SNAKE GAME

A MINI PROJECT REPORT

Submitted by

KALEES PANDI T 230701135

In partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)

THANDALAM

CHENNAI-602105

2024 – 25

BONAFIDE CERTIFICATE

Certified that this project report "**SNAKE GAME**" is the bonafide work of

"KALEES PANDI T (230701135)"

who carried out the project work under my supervision.

Submitted for the Practical Examination held on _____

SIGNATURE

Ms. Dharshini B S

Assistant Professor,

Computer Science and Engineering,

Rajalakshmi Engineering College (Autonomous),

Thandalam, Chennai-602 105

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ABSTRACT

The **Snake Game** is a classic arcade-style game that challenges players to control a growing snake in a confined space while avoiding collisions. The primary objective of the game is to guide the snake to consume food items, which increases its length and score, while preventing it from crashing into walls or its own body. The game is widely recognized for its simplicity, engaging gameplay, and progressively challenging mechanics as the snake grows longer.

This project involves designing and implementing the Snake Game using modern programming tools and techniques. It showcases concepts such as event handling, graphical user interfaces, and real-time updates. The game is built to be visually appealing and user-friendly, featuring a responsive interface and customizable settings, such as grid size and game speed.

The development of the Snake Game demonstrates fundamental programming principles, including modular design, algorithm implementation, and state management. Additionally, the project emphasizes scalability, allowing for future enhancements like multiplayer modes, power-ups, or AI-controlled snakes. Overall, the Snake Game project provides an excellent platform for exploring game development while offering a nostalgic and entertaining experience.

TABLE OF CONTENTS

CHAPTER 1- INTRODUCTION

1.1 INTRODUCTION

The Snake Game project is a classic arcade-style game that serves as an engaging example of programming concepts and game development principles. This project involves creating a simple, yet entertaining game where a player controls a snake that grows longer as it consumes food, avoiding collisions with the walls and its own body.

1.2 OBJECTIVES

The main objective of the game is to score as high as possible by consuming the food items while maneuvering the snake carefully. This project demonstrates various fundamental programming skills, including:

- **Object-Oriented Programming (OOP):** The project is structured using classes and objects to represent the snake, game board, and other components.
- **Graphics and User Interface (UI):** It uses a graphical interface to display the game elements like the snake, food, and score.
- **Game Mechanics:** Implements essential game features like movement control, collision detection, and scoring system.

The Snake Game project provides a fun and practical way to explore programming, making it an excellent learning tool for beginners and an interesting challenge for advanced programmers. This report will document the project's structure, features, and source code, giving a comprehensive overview of its functionality and design.

1.3 MODULES:

The project consists of two main modules: the **Board** and **SnakeGame**. The **Board** module manages the game state, including the snake's position, food placement, collision detection, and rendering. It updates the board based on the snake's movements and handles events like eating food or hitting a wall. The **SnakeGame** module acts as the controller, initializing the game, processing user input, managing the game loop, and coordinating actions between the snake and the board. Together, these modules implement the core functionality of the Snake game, ensuring smooth gameplay and responsive controls.

CHAPTER 2-SURVEY OF TECHNOLOGIES

2.1 PROGRAMMING LANGUAGE

The project utilizes the following technologies and tools:

Programming Language: **Java**: The source files (SnakeGame.java and Board.java) indicate that the game is implemented in Java, leveraging its object-oriented capabilities.

Development Environment: **NetBeans**: The presence of an nbproject directory with build scripts and configuration files suggests that NetBeans was used as the Integrated Development Environment (IDE) for developing this project.

Build and Deployment:

1. **Ant Build System**: The build.xml file indicates that Apache Ant is used for building and automating the project's compilation and packaging process.
2. **JAR Packaging**: The project includes a compiled JAR file (SnakeGame.jar), making it portable and executable on any system with a Java Runtime Environment (JRE).

Graphics and Resources:

1. **PNG Icons**: The icons directory includes .png files (apple.png, dot.png, head.png) used to visually represent game elements such as the apple and snake parts.

