LABORATORY REPORT

Programming with Python and Java Laboratory (CS 29008)

B.Tech Program in ECsc.

Submitted By

Group No. 10

Roll No.	Name
2230153	Arkajyoti Kundu
2230155	Arya Vats
2230158	Atul Rajput
2230160	Atulya Bihari Singh

Kalinga Institute of Industrial Technology (Deemed to be University) Bhubaneswar, India

Spring, 2024

Problem Statement: Create a Breakout Game in Java and Python separately.

Theory: Breakout is a classic arcade game where players control a paddle to bounce a ball, breaking bricks arranged at the top of the screen. As part of a learning exercise or a personal project, we have independently implemented the Breakout game using both Java and Python programming languages.

Code in Java:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class BreakoutGame extends JPanel implements
KeyListener, ActionListener {
    private static final int WIDTH = 900;
    private static final int HEIGHT = 600;
    private static final int PADDLE_WIDTH = 100;
    private static final int PADDLE_HEIGHT = 20;
    private static final int BALL_DIAMETER = 20;
    private static final int BRICK_WIDTH = 70;
    private static final int BRICK_HEIGHT = 20;
    private static final int PADDLE_SPEED = 15;
    private static final int BALL_SPEED = 10;
    private static final int NUM_BRICKS = 105;
    private static final int BRICK_ROWS = 6;
    private static final int BRICK_COLS = 9;

    private int paddleX = WIDTH / 2 - PADDLE_WIDTH / 2;
    private int ballX = WIDTH / 2 - BALL_DIAMETER / 2;
    private int ballY = HEIGHT - PADDLE_HEIGHT -
BALL_DIAMETER;
    private int ballDeltaX = BALL_SPEED;
    private int ballDeltaY = -BALL_SPEED;
    private int score = 0;
```

```
    private boolean gameOver = false;
    private boolean gameStarted = false;
    private String playerName = "";

    private boolean[] keysPressed = new
boolean[KeyEvent.KEY_LAST];

    private Rectangle paddle = new Rectangle(paddleX, HEIGHT
- PADDLE_HEIGHT, PADDLE_WIDTH, PADDLE_HEIGHT);
    private Rectangle ball = new Rectangle(ballX, ballY,
BALL_DIAMETER, BALL_DIAMETER);
    private Rectangle[] bricks = new Rectangle[NUM_BRICKS];

    private JButton restartButton;
    private JButton quitButton;
    private JButton startButton;
    private JLabel scoreLabel;
    private JTextField nameTextField;

    public BreakoutGame() {
        addKeyListener(this);
        setFocusable(true);
        setPreferredSize(new Dimension(WIDTH, HEIGHT));

        initializeBricks();

        startButton = new JButton("Start");
        startButton.addActionListener(this);
        add(startButton);

        restartButton = new JButton("Restart");
        restartButton.addActionListener(this);
        add(restartButton);
```

```
        quitButton = new JButton("Quit");
        quitButton.addActionListener(this);
        add(quitButton);

        scoreLabel = new JLabel("Your Score: " + score);
        scoreLabel.setFont(new Font("Sourcecodepro",
Font.BOLD, 20));
        scoreLabel.setForeground(Color.WHITE);

scoreLabel.setHorizontalAlignment(SwingConstants.CENTER);
        add(scoreLabel);

        nameTextField = new JTextField("Enter your name");
        nameTextField.addActionListener(this);
        add(nameTextField);
    }

    private void initializeBricks() {
        int gap = 5; // Set the gap between bricks
        int currentX = 0;
        int currentY = 50;

        for (int i = 0; i < NUM_BRICKS; i++) {
            bricks[i] = new Rectangle(currentX, currentY,
BRICK_WIDTH, BRICK_HEIGHT);
            currentX += BRICK_WIDTH + gap; // Add the gap to
the x-coordinate
            if (currentX >= WIDTH) {
                currentX = 0;
                currentY += BRICK_HEIGHT + gap; // Add the
gap to the y-coordinate
            }
```

```
        }
    }

    private void update() {
        if (!gameOver && gameStarted) {
            movePaddle();
            moveBall();
            checkCollisions();
            checkGameOver();
        }
    }

    private void movePaddle() {
        if (keysPressed[KeyEvent.VK_LEFT] && paddleX > 0) {
            paddleX -= PADDLE_SPEED;
        }
        if (keysPressed[KeyEvent.VK_RIGHT] && paddleX <
WIDTH - PADDLE_WIDTH) {
            paddleX += PADDLE_SPEED;
        }
        paddle.setLocation(paddleX, HEIGHT - PADDLE_HEIGHT);
    }

    private void moveBall() {
        ball.x += ballDeltaX;
        ball.y += ballDeltaY;

        if (ball.x <= 0 || ball.x >= WIDTH - BALL_DIAMETER)
{
            ballDeltaX *= -1;
        }
        if (ball.y <= 0) {
            ballDeltaY *= -1;
```

