Report On

Title of the Course Project

Submitted in partial fulfillment of the requirements of the Course project in
Semester III of Second Year Artificial Intelligence and Data Science

by
Afnan Pathan (Roll No. 39)
Ketan Mahadik (Roll No. 24)
Om Patil (Roll No. 43)

Supervisor
Prof. Sneha Yadav

University of Mumbai

Vidyavardhini's College of Engineering & Technology

Department of Artificial Intelligence and Data Science

(2023-24)

Vidyavardhini's College of Engineering & Technology
Department of Artificial Intelligence and Data Science

CERTIFICATE

This is to certify that the project entitled “Snake Game” is a bonafide work of "Afnan Pathan (Roll No. 39), Ketan Mahadik (Roll No. 24), Om Patil (Roll No. 43), submitted to the University of Mumbai in partial fulfillment of the requirement for the Course project in semester III of Second Year Artificial Intelligence and Data Science engineering.

Supervisor

Prof. Sneha Yadav

Dr. Tatwadarshi P. N.
Head of Department

Abstract

The Snake Game is a classic arcade-style video game that has been popular for decades. In this report, we will explore the development and implementation of a Snake Game using Java. The game involves controlling a snake, which grows longer as it consumes food while avoiding collisions with walls and itself. This report provides an overview of the game, its features, the technology used, and the challenges encountered during development.

Table of Contents

Pg. No

Problem Statement

The project's primary objective is to develop and implement a Snake Game using the Java programming language. The Snake Game is a classic and iconic video game, known for its simple yet engaging gameplay. In this project, we aim to recreate the experience of controlling a snake within the confines of a game board, where the snake's primary goal is to consume food items to grow longer.

Block Diagram

```
+------------------------+
|    Snake Game          |
+------------------------+
|  Game Initialization  |
|------------------------|
|  Game Loop             |
|------------------------|
|  User Input Handling  |
|------------------------|
|  Collision Detection  |
|------------------------|
|  Score Tracking        |
|------------------------|
|  Game Over Handling   |
+------------------------+
```

Module Description

Game Initialization Module:

- Description: The Game Initialization module is responsible for setting up the initial state of the game.
- Working: This module includes tasks such as creating the game board, initializing the snake's starting position, placing the first food item on the board, and initializing game variables. It establishes the game's initial conditions for the player.

User Input Handling Module:

- Description: The User Input Handling module captures and processes user input, allowing the player to control the snake's direction.
- Working: It listens for user input, typically through keyboard key presses, and interprets these inputs to adjust the snake's direction as the player desires. The direction input is then utilized by the game loop for the snake's movement.

Game Over Handling Module:

- Description: The Game Over Handling module is activated when the game concludes due to a collision event.
- Working: This module displays the player's final score on the screen and provides the option to restart the game. It handles the end of the game and allows the player to make a new attempt.

Brief Description

1. Java Programming Language:

   Description: Java is the primary programming language used for developing the Snake Game. It is known for its platform independence, making the game accessible on various operating systems.

2. Game Loop:

   Description: The game loop is a fundamental software component responsible for maintaining the real-time behavior of the game. It continuously updates the game's state, including the snake's movement, collision detection, and score tracking.

3. User Input Handling:

   Description: Software components for capturing and processing user input are essential for enabling player control. The game uses Java's input handling mechanisms to listen for keyboard inputs and translate them into snake direction changes.

3. Collision Detection Algorithms:

   Description: The software includes algorithms to accurately detect collisions within the game. These algorithms identify when the snake collides with the walls of the game board or itself. Proper collision detection is critical to enforce game rules and determine game over conditions.

4. Game Over Handling:

   Description: The game over handling software component is responsible for displaying the player's final score when the game ends due to a collision event. It also provides the option to restart the game.

