# SNAKE GAME

**Project Based Learning (PBL) Report**
**for the course**
**Object Oriented Programming – 20CS21002**

**BACHELOR OF TECHNOLOGY**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**

By
**G. LAKSHYA REDDY – 22R11A0514**
**KONREDDY HIMASRI – 22R11A0522**
**THRISHA N.C – 22R11A0544**

**Under the guidance of**
**N. Vinay Kumar**

**Department of Computer Science and Engineering**
**Accredited by NBA**

**Geethanjali College of Engineering and Technology**
**(UGC Autonomous)**
(Affiliated to J.N.T.U.H, Approved by AICTE, New Delhi)
Cheeryal (V), Keesara (M), Medchal.Dist.-501 301.

**December -2023**

**Geethanjali College of Engineering andTechnology**

**(UGC Autonomous)**

(Affiliated to J.N.T.U.H, Approved by AICTE, New Delhi)

Cheeryal (V), Keesara (M), Medchal.Dist.-501 301.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Accredited by NBA

# Certificate

This is to certify that the B.Tech Mini Project reported entitled "**SNAKE GAME**"is a bonafide work G.LAKSHYA REDDY (22R11A0514), KONREDDY HIMASRI (22R11A0522), THRISHA N.C (22R11A0544) in partial fulfillment of the requirement of the award for the degree of Bachelor of Technology in "**Computer Science and Engineering"** from Geethanjali College of Engineering and Technology, Medchal ,Telangana during the year 2023-2026.

| | |
|---|---|
| **Internal Guide** | **HOD-CSE** |
| **Mr. N. Vinay Kumar** | **Dr.Sree Lakshmi** |
| **(Assistant Professor)** | **(Professor)** |

**Geethanjali College of Engineering andTechnology**

**(UGC Autonomous)**

(Affiliated to J.N.T.U.H, Approved by AICTE, New Delhi)

Cheeryal (V), Keesara (M), Medchal.Dist.-501 301.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Accredited by NBA

## DECLARATION BY CANDIDATE

We **G. LAKSHYA REDDY-22R11A0514, KONREDDY HIMASRI-22R11A0522, THRISHA N.C -22R11A0544,** hearby declare that the project report entitled "**Snake Game**" is done under the guidance of **Mr.N.Vinay Kumar, Assistant Professor**, Department of Computer of Science and Engineering, Geethanjali College of Engineering and Technology, is submitted in partial fulfillment of the requirements for the award of the degree of **Bachelor of Engineering in Computer Science and Engineering.**

# ACKNOWLEDGMENT

We are greatly indebted to the authorities of Geethanjali College of Engineering and Technology Cheeryal, Medchal District. For providing us the necessary facilities to successfully completion of this mini project titled “**SNAKE GAME**”.

Firstly we thank and express our solicit gratitude to **Mr.N.Vinay Kumar, Assistant Professor,** Internal guide, Geethanjali College of Engineering and Technology, for her valuable help and support which helped us a lot in successfully completing our mini project.

Secondly, we express our gratitude to **Dr. Sree Lakshmi**, **Head of Department, CSE** for being moral support throughout the period of study in **GCET.**

We would like to express our sincere gratitude to our **principal**, **Dr. Udaya Kumar Susarla** for providing the necessary infrastructure to complete our project. Finally, we would like to express our heartfelt thanks to our parents who were very supportive both financially and mentally and for their encouragement to achieve our goals.

# INDEX

| S.No. | Contents | Page No |
|---|---|---|
| **1** | **Abstract** | 6 |
| **2** | **Introduction** | 7 |
| **3** | **System Design**<br>• Game Flowchart | 8<br>9 |
| **4** | **Implementation** | 10 |
| **5** | **Java Program** | 11 |
| **6** | **Sample Output** | 22 |
| **7** | **Conclusion** | 25 |
| **8** | **References** | 26 |

## OUTPUTS(INDEX)

| S.No | Contents | Page No |
|---|---|---|
| 1 | Fig-1: Main Frame | 22 |
| 2 | Fig-2: Game Frame | 23 |
| 3 | Fig-3: Game Over Frame | 24 |

# Abstract

The Snake Game, implemented in Java, exemplifies a sophisticated application of an object-oriented paradigm to create a classic and captivating arcade experience. In this rendition, the game unfolds within a grid-based environment, where each cell serves as the canvas for the snake's dynamic journey. The snake, a primary game entity, is ingeniously represented as a collection of interconnected segments, each encapsulated as an object. These objects possess essential properties such as position and direction, contributing to the overall modularity and maintainability of the codebase.