```
        }
    }

    private void checkCollisions() {
        if (paddle.intersects(ball)) {
            ballDeltaY *= -1;
        }
        for (Rectangle brick : bricks) {
            if (brick != null && brick.intersects(ball)) {
                brick.setLocation(0, 0);
                score += 5;
                ballDeltaY *= -1;
            }
        }
        scoreLabel.setText("Your Score: " + score);
    }

    private void checkGameOver() {
        if (ball.y >= HEIGHT - BALL_DIAMETER) {
            gameOver = true;
            restartButton.setEnabled(true); // Enable
restart button when the game is over
        }
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.setColor(new Color(48, 0, 74));
        g.fillRect(0, 0, WIDTH, HEIGHT);

        g.setColor(new Color(184, 51, 255));
```

```
        g.fillRect(paddleX, HEIGHT - PADDLE_HEIGHT,
PADDLE_WIDTH, PADDLE_HEIGHT);
        g.setColor(new Color(184, 51, 255)); // Change ball
color to blue
        g.fillOval(ball.x, ball.y, BALL_DIAMETER,
BALL_DIAMETER);

        g.setColor(Color.RED); // Change brick outline color
to red
        for (Rectangle brick : bricks) {
            if (brick != null) {
                g.setColor(new Color(219, 153, 255)); //
Change brick inside color to purple
                g.fillRect(brick.x, brick.y, BRICK_WIDTH,
BRICK_HEIGHT);
                g.setColor(new Color(219, 153, 255)); //
Resetting color to draw outline
                g.drawRect(brick.x, brick.y, BRICK_WIDTH,
BRICK_HEIGHT);
            }
        }

        if (gameOver) {
            g.setColor(Color.WHITE);
            g.setFont(new Font("Sourcecodepro", Font.BOLD,
30));
            g.drawString("Game Over! " + playerName + "'s
Score: " + score, WIDTH / 2 - 150, HEIGHT / 2);
        }

        g.setColor(Color.WHITE);
        g.drawString("Player: " + playerName, 10, 20);
    }
```

```
    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == restartButton) {
            restartGame();
        } else if (e.getSource() == quitButton) {
            System.exit(0);
        } else if (e.getSource() == nameTextField) {
            playerName = nameTextField.getText();
            nameTextField.setEditable(false);
            nameTextField.setFocusable(false);
        } else if (e.getSource() == startButton) {
            startGame();
        }
    }

    public void startGame() {
        gameStarted = true;
        startButton.setEnabled(false); // Disable start
button when the game starts
        requestFocus(); // Focus on the game panel
    }

    public void restartGame() {
        score = 0;
        gameOver = false;
        ball.x = paddleX + PADDLE_WIDTH / 2 - BALL_DIAMETER
/ 2; // Reset ball position onto the paddle
        ball.y = HEIGHT - PADDLE_HEIGHT - BALL_DIAMETER;
        restartButton.setEnabled(false); // Disable restart
button when game restarts
        initializeBricks();
```

```
            requestFocusInWindow(); // Request focus on the game
panel after restarting
        }

        @Override
        public void keyPressed(KeyEvent e) {
            keysPressed[e.getKeyCode()] = true;
        }

        @Override
        public void keyReleased(KeyEvent e) {
            keysPressed[e.getKeyCode()] = false;
        }

        @Override
        public void keyTyped(KeyEvent e) {
        }

        public static void main(String[] args) {
            JFrame frame = new JFrame("Breakout Game");
            BreakoutGame game = new BreakoutGame();
            frame.add(game);
            frame.pack();

frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setLocationRelativeTo(null);
            frame.setVisible(true);

            Timer timer = new Timer(1000 / 60, e -> {
                game.update();
                game.repaint();
            });
            timer.start();
```

```
    }
}
```

Code in Python:

```
import pygame
import sys
import random
from pygame.locals import *

# Constants
WIDTH = 1000
HEIGHT = 600
PADDLE_WIDTH = 100
PADDLE_HEIGHT = 10
BALL_RADIUS = 10
BRICK_WIDTH = 60
BRICK_HEIGHT = 20
BRICK_ROWS = 5
BRICK_COLS = 20
WHITE = (255, 255, 255)
BLACK = (48, 0, 74)
FPS = 60