Code

```
import javax.swing.*;

import java.awt.*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.KeyEvent;

import java.awt.event.KeyListener;

import java.util.ArrayList;

import java.util.Random;


public class SnakeGameGUI extends JPanel implements ActionListener, KeyListener {

    private static final int CELL_SIZE = 30;

    private static final int BOARD_WIDTH = 25;

    private static final int BOARD_HEIGHT = 25;

    private static final int DELAY = 400;

    private static final int INITIAL_SNAKE_LENGTH = 3;


    private ArrayList<Point> snake;

    private Point food;

    private char[][] board;

    private int direction;
private int score;
private boolean gameStarted;

private JButton startButton;
private JButton upButton;
private JButton leftButton;
private JButton downButton;
private JButton rightButton;

public SnakeGameGUI() {
    snake = new ArrayList<>();
    initializeBoard();
    initializeSnake();
    food = generateFood();
    direction = 1;
    score = 0;
    gameStarted = false;

    startButton = new JButton("Start");
    startButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            startGame();
        }
    });
```

```
upButton = new JButton("Up");
upButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        setDirection(0); // Up
    }
});

leftButton = new JButton("Left");
leftButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        setDirection(3); // Left
    }
});

downButton = new JButton("Down");
downButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        setDirection(2); // Down
    }
});

rightButton = new JButton("Right");
rightButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        setDirection(1); // Right
```

```
            }
        });

        this.add(startButton);
        this.add(upButton);
        this.add(leftButton);
        this.add(downButton);
        this.add(rightButton);

        Timer timer = new Timer(DELAY, this);
        timer.start();

        setPreferredSize(new Dimension(BOARD_WIDTH * CELL_SIZE,
BOARD_HEIGHT * CELL_SIZE));
        setFocusable(true);
        addKeyListener(this);
    }

    public void keyTyped(KeyEvent e) {
    }

    public void keyPressed(KeyEvent e) {
        char key = e.getKeyChar();
        if (gameStarted) {
            switch (key) {
```

```
            case 'w':
                setDirection(0); // Up
                break;
            case 'a':
                setDirection(3); // Left
                break;
            case 's':
                setDirection(2); // Down
                break;
            case 'd':
                setDirection(1); // Right
                break;
            case '\n':
                startGame();
                break;
        }
    }
}

public void keyReleased(KeyEvent e) {
}

private void initializeBoard() {
    board = new char[BOARD_HEIGHT][BOARD_WIDTH];
    for (int i = 0; i < BOARD_HEIGHT; i++) {
        for (int j = 0; j < BOARD_WIDTH; j++) {
            board[i][j] = 0;
        }
    }
}

private void initializeSnake() {
    for (int i = 0; i < INITIAL_SNAKE_LENGTH; i++) {
        snake.add(new Point(BOARD_WIDTH / 2 - i, BOARD_HEIGHT / 2));
    }
}

private Point generateFood() {
    Random random = new Random();
    int x, y;
    do {
        x = random.nextInt(BOARD_WIDTH);
        y = random.nextInt(BOARD_HEIGHT);
    } while (board[y][x] != 0 || snake.contains(new Point(x, y)));

    return new Point(x, y);
}

protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    drawBoard(g);

    drawFood(g);

    drawSnake(g);

}


private void drawBoard(Graphics g) {

    for (int y = 0; y < BOARD_HEIGHT; y++) {

        for (int x = 0; x < BOARD_WIDTH; x++) {

            g.setColor(Color.WHITE);

            g.fillRect(x * CELL_SIZE, y * CELL_SIZE, CELL_SIZE, CELL_SIZE);

            g.setColor(Color.BLACK);

            g.drawRect(x * CELL_SIZE, y * CELL_SIZE, CELL_SIZE, CELL_SIZE);

        }

    }

}


private void drawFood(Graphics g) {

    g.setColor(Color.RED);

    int x = food.x * CELL_SIZE;

    int y = food.y * CELL_SIZE;

    g.fillRect(x, y, CELL_SIZE, CELL_SIZE);


    g.setColor(Color.WHITE);

    g.setFont(new Font("Arial", Font.PLAIN, 12));

    String pointsString = Integer.toString(score);
```

```
        int pointsStringWidth = g.getFontMetrics().stringWidth(pointsString);
        g.drawString(pointsString, x + CELL_SIZE - pointsStringWidth - 2, y +
CELL_SIZE - 2);
    }

    private void drawSnake(Graphics g) {
        g.setColor(Color.GREEN);
        for (Point point : snake) {
            int x = point.x * CELL_SIZE;
            int y = point.y * CELL_SIZE;
            g.fillRect(x, y, CELL_SIZE, CELL_SIZE);
        }
    }

    public void actionPerformed(ActionEvent e) {
        if (gameStarted) {
            moveSnake();
            checkCollision();
            repaint();
        }
    }

    private void startGame() {
        gameStarted = true;
    }
```

```
}

private void moveSnake() {
    Point head = snake.get(0);
    Point newHead = new Point(head.x, head.y);


    switch (direction) {
        case 0: // Up
            newHead.y--;
            break;
        case 1: // Right
            newHead.x++;
            break;
        case 2: // Down
            newHead.y++;
            break;
        case 3: // Left
            newHead.x--;
            break;
    }
    if (newHead.equals(food)) {
        food = generateFood();
        score++;
    } else {
        snake.remove(snake.size() - 1);
    }
```

```
        snake.add(0, newHead);
    }


    private void checkCollision() {
        Point head = snake.get(0);


        if (head.x < 0 || head.x >= BOARD_WIDTH || head.y < 0 || head.y >=
BOARD_HEIGHT) {
            gameOver();
            return;
        }
        for (int i = 1; i < snake.size(); i++) {
            if (head.equals(snake.get(i))) {
                gameOver();
                return;
            }
        }
    }
    private void gameOver() {
        JOptionPane.showMessageDialog(this, "Game Over. Final Score: " + score);
        System.exit(0);
    }


    private void setDirection(int newDirection) {
        if (Math.abs(newDirection - direction) != 2) {
            direction = newDirection;
        }
    }


    private class Point {
        int x, y;


        Point(int x, int y) {
            this.x = x;
            this.y = y;
        }
    }
    public static void main(String[] args) {
        JFrame frame = new JFrame("Snake Game");
        SnakeGameGUI snakeGameGUI = new SnakeGameGUI();
        frame.add(snakeGameGUI);
        frame.pack();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(BOARD_WIDTH * CELL_SIZE, BOARD_HEIGHT *
CELL_SIZE);
        frame.setLocationRelativeTo(null);
        frame.setVisible(true);
    }
}
```

