**Internship Project Report**

**Name :** Hariom Vishwakarma

**Email ID :** [hariomvishwakarma115@gmail.com](mailto:hariomvishwakarma115@gmail.com)

**Task Start Date** : 20/02/2024

**Task Submition** **Date** : 26/02/2024

**Task Title** : Java Development Internship Project

**Task Description :** The task involved developing a classic Snake game using Java and Swing.

**Github link :** <https://github.com/Hariom115/Snake-game>

**Contant**

1. **Steps Taken**
2. **Challenges Faced**
3. **Solutions Implemented**
4. **Learnings**
5. **Project Update**
6. **Reporting Benefits**
7. **Source Code**
8. **Resorce**

**Steps Taken :**

1. **Understanding Requirements :** Initially, I thoroughly analyzed the requirements provided for the internship project, which included developing a classic Snake game using Java.

2. **Setting Up Development Environment :** I set up my development environment with Java JDK and integrated development environment (IDE).

3. **Designing the Game Board :** I started by designing the game board and its components, including the snake, apple, and background.

4. **Implementing Game Logic :** Next, I implemented the game logic, including snake movement, apple generation, collision detection, and game over conditions.

5. **Testing and Debugging :** I extensively tested the game for various scenarios, including boundary conditions, collision detection, and smooth movement.

6. **Refactoring and Optimization :** After testing, I refactored the code for better readability and optimized performance wherever necessary.

7. **Finalization :** Finally, I completed the project by adding finishing touches, such as adding comments for better code documentation and ensuring adherence to coding standards.

**Challenges Faced :**

1. **Collision Detection :** Implementing collision detection logic posed a challenge, especially in scenarios involving the snake colliding with itself or the boundaries.

2. **Game Over Conditions :** Defining precise conditions for triggering a game over was crucial, and it required careful consideration to ensure accuracy.

3. **Smooth Movement :** Ensuring smooth and responsive movement of the snake without any lag or flickering was another challenge that required optimization.

**Solutions Implemented :**

1. **Collision Detection Algorithm :** I implemented a robust collision detection algorithm that accurately detects collisions between the snake, apples, and boundaries.

2. **Refined Game Over Logic :** By refining the game over logic, I ensured that the game ends gracefully when the snake collides with itself or the boundaries.

3. **Optimized Rendering :** I optimized the rendering process to ensure smooth movement of the snake on the game board, enhancing the overall gaming experience.

**Learnings :**

- This internship project provided me with valuable hands-on experience in Java programming, especially in game development using Java Swing.

- I gained insights into various game development concepts, including game loop, collision detection, and user input handling.

- Debugging and testing the game helped me enhance my problem-solving skills and attention to detail.

**Project Update :**

- The Snake game project is successfully completed and fully functional.

- The game features smooth gameplay, accurate collision detection, and intuitive controls.

- Further enhancements and additions can be made in the future, such as adding levels, scores, and sound effects.

**Reporting Benefits :**

- Regularly updating the project progress through reports enhances transparency and facilitates effective communication.

- Reporting provides an opportunity to showcase accomplishments, challenges faced, and solutions implemented to the technical team, fostering a collaborative environment.

**Source Code :**

**File 1 :** Snake.java

package game;

import java.awt.EventQueue;

import javax.swing.JFrame;

public class Snake extends JFrame {

    public Snake() {

        initUI();

    }

    private void initUI() {

        add(new Board());

        setResizable(false);

        pack();

        setTitle("Snake");

        setLocationRelativeTo(null);

        setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

    }

    public static void main(String[] args) {

        EventQueue.invokeLater(() -> {

            JFrame ex = new Snake();

            ex.setVisible(true);

        });

    }

}

**File 2 :** Board.java

package game;

import java.awt.Color;

import java.awt.Dimension;

import java.awt.Font;

import java.awt.FontMetrics;

import java.awt.Graphics;

import java.awt.Image;

import java.awt.Toolkit;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.KeyAdapter;

import java.awt.event.KeyEvent;

import javax.swing.ImageIcon;

import javax.swing.JPanel;

import javax.swing.Timer;

public class Board extends JPanel implements ActionListener {

    private final int B\_WIDTH = 300;

    private final int B\_HEIGHT = 300;

    private final int DOT\_SIZE = 10;

    private final int ALL\_DOTS = 900;

    private final int RAND\_POS = 29;

    private final int DELAY = 140;

    private final int x[] = new int[ALL\_DOTS];

    private final int y[] = new int[ALL\_DOTS];

    private int dots;

    private int apple\_x;

    private int apple\_y;

    private boolean leftDirection = false;

    private boolean rightDirection = true;

    private boolean upDirection = false;

    private boolean downDirection = false;

    private boolean inGame = true;

    private Timer timer;

    private Image ball;

    private Image apple;

    private Image head;

    public Board() {

        initBoard();