# Initialize Pygame
pygame.init()

# Set up the window
window = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Breakout Game")
```

```
# Clock for controlling the frame rate
clock = pygame.time.Clock()

# Fonts
font = pygame.font.Font(None, 36)

# Function to draw the paddle
def draw_paddle(paddle_x):
    pygame.draw.rect(window, WHITE, (paddle_x, HEIGHT -
PADDLE_HEIGHT, PADDLE_WIDTH, PADDLE_HEIGHT))

# Function to draw the ball
def draw_ball(ball_x, ball_y):
    pygame.draw.circle(window, WHITE, (ball_x, ball_y),
BALL_RADIUS)

# Function to draw the bricks
def draw_bricks(bricks):
    border_size = 1  # Size of the border around each brick
    inner_color = (184, 51, 255)  # Color for the inner part
of the brick
    border_color = WHITE  # Color for the border

    for brick in bricks:
        pygame.draw.rect(window, border_color, brick)
        inner_brick = (brick.x + border_size, brick.y +
border_size,
                       brick.width - 2 * border_size,
brick.height - 2 * border_size)
        pygame.draw.rect(window, inner_color, inner_brick)

# Function to check collision with the paddle
```

```
def check_paddle_collision(ball_x, ball_y, paddle_x):
    if ball_y + BALL_RADIUS >= HEIGHT - PADDLE_HEIGHT and
paddle_x <= ball_x <= paddle_x + PADDLE_WIDTH:
        return True
    return False

# Function to check collision with the bricks
def check_brick_collision(ball_x, ball_y, bricks):
    for brick in bricks:
        if brick.colliderect(pygame.Rect(ball_x -
BALL_RADIUS, ball_y - BALL_RADIUS, BALL_RADIUS * 2,
BALL_RADIUS * 2)):
            bricks.remove(brick)
            return True
    return False

# Function to initialize the bricks
def init_bricks():
    bricks = []
    for row in range(BRICK_ROWS):
        for col in range(BRICK_COLS):
            brick = pygame.Rect(col * (BRICK_WIDTH + 5), 30
+ row * (BRICK_HEIGHT + 5), BRICK_WIDTH, BRICK_HEIGHT)
            bricks.append(brick)
    return bricks

# Function to display score and time
def display_info(score, time):
    score_text = font.render("Score: " + str(score), True,
WHITE)
    time_text = font.render("Time: " + str(time), True,
WHITE)
    window.blit(score_text, (10, 10))
```

```
    window.blit(time_text, (WIDTH - 150, 10))

# Function to display game over screen
def game_over_screen(score, time):
    window.fill((48, 0, 74))
    game_over_text = font.render("Game Over", True, WHITE)
    final_score_text = font.render("Score: " + str(score),
True, WHITE)
    final_time_text = font.render("Time: " + str(time),
True, WHITE)
    restart_text = font.render("Restart", True, (48, 0, 74))
    quit_text = font.render("Quit", True, (48, 0, 74))
    window.blit(game_over_text, (WIDTH // 2 - 100, HEIGHT //
2 - 50))
    window.blit(final_score_text, (WIDTH // 2 - 100, HEIGHT
// 2))
    window.blit(final_time_text, (WIDTH // 2 - 100, HEIGHT
// 2 + 50))
    pygame.draw.rect(window, WHITE, (WIDTH // 2 - 50, HEIGHT
// 2 + 100, 100, 50))
    pygame.draw.rect(window, WHITE, (WIDTH // 2 - 50, HEIGHT
// 2 + 160, 100, 50))
    window.blit(restart_text, (WIDTH // 2 - 40, HEIGHT // 2
+ 110))
    window.blit(quit_text, (WIDTH // 2 - 30, HEIGHT // 2 +
170))
    pygame.display.update()