The grid, the virtual playground for the snake, is partitioned into cells, distinguishing between empty spaces, snake segments, and randomly placed food items. User interaction is facilitated through keyboard input, typically utilizing arrow keys to dictate the snake's movement. The game's essence lies in the strategic navigation of the snake across the grid, with the primary objectives being the consumption of food items and the avoidance of collisions, both with itself and the grid boundaries.

A pivotal aspect of the game's functionality is the implementation of collision detection mechanisms. These mechanisms dynamically respond to various interactions within the game, determining the consequences of the snake colliding with itself or breaching the grid boundaries. A scoring system is integrated to quantitatively measure the player's success, incrementing with each successfully consumed food item. The player's score is prominently displayed on the game interface, providing immediate feedback on their performance.

As the game progresses, the possibility of collision looms, and upon its occurrence, a "Game Over" message is presented alongside the final score. However, the engagement doesn't end there; players have the option to restart the game, initiating a fresh round and allowing for continuous gameplay.

Graphical elements in the game are intentionally kept simple yet effective. Basic shapes and minimalistic designs are employed to represent the snake, food items, and the grid itself. This simplicity enhances the game's accessibility while maintaining its nostalgic appeal.

By adhering to object-oriented programming principles, the Snake Game implementation achieves a high level of code modularity and reusability. This Java-based rendition not only serves as an educational tool but also stands as an entertaining example of game development, illustrating fundamental concepts like the game loop, event handling, and object-oriented design within the framework of a classic arcade game.

**KEYWORDS:**

Object-Oriented paradigm, Strategic Navigation, Nostalgic appeal, Code Modularity, Accessibility.

# Introduction

## About the Project

The Snake Game project involves the development of a classic game in Java. Inspired by the original Snake game popularized in early mobile phones and arcade systems, this project aims to recreate the gaming experience. Players control a snake moving within a bordered area, with the objective of consuming food items to grow longer while avoiding collisions with itself or the walls.

## Project Outcomes and Objectives

The primary objective of this project is to develop a functional and interactive Snake Game using Java. The core objectives include:

- Implementing game mechanics that involve controlling the snake's movement using keyboard inputs.
- Designing an engaging graphical interface to display the game board, snake, food items, and game statistics.
- Demonstrating key programming concepts such as data structures (for the snake's body), game logic, and event handling.
- Offering a user-friendly and visually appealing gaming experience while maintaining the essence of the original Snake Game.

# System Design

## System Architecture

The game architecture revolves around three main components: User Input, Game Logic, and Graphics Rendering.

**User Input Module:** Listens for user commands via keyboard input, interpreting arrow keys for controlling the snake's movement.

**Game Logic Module:** Manages the game's core logic, including the snake's movement, growth, collision detection, and game state updates.

**Graphics Module:** Utilizes Java's built-in graphics libraries (e.g., Swing or AWT or JavaFX etc..) to render the game environment, displaying the snake, food items, boundaries, and game statistics.

## Modules

## User Input Module

The User Input Module captures keyboard inputs using Java's event-driven architecture. It listens for arrow key presses (UP, DOWN, LEFT, RIGHT) to determine the snake's direction.

## Game Logic Module

This module governs the behavior of the snake and manages the game state. It includes functionalities such as:

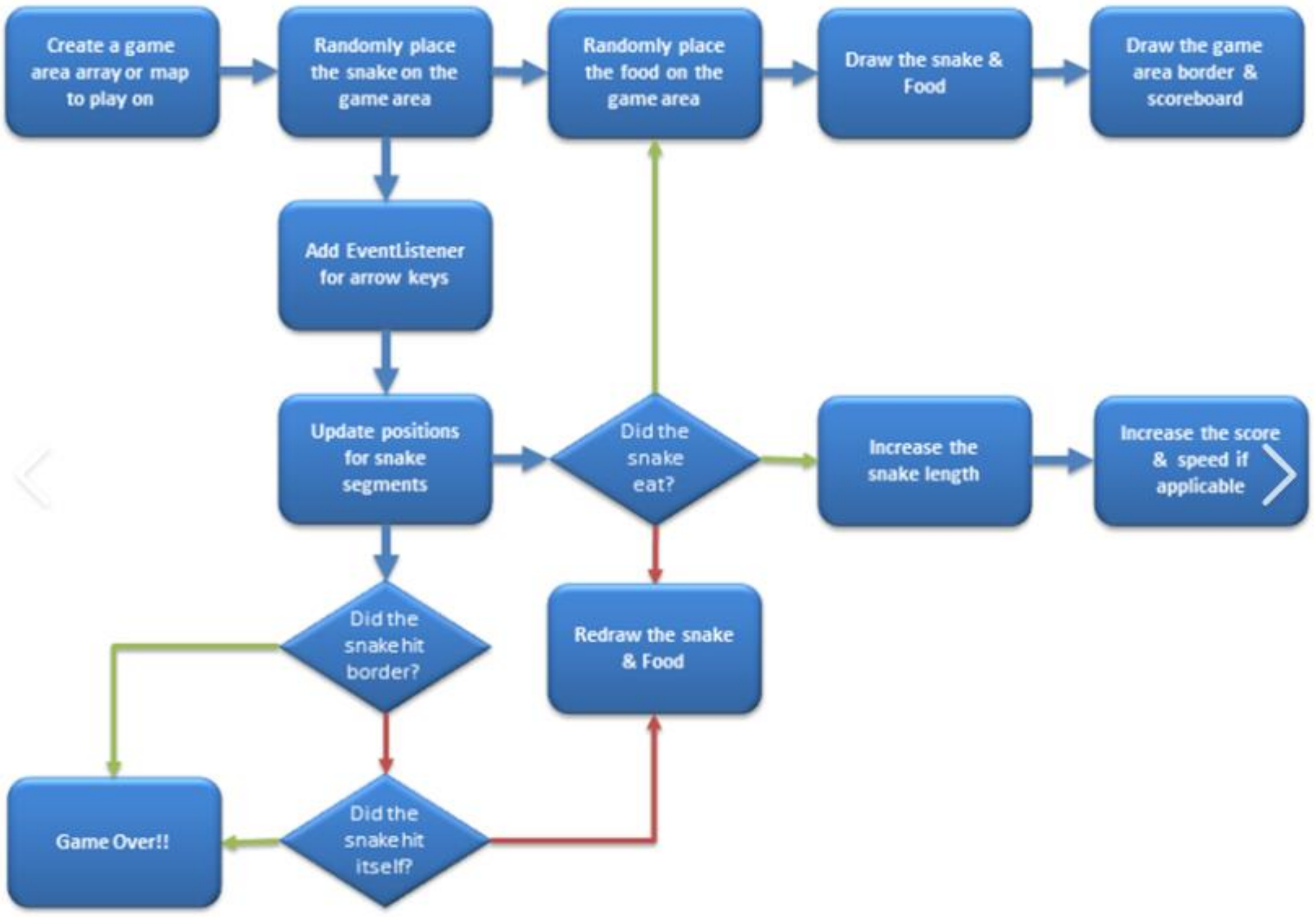
- Snake movement mechanics based on the user's input direction.
- Collision detection to check if the snake collides with itself or the game boundaries.
- Food generation and scoring mechanism upon food consumption.

## Graphics Module

Responsible for rendering the game's graphical elements on the screen, the Graphics Module involves:

- Drawing the game board with grid lines to represent the playable area.
- Displaying the snake as a series of interconnected segments.
- Rendering food items and updating the visual representation upon each game event.
- Backend Design
- The backend handles the internal game state, managing variables such as the snake's position, size, score, and game-over conditions. It orchestrates the communication between different modules, ensuring synchronized gameplay and graphical representation.

## Game Flowchart


Create a game area array or map to play on
Randomly place the snake on the game area
Randomly place the food on the game area
Draw the snake & Food
Draw the game area border & scoreboard
Add EventListener for arrow keys
Update positions for snake segments
Did the snake eat?
Increase the snake length
Increase the score & speed if applicable
Did the snake hit border?
Redraw the snake & Food
Game Over!!
Did the snake hit itself?

# Implementation

## Modules Implementation

### User Input

- Utilizes Java's KeyListener or event handling mechanisms to capture arrow key presses.
- Translates the user's input into commands to control the snake's direction.

### Game Logic

- Implements a data structure (e.g., ArrayList) to represent the snake's body segments.
- Calculates the snake's movement based on the chosen direction and updates its position accordingly.
- Implements collision detection to identify collisions with walls or the snake's own body.
- Handles food generation, consumption, and scoring.