Game Framework Design:

1. The project appears to follow a modular structure, with distinct components for the game board and overall game logic. This separation aligns with best practices in software design for maintainability and scalability.

This survey indicates a lightweight but robust technology stack tailored for small-scale game development in Java.

2.2 Usage & Advantages

Usage

The Snake Game project is a Java-based implementation of the classic Snake game that can be used for entertainment or as a learning resource for understanding basic game development concepts. It is designed for execution in a Java-supported environment, and

the packaged JAR file allows easy distribution and playability. Developers can use the source code as a template or foundation to expand the game, adding features such as levels, enhanced graphics, or multiplayer modes.

Advantages

Portability: The packaged JAR file ensures that the game can run on any system with a Java Runtime Environment (JRE), making it platform-independent.

Educational Value: The project demonstrates core programming concepts like object-oriented design, event handling, and graphical rendering, serving as a learning tool for beginners.

Modular Design: The separation into modules like `Board` and `SnakeGame` enhances maintainability, making it easier to modify or extend specific functionalities.

Lightweight: The game has minimal resource requirements and is straightforward to run, making it accessible on most systems.

Customizability: With access to the source code, users can customize the game by changing graphics, gameplay mechanics, or adding new features.

Interactive Gameplay: The game provides a fun and engaging experience, making it suitable for casual gaming or as a small addition to larger projects.

This combination of usability and adaptability makes the Snake Game project both a practical application and a useful resource for developers and hobbyists alike.

CHAPTER 3- REQUIREMENTS AND ANALYSIS

3.1 Software Requirements:

1. **Java Runtime Environment (JRE)**: The game requires a JRE to run the packaged JAR file.
2. **Java Development Kit (JDK)** (Optional): Needed if users or developers intend to compile or modify the source code.
3. **NetBeans IDE** (Optional): Recommended for development and debugging, as the project contains NetBeans-specific configuration files.
4. **Apache Ant**: Used to build the project via the provided build.xml file.

3.2 Hardware Requirements:

1. A basic system with minimal CPU and memory requirements (sufficient for running Java applications).
2. Support for basic graphics rendering, as the game uses 2D images for gameplay elements.

Additional Assets:

1. The project requires the .png images located in the icons directory for visual representation of game elements like the snake's head, body, and apples.

Analysis

3.3 Functional Requirements:

1. The game should allow users to control a snake that moves on a grid-like board.
2. The snake must grow when it consumes apples, and the game should end if the snake collides with itself or the board boundaries.
3. The game should update the score dynamically based on the number of apples consumed.

3.4 Non-Functional Requirements:

1. **Performance**: The game should execute smoothly with low latency for input handling and rendering.
2. **Portability**: The use of Java ensures cross-platform compatibility, as the game can run on any operating system with a compatible JRE.
3. **Ease of Use**: Simple controls and gameplay ensure accessibility for users of all ages.

Strengths:

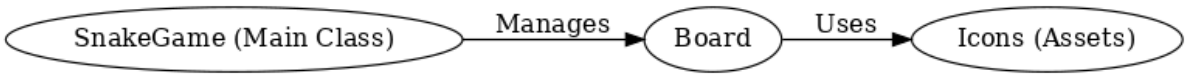
1. **Simplicity**: The project's modular structure ensures straightforward comprehension and modification.
2. **Robustness**: Java's strong exception handling and cross-platform nature enhance reliability and usability.

Potential Areas for Improvement:

1. **Graphics**: The game uses basic icons and could be enhanced with more sophisticated visuals or animations.
2. **Features**: Additional features such as levels, obstacles, or multiplayer mode could be incorporated to improve replay value.
3. **Scalability**: The current design is suitable for small projects but may need optimization for complex features or larger player bases.

In summary, this project provides a foundational implementation of a Snake game with clear requirements and a functional design. It is well-suited for both casual play and educational purposes but offers room for enhancement in graphics and gameplay features.