    }

    private void initBoard() {

        addKeyListener(new TAdapter());

        setBackground(Color.black);

        setFocusable(true);

        setPreferredSize(new Dimension(B\_WIDTH, B\_HEIGHT));

        loadImages();

        initGame();

    }

    private void loadImages() {

        ImageIcon iid = new ImageIcon("src/resources/dot.png");

        ball = iid.getImage();

        ImageIcon iia = new ImageIcon("src/resources/apple.png");

        apple = iia.getImage();

        ImageIcon iih = new ImageIcon("src/resources/head.png");

        head = iih.getImage();

    }

    private void initGame() {

        dots = 3;

        for (int z = 0; z < dots; z++) {

            x[z] = 50 - z \* 10;

            y[z] = 50;

        }

        locateApple();

        timer = new Timer(DELAY, this);

        timer.start();

    }

    @Override

    public void paintComponent(Graphics g) {

        super.paintComponent(g);

        doDrawing(g);

    }

    private void doDrawing(Graphics g) {

        if (inGame) {

            g.drawImage(apple, apple\_x, apple\_y, this);

            for (int z = 0; z < dots; z++) {

                if (z == 0) {

                    g.drawImage(head, x[z], y[z], this);

                } else {

                    g.drawImage(ball, x[z], y[z], this);

                }

            }

            Toolkit.getDefaultToolkit().sync();

        } else {

            gameOver(g);

        }

    }

    private void gameOver(Graphics g) {

        String msg = "Game Over";

        Font small = new Font("Helvetica", Font.BOLD, 14);

        FontMetrics metr = getFontMetrics(small);

        g.setColor(Color.white);

        g.setFont(small);

        g.drawString(msg, (B\_WIDTH - metr.stringWidth(msg)) / 2, B\_HEIGHT / 2);

    }

    private void checkApple() {

        if ((x[0] == apple\_x) && (y[0] == apple\_y)) {

            dots++;

            locateApple();

        }

    }

    private void move() {

        for (int z = dots; z > 0; z--) {

            x[z] = x[(z - 1)];

            y[z] = y[(z - 1)];

        }

        if (leftDirection) {

            x[0] -= DOT\_SIZE;

        }

        if (rightDirection) {

            x[0] += DOT\_SIZE;

        }

        if (upDirection) {

            y[0] -= DOT\_SIZE;

        }

        if (downDirection) {

            y[0] += DOT\_SIZE;

        }

    }

    private void checkCollision() {

        for (int z = dots; z > 0; z--) {

            if ((z > 4) && (x[0] == x[z]) && (y[0] == y[z])) {

                inGame = false;

            }

        }

        if (y[0] >= B\_HEIGHT) {

            inGame = false;

        }

        if (y[0] < 0) {

            inGame = false;

        }

        if (x[0] >= B\_WIDTH) {

            inGame = false;

        }

        if (x[0] < 0) {

            inGame = false;

        }

        if (!inGame) {

            timer.stop();

        }

    }

    private void locateApple() {

        int r = (int) (Math.random() \* RAND\_POS);

        apple\_x = ((r \* DOT\_SIZE));

        r = (int) (Math.random() \* RAND\_POS);

        apple\_y = ((r \* DOT\_SIZE));

    }

    @Override

    public void actionPerformed(ActionEvent e) {

        if (inGame) {

            checkApple();

            checkCollision();

            move();

        }

        repaint();

    }

    private class TAdapter extends KeyAdapter {

        @Override

        public void keyPressed(KeyEvent e) {

            int key = e.getKeyCode();

            if ((key == KeyEvent.VK\_LEFT) && (!rightDirection)) {

                leftDirection = true;

                upDirection = false;

                downDirection = false;

            }

            if ((key == KeyEvent.VK\_RIGHT) && (!leftDirection)) {

                rightDirection = true;

                upDirection = false;

                downDirection = false;

            }

            if ((key == KeyEvent.VK\_UP) && (!downDirection)) {

                upDirection = true;

                rightDirection = false;

                leftDirection = false;

            }

            if ((key == KeyEvent.VK\_DOWN) && (!upDirection)) {

                downDirection = true;

                rightDirection = false;

                leftDirection = false;

            }

        }

    }

}

**Resource :**

**dot.png **

**apple.png**

****

**head.png**



Project-run-status.png