### Graphics

- Utilizes Java's graphics libraries (e.g., Java Swing) to create a visual representation of the game board.
- Renders the snake, food items, boundaries, and additional graphical elements.
- Updates the graphical interface based on game events, such as snake movement, food consumption, and game-over scenarios.

## Sample Code

```
import java.util.*;
import javax.imageio.ImageIO;
import java.util.Timer;
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import java.io.*;
import javax.swing.*;

class Game extends JPanel {
    private Timer timer;
    private Snake snake;
    private Point cherry;
    private int points = 0;
    private int best = 0;
    private BufferedImage image;
    private GameStatus status;
    private boolean didLoadCherryImage = true;

    private static Font FONT_M = new Font("MV Boli", Font.PLAIN, 24);
    private static Font FONT_M_ITALIC = new Font("MV Boli", Font.ITALIC, 24);
    private static Font FONT_L = new Font("MV Boli", Font.PLAIN, 84);
    private static Font FONT_XL = new Font("MV Boli", Font.PLAIN, 150);
    private static int WIDTH = 760;
    private static int HEIGHT = 520;
    private static int DELAY = 50;
```

```
// Constructor
  public Game() {
      try {
          image = ImageIO.read(new File("cherry.png"));
      } catch (IOException e) {
        didLoadCherryImage = false;
      }

      addKeyListener(new KeyListener());
      setFocusable(true);
      setBackground(new Color(130, 205, 71));
      setDoubleBuffered(true);

      snake = new Snake(WIDTH / 2, HEIGHT / 2);
      status = GameStatus.NOT_STARTED;
      repaint();
  }

  @Override
  public void paintComponent(Graphics g) {
      super.paintComponent(g);
      render(g);
      Toolkit.getDefaultToolkit().sync();
  }
  // Render the game
  private void update() {
      snake.move();
      if (cherry != null && snake.getHead().intersects(cherry, 20)) {
          snake.addTail();
          cherry = null;
          points++;
      }
```

```
    if (cherry == null) {
        spawnCherry();
    }
    checkForGameOver();
}
private void reset() {
    points = 0;
    cherry = null;
    snake = new Snake(WIDTH / 2, HEIGHT / 2);
    setStatus(GameStatus.RUNNING);
}
private void setStatus(GameStatus newStatus) {
    switch(newStatus) {
        case RUNNING:
            timer = new Timer();
            timer.schedule(new GameLoop(), 0, DELAY);
            break;
        case PAUSED:
            timer.cancel();
        case GAME_OVER:
            timer.cancel();
            best = points > best ? points : best;
            break;
    }
    status = newStatus;
}
private void togglePause() {
    setStatus(status == GameStatus.PAUSED ? GameStatus.RUNNING : GameStatus.PAUSED);
}
```

```
// Check if the snake has hit the wall or itself
  private void checkForGameOver() {
      Point head = snake.getHead();
      boolean hitBoundary = head.getX() <= 20
          || head.getX() >= WIDTH + 10
          || head.getY() <= 40
          || head.getY() >= HEIGHT + 30;

      boolean ateItself = false;

      for(Point t : snake.getTail()) {
          ateItself = ateItself || head.equals(t);
      }

      if (hitBoundary || ateItself) {
          setStatus(GameStatus.GAME_OVER);
      }
  }
  // Spawn a cherry at a random location
  public void drawCenteredString(Graphics g, String text, Font font, int y) {
      FontMetrics metrics = g.getFontMetrics(font);
      int x = (WIDTH - metrics.stringWidth(text)) / 2;

      g.setFont(font);
      g.drawString(text, x, y);
  }
  private void render(Graphics g) {
      Graphics2D g2d = (Graphics2D) g;

      g2d.setColor(Color.BLACK);
      g2d.setFont(FONT_M);
```

```
if (status == GameStatus.NOT_STARTED) {
  drawCenteredString(g2d, "SNAKE", FONT_XL, 200);
  drawCenteredString(g2d, "GAME", FONT_XL, 300);
  drawCenteredString(g2d, "Press  any  key  to  begin", FONT_M_ITALIC, 330);
  return;
}
Point p = snake.getHead();
g2d.drawString("SCORE: " + String.format ("%02d", points), 20, 30);
g2d.drawString("BEST: " + String.format ("%02d", best), 630, 30);
if (cherry != null) {
  if (didLoadCherryImage) {
    g2d.drawImage(image, cherry.getX(), cherry.getY(), 60, 60, null);
  } else {
    g2d.setColor(Color.BLACK);
    g2d.fillOval(cherry.getX(), cherry.getY(), 10, 10);
    g2d.setColor(Color.BLACK);
  }
}
if (status == GameStatus.GAME_OVER) {
    drawCenteredString(g2d, "Press  enter  to  start  again", FONT_M_ITALIC, 330);
    drawCenteredString(g2d, "GAME OVER", FONT_L, 300);
}
if (status == GameStatus.PAUSED) {
    g2d.drawString("Paused", 600, 14);
}