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

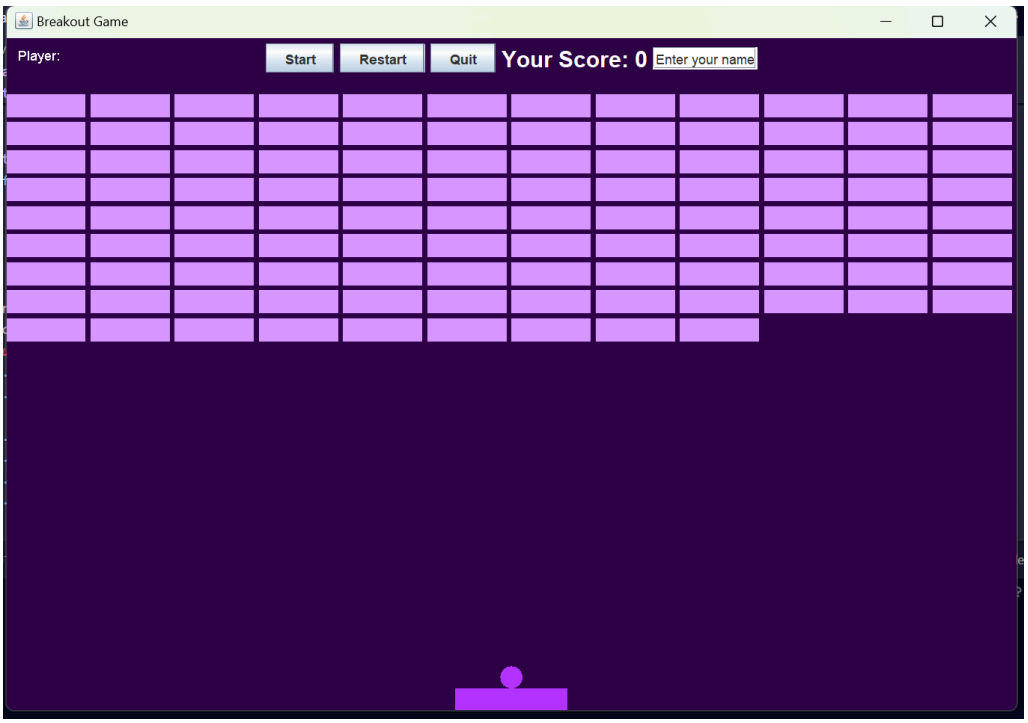
```
            elif event.type == MOUSEBUTTONDOWN:
                mouse_x, mouse_y = pygame.mouse.get_pos()
                if WIDTH // 2 - 50 <= mouse_x <= WIDTH // 2
+ 50 and HEIGHT // 2 + 100 <= mouse_y <= HEIGHT // 2 + 150:
                    main()
                elif WIDTH // 2 - 50 <= mouse_x <= WIDTH //
2 + 50 and HEIGHT // 2 + 160 <= mouse_y <= HEIGHT // 2 +
210:
                    pygame.quit()
                    sys.exit()

# Main function
def main():
    # Initialize game variables
    paddle_x = (WIDTH - PADDLE_WIDTH) // 2
    ball_x = random.randint(BALL_RADIUS, WIDTH -
BALL_RADIUS)
    ball_y = HEIGHT // 2
    ball_dx = random.choice([-5, 5])
    ball_dy = -5
    bricks = init_bricks()
    score = 0
    start_time = pygame.time.get_ticks() // 1000
    game_over = False

    # Game loop
    while True:
        window.fill((48, 0, 74))

        # Event handling
        for event in pygame.event.get():
            if event.type == QUIT:
                pygame.quit()
```

```
                sys.exit()

        # Move the paddle
        keys = pygame.key.get_pressed()
        if keys[K_LEFT] and paddle_x > 0:
            paddle_x -= 5
        if keys[K_RIGHT] and paddle_x < WIDTH -
PADDLE_WIDTH:
            paddle_x += 5

        # Move the ball
        ball_x += ball_dx
        ball_y += ball_dy

        # Check collision with walls
        if ball_x - BALL_RADIUS <= 0 or ball_x + BALL_RADIUS
>= WIDTH:
            ball_dx *= -1
        if ball_y - BALL_RADIUS <= 0:
            ball_dy *= -1

        # Check collision with paddle
        if check_paddle_collision(ball_x, ball_y, paddle_x):
            ball_dy *= -1

        # Check collision with bricks
        if check_brick_collision(ball_x, ball_y, bricks):
            score += 5
            ball_dy *= -1

        # Check if the ball missed the paddle
        if ball_y + BALL_RADIUS >= HEIGHT:
            game_over = True
```

```
        # Draw elements
        draw_paddle(paddle_x)
        draw_ball(ball_x, ball_y)
        draw_bricks(bricks)
        display_info(score, pygame.time.get_ticks() // 1000
- start_time)

        # Update the display
        pygame.display.update()
        clock.tick(FPS)

        # Game over condition
        if game_over:
            final_score = score
            final_time = pygame.time.get_ticks() // 1000 -
start_time
            game_over_screen(final_score, final_time)

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

