw_rocks(self):
        # Draw all the rocks on the screen
        for rock in self.rocks:
```

```
            rock.draw()

    def main_menu(self):
        # Display the main menu and wait for user input
        menu_running = True
        while menu_running:
            screen.blit(menu_background, (0, 0))
            self.draw_text("Rock Falling Game", WIDTH // 2 - 150, HEIGHT // 2 - 100, WHITE)
            self.draw_text("Press ENTER to Start", WIDTH // 2 - 150, HEIGHT // 2, BLUE)
            self.draw_text("Press ESC to Quit", WIDTH // 2 - 150, HEIGHT // 2 + 50, RED)
            self.draw_text("Use ARROW KEYS or A/D to Move", WIDTH // 2 - 200, HEIGHT // 2 + 100, YELLOW)
            self.draw_text("Avoid falling rocks!", WIDTH // 2 - 200, HEIGHT // 2 + 150, GREEN)
            pygame.display.flip()

            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    pygame.quit()
                    sys.exit()
                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_RETURN:
                        return  # Start the game
                    if event.key == pygame.K_ESCAPE:
                        pygame.quit()
                        sys.exit()

    def game_over_screen(self):
        # Display the game over screen with final score and leaderboard
        self.update_leaderboard()
        game_over_running = True
        while game_over_running:
            screen.blit(game_over_background, (0, 0))
            screen.blit(game_over_image, (WIDTH // 2 - 300, HEIGHT // 2 - 225))
            self.draw_text("GAME OVER :(", WIDTH // 2 - 100, HEIGHT // 2, RED)
            self.draw_text(f"Final Score: {self.score}", WIDTH // 2 - 150, HEIGHT // 2 + 50, BLUE)
            self.draw_text("Press ENTER to return to Main Menu", WIDTH // 2 - 150, HEIGHT // 2 + 100, GREEN)
            self.draw_text("Press ESC to Quit", WIDTH // 2 - 150, HEIGHT // 2 + 150, YELLOW)
            self.draw_text("Leaderboard", WIDTH - 200, 50, YELLOW)

            for i, high_score in enumerate(self.high_scores):
                self.draw_text(f"{i + 1}. {high_score}", WIDTH - 200, 100 + i * 40, WHITE)

            pygame.display.flip()

            for event in pygame.event.get():
                if event.type == pygame.QUIT:
```

```
                pygame.quit()
                sys.exit()
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_RETURN:
                    return  # Return to main menu
                if event.key == pygame.K_ESCAPE:
                    pygame.quit()
                    sys.exit()

    def play_game(self):
        # Core game loop
        self.running = True
        self.player.x, self.player.y = WIDTH // 2, HEIGHT - 60  # Reset player position
        self.rocks.clear()  # Clear existing rocks
        self.score = 0  # Reset score
        pygame.mixer.music.play()  # Play background music

        while self.running:
            screen.fill(BLACK)  # Clear screen
            screen.blit(background_image, (0, 0))

            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    pygame.quit()
                    sys.exit()

            keys = pygame.key.get_pressed()
            self.player.move(keys)  # Handle player movement

            self.create_rock()  # Spawn new rocks
            self.update_rocks()  # Move and manage rocks
            self.detect_collision()  # Check for collisions
            self.update_score()  # Update the score

            self.player.draw()  # Draw the player
            self.draw_rocks()  # Draw all rocks
            self.draw_text(f"Score: {self.score}", 10, 10)  # Display score

            pygame.display.flip()
            clock.tick(FPS)  # Maintain consistent frame rate

        pygame.mixer.music.stop()  # Stop music after game over
        self.game_over_screen()  # Show game over screen