Results and Conclusion

The development of the Snake Game in Java has been a successful and insightful endeavor. This project allowed us to create a classic and enjoyable gaming experience while gaining valuable knowledge and experience in game development and Java programming. the Snake Game in Java offers a valuable introduction to game development, demonstrating the synergy between software components, modular design, and user engagement.

References

For book/article

[1] Q. S. Wang, “Active buckling control of beams using piezoelectric actuators and strain

gauge sensors Smart Materials Structures”, IOP Publishing, 2010. pp 065022-065030.

[2] R. R. Craig, and A. J. Kurdila, “Fundamentals of Structural Dynamics”, John Wiley & Sons, Second edition, 2006. pp 179.

[3] H. M. Ma, X. L. Gao, and J. N. Reddy, “A microstructure-dependent Timoshenko beam

model based on a modi_ed couple stress theory”, Journal of the Mechanics and Physics of Solids, 56, 2008. pp 3379-3391.

[4] G. Nishida, and M. Yamakita, :Distributed Port Hamiltonian formulation of Flexible

beams under Large Deformation”, IEEE Conference on Control Applications, Toronto, Canada, 2005. pp 589-594.

For web reference

1. Q. S. Wang, “Active buckling control of beams using piezoelectric actuators”. url. Last Accessed: 16th October, 2023.