Output:

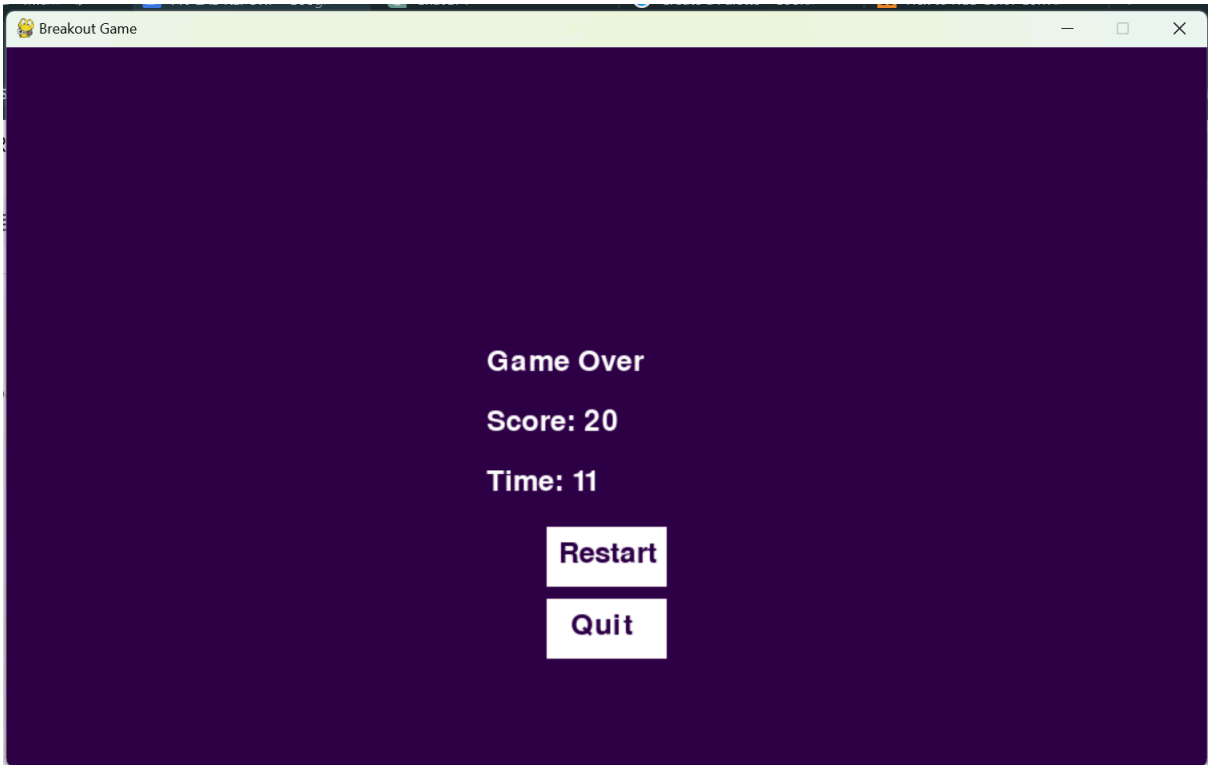
Screenshot of Breakout Game using Java before starting the game:



Screenshot of Breakout Game using Java before after playing the game:

Screenshot of Breakout Game using Python before starting the game:

Screenshot of Breakout Game using Python after playing the game:



Application:

The Breakout game, implemented in both Java and Python, has several potential applications and benefits:

Educational Tool: The Breakout game can serve as an educational tool for teaching programming concepts, especially to beginners. By examining and comparing the Java and Python implementations, learners can gain insights into language syntax, data structures, algorithms, and game development techniques.

Language Comparison: Comparing the Java and Python implementations provides an opportunity to understand the differences and similarities between the two programming languages. This comparison can help developers make informed decisions about language selection for future projects based on factors such as ease of use, performance, and ecosystem support.

Software Development Practices: Analysing the code structure, design patterns, and implementation choices in both versions of the game offers valuable lessons in software development practices. Developers can learn from best practices demonstrated in the implementations and identify areas for improvement or optimization.

Performance Optimization: Evaluating the performance metrics of the Java and Python implementations can lead to insights into performance optimization strategies. Developers can learn how to optimise code, improve algorithm efficiency, and minimise resource usage to enhance the performance of their applications.

Entertainment and Recreation: Beyond educational and learning purposes, the Breakout game provides entertainment and recreation for players of all ages. It offers a nostalgic gaming experience for those familiar with classic arcade games while engaging new players with its simple yet addictive gameplay.

Conclusion:

By conducting a thorough evaluation of the Java and Python implementations of the Breakout game, this project aims to provide valuable insights into the

strengths and weaknesses of each version. Additionally, it offers opportunities for code improvement, optimization, and knowledge transfer between the two programming languages. Ultimately, this project contributes to the developer's understanding of game development principles and the nuances of programming languages in real-world applications.