g2d.setColor(new Color(33, 70, 199));
g2d.fillRect(p.getX(), p.getY(), 10, 10);
for(int i = 0, size = snake.getTail().size(); i < size; i++) {
    Point t = snake.getTail().get(i);
    g2d.fillRect(t.getX(), t.getY(), 10, 10);
}
```

```
        g2d.setColor(Color.RED);
        g2d.setStroke(new BasicStroke(4));
        g2d.drawRect(20, 40, WIDTH, HEIGHT);
    }

    // spawn cherry in random position
    public void spawnCherry() {
        cherry = new Point((new Random()).nextInt(WIDTH - 60) + 20,
            (new Random()).nextInt(HEIGHT - 60) + 40);
    }



  // game loop
    private class KeyListener extends KeyAdapter {
        @Override
        public void keyPressed(KeyEvent e) {
            int key = e.getKeyCode();

            if (status == GameStatus.RUNNING) {
                switch(key) {
                    case KeyEvent.VK_LEFT: snake.turn(Direction.LEFT); break;
                    case KeyEvent.VK_RIGHT: snake.turn(Direction.RIGHT); break;
                    case KeyEvent.VK_UP: snake.turn(Direction.UP); break;
                    case KeyEvent.VK_DOWN: snake.turn(Direction.DOWN); break;
                }
            }
            if (status == GameStatus.NOT_STARTED) {
                setStatus(GameStatus.RUNNING);
            }
            if (status == GameStatus.GAME_OVER && key == KeyEvent.VK_ENTER) {
                reset();
            }
            if (key == KeyEvent.VK_P) {
                togglePause();
            }
        }
    }
    private class GameLoop extends java.util.TimerTask {
        public void run() {
            update();
            repaint();
        }
    }
}
enum GameStatus
{
    NOT_STARTED, RUNNING, PAUSED, GAME_OVER
}

// direction of snake
enum Direction {
    UP, DOWN, LEFT, RIGHT;

    public boolean isX() {
        return this == LEFT || this == RIGHT;
    }

    public boolean isY() {
        return this == UP || this == DOWN;
    }
}
class Point {
    private int x;
    private int y;
```

```
public Point(int x, int y) {
   this.x = x;
   this.y = y;
}
public Point(Point p) {
   this.x = p.getX();
   this.y = p.getY();
}
public void move(Direction d, int value) {
   switch(d) {
      case UP: this.y -= value; break;
      case DOWN: this.y += value; break;
      case RIGHT: this.x += value; break;
      case LEFT: this.x -= value; break;
   }
}
public int getX() {
   return x;
}
public int getY() {
   return y;
}
public Point setX(int x) {
   this.x = x;
   return this;
}
public Point setY(int y) {
   this.y = y;
   return this;
}
```

```
 public boolean equals(Point p) {
        return this.x == p.getX() && this.y == p.getY();
    }
    public String toString() {
        return "(" + x + ", " + y + ")";
    }
    public boolean intersects(Point p) {
        return intersects(p, 10);
    }
    public boolean intersects(Point p, int tolerance) {
        int diffX = Math.abs(x - p.getX());
        int diffY = Math.abs(y - p.getY());
        return this.equals(p) || (diffX <= tolerance && diffY <= tolerance);
    }
}

class Snake {
    private Direction direction;
    private Point head;
    private ArrayList<Point> tail;

    public Snake(int x, int y) {
        this.head = new Point(x, y);
        this.direction = Direction.RIGHT;
        this.tail = new ArrayList<Point>();
        this.tail.add(new Point(0, 0));
        this.tail.add(new Point(0, 0));
        this.tail.add(new Point(0, 0));
    }
    public void move() {
        ArrayList<Point> newTail = new ArrayList<Point>();
```

```
  for (int i = 0, size = tail.size(); i < size; i++) {
            Point previous = i == 0 ? head : tail.get(i - 1);
            newTail.add(new Point(previous.getX(), previous.getY()));
        }
        this.tail = newTail;
        this.head.move(this.direction, 10);
    }
    public void addTail() {
        this.tail.add(new Point(-10, -10));
    }
    public void turn(Direction d) {
        if (d.isX() && direction.isY() || d.isY() && direction.isX()) {
           direction = d;
        }
    }
    public ArrayList<Point> getTail() {
        return this.tail;
    }
    public Point getHead() {
        return this.head;
    }
}
public class Main extends JFrame {
    public Main() {
        initUI();
    }
```

```
private void initUI() {
     add(new Game());
     setTitle("Snake");
     setSize(800, 610);
     setLocationRelativeTo(null);
     setResizable(false);
     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 }

 public static void main(String[] args) {
     EventQueue.invokeLater(() -> {
         Main ex = new Main();
         ex.setVisible(true);
     });
```