3.5 ER DIAGRAM



CHAPTER 4 – PROGRAM CODE

4.1 Coding

JAVA PROJECT

SNAKE GAME CODING

```
package snakegame;

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

public class Board extends JPanel implements ActionListener {

    private Image apple;
    private Image dot;
    private Image head;

    private final int ALL_DOTS = 900;
    private final int DOT_SIZE = 10;
    private final int RANDOM_POSITION = 29;

    private int apple_x;
    private int apple_y;

    private final int x[] = new int[ALL_DOTS];
    private final int y[] = new int[ALL_DOTS];

    private boolean leftDirection = false;
```

```
    private boolean rightDirection = true;
    private boolean upDirection = false;
    private boolean downDirection = false;

    private boolean inGame = true;

    private int dots;
    private Timer timer;

    Board() {                                           //Designing the Board
        addKeyListener(new TAdapter());

        setBackground(Color.BLACK);
        setPreferredSize(new Dimension(300, 300));
        setFocusable(true);

        loadImages();
        initGame();
    }

    public void loadImages() {                          //Loading Images
        ImageIcon i1 = new
ImageIcon(ClassLoader.getSystemResource("snakegame/icons/apple.png"));
        apple = i1.getImage();

        ImageIcon i2 = new
ImageIcon(ClassLoader.getSystemResource("snakegame/icons/dot.png"));
        dot = i2.getImage();
```

```
        ImageIcon i3 = new
ImageIcon(ClassLoader.getSystemResource("snakegame/icons/head.png"));
        head = i3.getImage();
    }

    public void initGame() {                                    //Initializing the Game
        dots = 3;

        for (int i = 0; i < dots; i++) {
            y[i] = 50;
            x[i] = 50 - i * DOT_SIZE;
        }

        locateApple();

        timer = new Timer(140, this);
        timer.start();
    }

    public void locateApple() {                                 //Locating Apple
        int r = (int)(Math.random() * RANDOM_POSITION);
        apple_x = r * DOT_SIZE;

        r = (int)(Math.random() * RANDOM_POSITION);
        apple_y = r * DOT_SIZE;
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);
```

```
    draw(g);
}

public void draw(Graphics g) {
    if (inGame) {
        g.drawImage(apple, apple_x, apple_y, this);

        for (int i = 0 ; i < dots; i++) {
            if (i == 0) {
                g.drawImage(head, x[i], y[i], this);
            } else {
                g.drawImage(dot, x[i], y[i], this);
            }
        }

        Toolkit.getDefaultToolkit().sync();
    } else {
        gameOver(g);
    }
}

public void gameOver(Graphics g) {                    //Game Over Part
    String msg = "Game Over!";
    Font font = new Font("SAN_SERIF", Font.BOLD, 14);
    FontMetrics metrices = getFontMetrics(font);

    g.setColor(Color.WHITE);
```

```
    g.setFont(font);
    g.drawString(msg, (300 - metrices.stringWidth(msg)) / 2, 300/2);
}

public void move() {                                    //Movements
    for (int i = dots ; i > 0 ; i--) {
        x[i] = x[i - 1];
        y[i] = y[i - 1];
    }

    if (leftDirection) {
        x[0] = x[0] - DOT_SIZE;
    }
    if (rightDirection) {
        x[0] = x[0] + DOT_SIZE;
    }
    if (upDirection) {
        y[0] = y[0] - DOT_SIZE;
    }
    if (downDirection) {
        y[0] = y[0] + DOT_SIZE;
    }
}

public void checkApple() {
    if ((x[0] == apple_x) && (y[0] == apple_y)) {
        dots++;
        locateApple();
```

```
    }
}

public void checkCollision() {                          //Checking collision
    for(int i = dots; i > 0; i--) {
        if (( i > 4) && (x[0] == x[i]) && (y[0] == y[i])) {
            inGame = false;
        }
    }

    if (y[0] >= 300) {
        inGame = false;
    }
    if (x[0] >= 300) {
        inGame = false;
    }
    if (y[0] < 0) {
        inGame = false;
    }
    if (x[0] < 0) {
        inGame = false;
    }

    if (!inGame) {
        timer.stop();
    }
}
```

```
public void actionPerformed(ActionEvent ae) {
    if (inGame) {
        checkApple();
        checkCollision();
        move();
    }

    repaint();
}

public class TAdapter extends KeyAdapter {
    @Override
    public void keyPressed(KeyEvent e) {                    //Key Directions
        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;
            leftDirection = false;
            rightDirection = false;
        }

        if (key == KeyEvent.VK_DOWN && (!upDirection)) {
            downDirection = true;
            leftDirection = false;
            rightDirection = false;
        }
    }
  }

}
```