    def run(self):
```

```
    # Main loop to display menu and play the game
    while True:
        self.main_menu()  # Show main menu
        self.play_game()  # Start game
```

II. Extensibility

The program uses object-oriented principles, making it extensible for adding new features. Each class is modular and independent, allowing for seamless integration of new functionality.

III. Explanation of the Functions Made and Used

1. **Player Class**:
    - move(keys): Updates the player's position based on user input.
    - draw(): Draws the player image at the current position.
    - get_rect(): Returns a rectangle for collision detection.
2. **Rock Class**:
    - move(): Moves the rock downward at a defined speed.
    - draw(): Draws the rock image at its current position.
    - get_rect(): Returns a rectangle for collision detection.
3. **Game Class**:
    - load_high_scores(): Load high scores from a file.
    - save_high_scores(): Save top 5 high scores to a file.
    - create_rock(): Generates new rocks at random intervals.
    - update_rocks(): Move each rock and remove rocks that fall below the screen.
    - detect_collision(): Checks for collisions between the player and rocks.
    - update_score(): Increments the score based on survival time.
    - update_leaderboard(): Update high scores with current score.
    - draw_text(): Render and display text on screen.
    - draw_rocks(): Draw all rocks on screen.
    - main_menu(): Displays the main menu and waits for user input.
    - game_over_screen(): Displays the game over screen and updates the leaderboard.
    - play_game(): Core game loop.
    - run(): Main loop to display menu and play game.

D. Lessons Learned

I. Error Handling and Debugging

I made small typos which I overlooked and took me quite some time to find them and it was pretty frustrating. I couldn't get the rocks to generate randomly and my character could go out of the screen at first which I fixed both of them later on.

E. Evaluation

I. Does the Program Work Properly?

Yes, the program does what it is supposed to do and runs smoothly. User inputs are responsive, collisions are detected, and the game can transition between the different screens.

II. Future Improvements That Can Be Done

1. **Power-Ups**: Introduce temporary boosts like invincibility.
2. **Levels**: Add thematic levels with unique backgrounds and obstacle types.

F. Evidence of Working Program

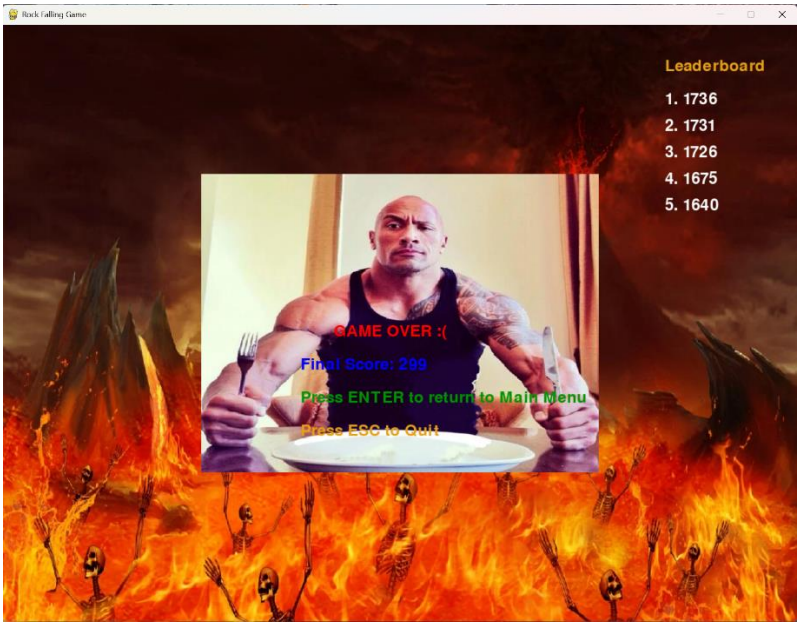
I. Testing and Pictures

1. **Testing**:
    - Verified collision detection by simulating edge cases.
    - Ensured all menu options function as intended.
    - Tested performance across different screen resolutions.
2. **Pictures**:
    - Screenshots of the main menu, gameplay, and game over screen.

Main Menu:

Gameplay:

Game Over:


Rock Falling Game
Leaderboard
1. 1736
2. 1731
3. 1726
4. 1675
5. 1640
GAME OVER :(
Final Score: 299
Press ENTER to return to Main Menu
Press ESC to Quit