## Sample Output Screenshots/Results

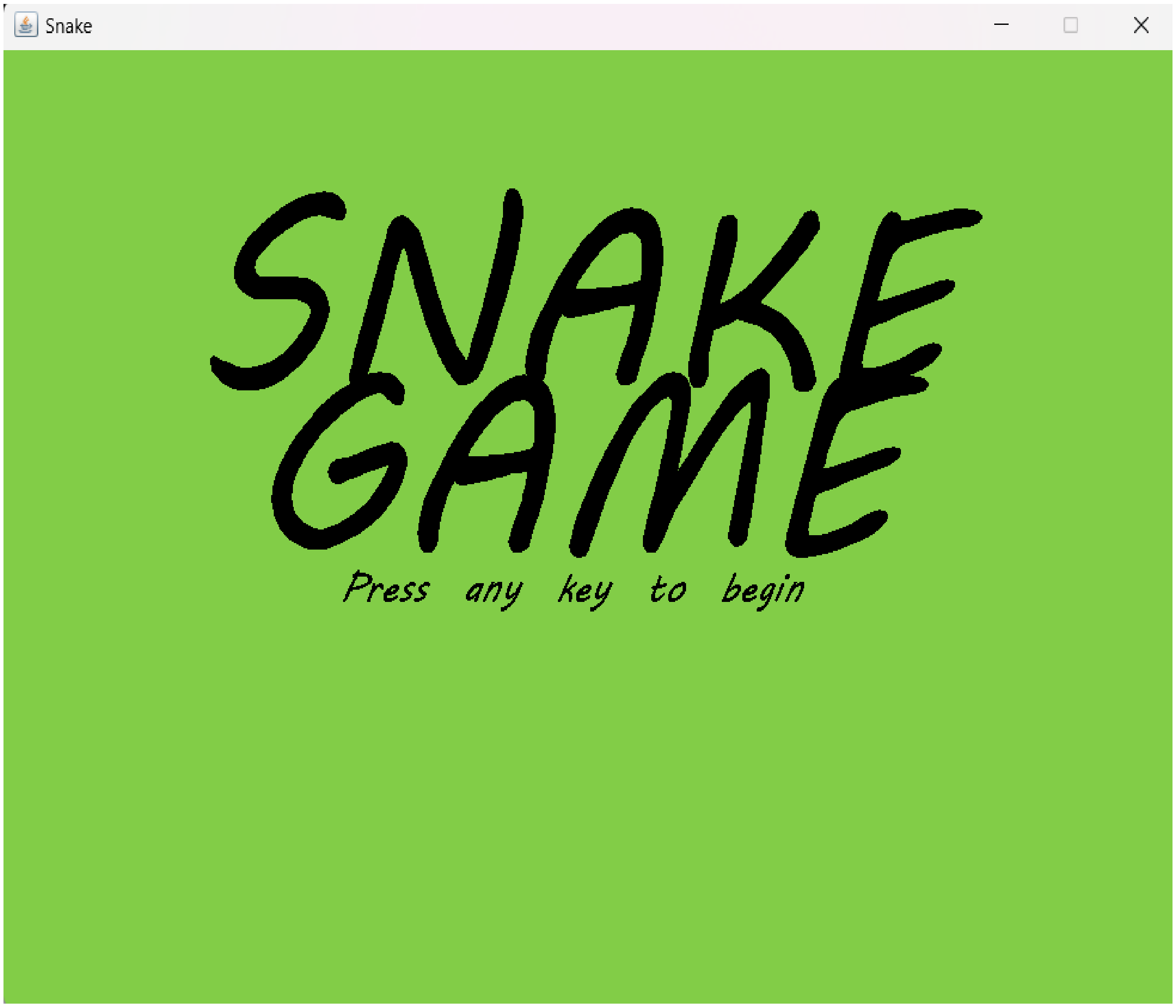


**Fig: 1:** Main Frame

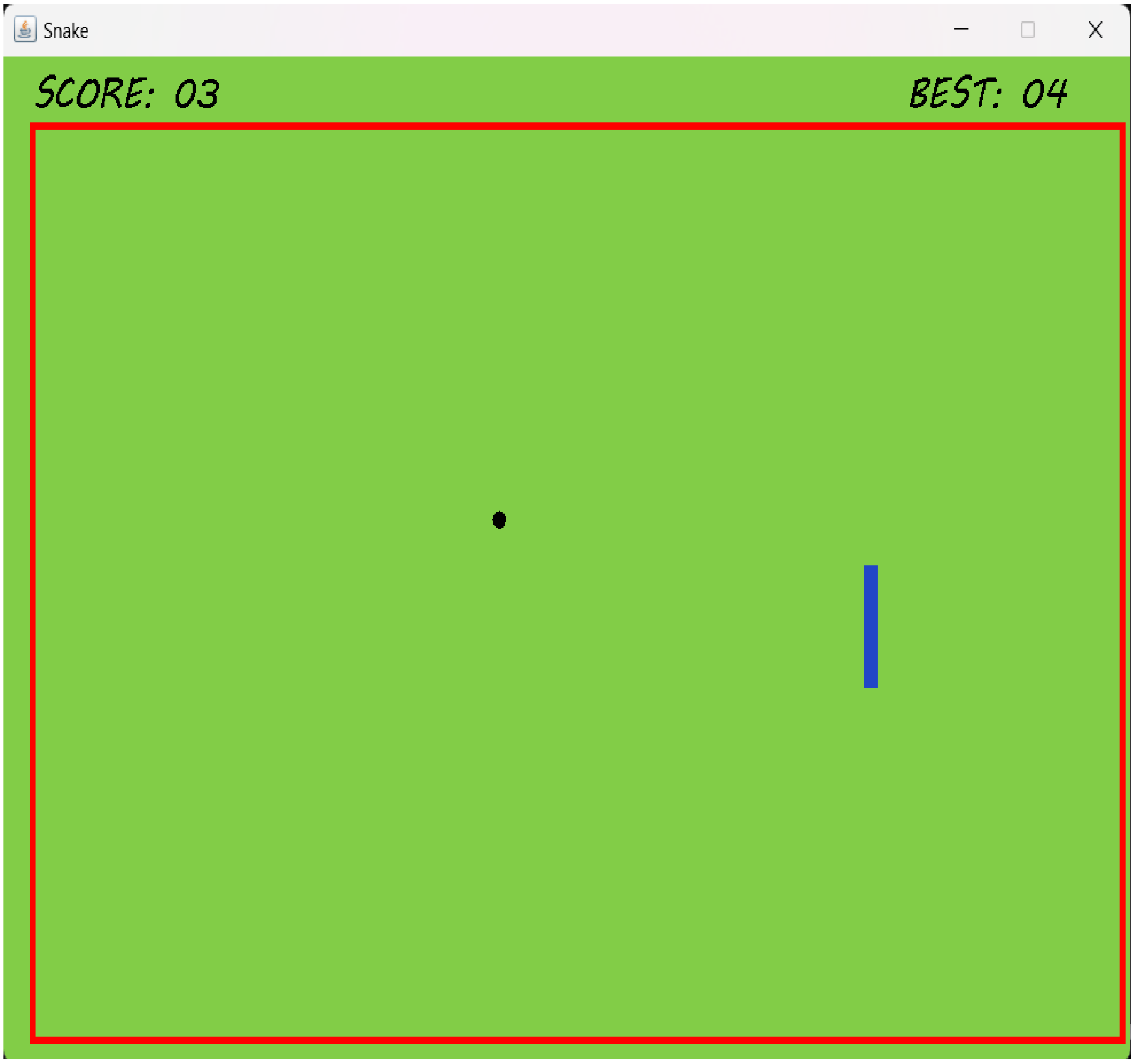


**Fig: 2:** Game Frame

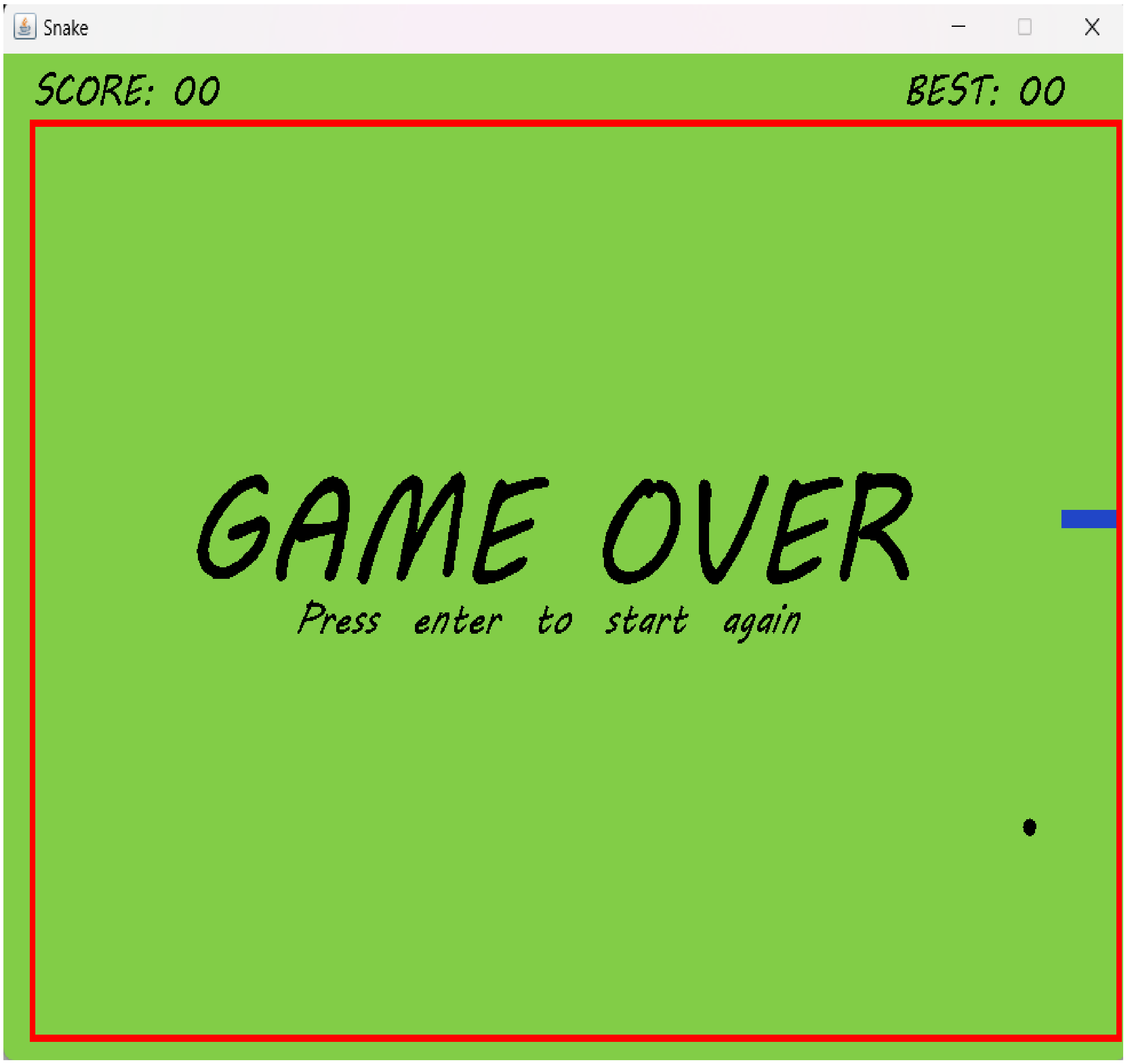


**Fig: 3:** Game Over Frame

# Conclusion

The Snake Game in Java successfully achieves its set objectives by implementing essential gaming components, offering an engaging and enjoyable gameplay experience. The project demonstrates the practical application of core programming concepts in game development, enhancing user interaction and entertainment.

## References

List of resources, tutorials, or references used in developing the Snake Game in Java.

1. https://www.javatpoint.com/
2. https://www.geeksforgeeks.org/
3. Newspapers referred: the Guardian, The Wall Street Journal, Tech-focused publications