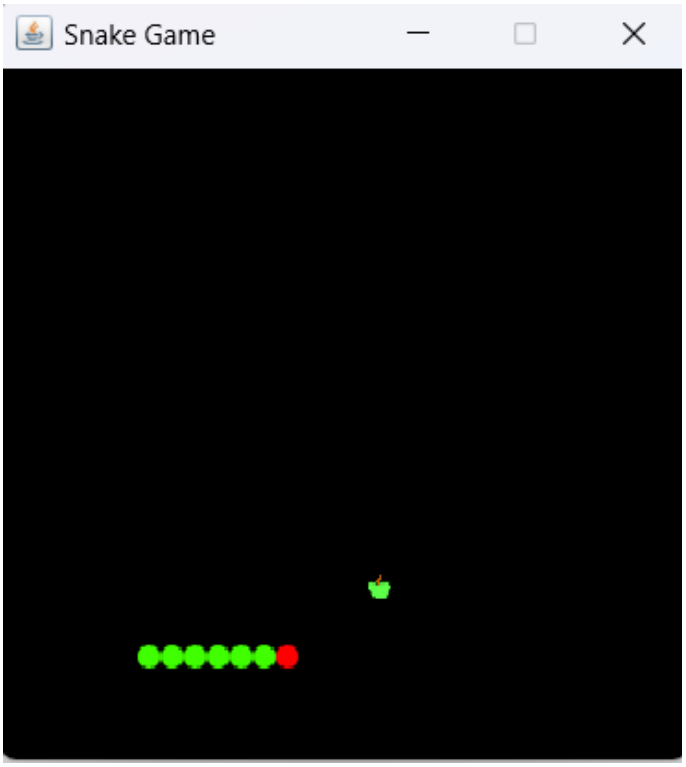
TO RUN THE CODE:

```
package snakegame;

import javax.swing.*;

public class SnakeGame extends JFrame {

  SnakeGame() {
    super("Snake Game");
    add(new Board());
    pack();

    setLocationRelativeTo(null);
    setResizable(false);
```

```
    }

    public static void main(String[] args) {
        new SnakeGame().setVisible(true);
    }
}
```

CHAPTER 5 – RESULTS

5.1 OUTPUT:



CHAPTER 6 – CONCLUSION

6.1 Conclusion: The conclusion of the *Snake Game* is marked by either the player losing or finishing a round. The game ends when the snake collides with itself or hits the walls, creating an intense challenge as the snake grows longer after eating food. Players aim for high scores, with each food item increasing their length and difficulty.

The game encourages skill development, improving reflexes and spatial awareness over time. It's also highly replayable, with each session feeling different due to random food placement and the increasing difficulty.

Snake has a lasting impact on gaming culture, especially as one of the first mobile games, influencing modern games with its simple, yet addictive gameplay. The game's conclusion is typically a moment of reflection, as players either strive to beat their high score or enjoy the nostalgia of a classic game.

CHAPTER 7-REFERENCES

7.1 References:

1. **Java Swing-https://www.geeksforgeeks.org/introduction-to-java-swing/**
2. **Zetcode-https://zetcode.com/javagames/snake/**
3. **Javatpoint-https://www.javatpoint.com/**