# Snake Game Application Documentation

## Table of Contents

1. Introduction
2. Design Choices
3. Implementation Details
    - Main Class
    - GamePanel Class
    - Snake Class
    - Food Class
4. User Interface
5. Challenges Faced
6. Conclusion

## Introduction

The Snake Game Application is a Java program that recreates the classic snake game. The goal is to control a snake, navigating it around the screen to consume food, which causes the snake to grow longer. The game ends if the snake collides with the walls or itself. The application utilizes Java's `Swing` library for the graphical user interface and game mechanics.

## Design Choices

### 1. Object-Oriented Design

The application is designed using an object-oriented approach to encapsulate the different components of the game such as the snake, food, and game panel. This makes the code modular, maintainable, and scalable.

### 2. User-Friendly Interface

The game uses a graphical user interface (GUI) created with Java Swing, providing an interactive and visually appealing experience. Swing is chosen for its simplicity and ease of use for creating window-based applications in Java.

### 3. Real-Time Interaction

The game requires real-time interaction for moving the snake, which is achieved using a game loop that updates the game state at regular intervals. Key events are captured to control the snake's direction.

## Implementation Details

The application consists of the following main components: `Main` class, `GamePanel` class, `Snake` class, and `Food` class.

### Main Class

The `Main` class serves as the entry point of the application. It sets up the game window and starts the game.

```
import javax.swing.JFrame;


public class Main {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Snake Game");
        GamePanel gamePanel = new GamePanel();


        frame.add(gamePanel);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setResizable(false);

        frame.pack();

        frame.setVisible(true);


        gamePanel.startGame();

    }

}
```

## GamePanel Class

The `GamePanel` class extends `JPanel` and implements `ActionListener` and `KeyListener`. It contains the game loop, renders the game graphics, and handles user input.

```
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;


public class GamePanel extends JPanel implements ActionListener,
KeyListener {

    private static final int WIDTH = 600;
    private static final int HEIGHT = 600;

    private static final int UNIT_SIZE = 25;

    private static final int GAME_UNITS = (WIDTH * HEIGHT) /
(UNIT_SIZE * UNIT_SIZE);

    private static final int DELAY = 75;



    private final Snake snake;

    private final Food food;

    private boolean running = false;

    private Timer timer;



    public GamePanel() {

        this.setPreferredSize(new Dimension(WIDTH, HEIGHT));

        this.setBackground(Color.BLACK);

        this.setFocusable(true);

        this.addKeyListener(this);



        snake = new Snake();

        food = new Food();

    }
    public void startGame() {

        spawnFood();

        running = true;

        timer = new Timer(DELAY, this);

        timer.start();

    }



    @Override

    public void paintComponent(Graphics g) {

        super.paintComponent(g);

        draw(g);

    }



    public void draw(Graphics g) {

        if (running) {

            g.setColor(Color.RED);

            g.fillOval(food.getX(), food.getY(), UNIT_SIZE,
UNIT_SIZE);
```

```
            for (int i = 0; i < snake.getBodyParts(); i++) {
                if (i == 0) {
                    g.setColor(Color.GREEN);
                } else {
                    g.setColor(new Color(45, 180, 0));
                }
                g.fillRect(snake.getX(i), snake.getY(i), UNIT_SIZE,
UNIT_SIZE);
            }
        } else {
            gameOver(g);
        }
    }

    public void spawnFood() {
        food.spawn(WIDTH, HEIGHT, UNIT_SIZE);
    }

    public void checkFood() {
        if ((snake.getX(0) == food.getX()) && (snake.getY(0) ==
food.getY())) {
```

```
            snake.grow();

            spawnFood();

        }

    }



    public void checkCollisions() {

        for (int i = snake.getBodyParts(); i > 0; i--) {

            if ((snake.getX(0) == snake.getX(i)) && (snake.getY(0) ==
snake.getY(i))) {

                running = false;

            }

        }

        if (snake.getX(0) < 0 || snake.getX(0) >= WIDTH ||
snake.getY(0) < 0 || snake.getY(0) >= HEIGHT) {

            running = false;

        }

        if (!running) {

            timer.stop();

        }

    }
    public void gameOver(Graphics g) {

        g.setColor(Color.RED);

        g.setFont(new Font("Ink Free", Font.BOLD, 75));

        FontMetrics metrics = getFontMetrics(g.getFont());

        g.drawString("Game Over", (WIDTH -
metrics.stringWidth("Game Over")) / 2, HEIGHT / 2);

    }



    @Override

    public void actionPerformed(ActionEvent e) {

        if (running) {

            snake.move();

            checkFood();

            checkCollisions();

        }

        repaint();

    }



    @Override

    public void keyPressed(KeyEvent e) {
```

```
switch (e.getKeyCode()) {
    case KeyEvent.VK_LEFT:
        if (snake.getDirection() != 'R') {
            snake.setDirection('L');
        }
        break;
    case KeyEvent.VK_RIGHT:
        if (snake.getDirection() != 'L') {
            snake.setDirection('R');
        }
        break;
    case KeyEvent.VK_UP:
        if (snake.getDirection() != 'D') {
            snake.setDirection('U');
        }
        break;
    case KeyEvent.VK_DOWN:
        if (snake.getDirection() != 'U') {
            snake.setDirection('D');
        }
```

```
            break;
        }
    }

    @Override
    public void keyReleased(KeyEvent e) {}

    @Override
    public void keyTyped(KeyEvent e) {}
}
```

## Snake Class

The `Snake` class represents the snake, including its position, size, and movement logic.

```
public class Snake {
    private final int[] x;
    private final int[] y;
    private int bodyParts;
    private char direction;

    public Snake() {
        x = new int[GamePanel.GAME_UNITS];

        y = new int[GamePanel.GAME_UNITS];

        bodyParts = 6;

        direction = 'R';



        for (int i = 0; i < bodyParts; i++) {

            x[i] = 50 - i * 10;

            y[i] = 50;

        }

    }



    public void move() {

        for (int i = bodyParts; i > 0; i--) {

            x[i] = x[i - 1];

            y[i] = y[i - 1];

        }



        switch (direction) {

            case 'U':

                y[0] = y[0] - GamePanel.UNIT_SIZE;
                break;
            case 'D':
                y[0] = y[0] + GamePanel.UNIT_SIZE;
                break;
            case 'L':
                x[0] = x[0] - GamePanel.UNIT_SIZE;
                break;
            case 'R':
                x[0] = x[0] + GamePanel.UNIT_SIZE;
                break;
        }
    }

    public void grow() {
        bodyParts++;
    }

    public int getX(int index) {
        return x[index];
    }
```

```
    public int getY(int index) {
        return y[index];
    }

    public int getBodyParts() {
        return bodyParts;
    }

    public char getDirection() {
        return direction;
    }

    public void setDirection(char direction) {
        this.direction = direction;
    }
}
```

## Food Class

The `Food` class represents the food that the snake consumes to grow.

```
import java.util.Random;
```

```
public class Food {

    private int x;

    private int y;

    private final Random random;


    public Food() {

        random = new Random();

        spawn(600, 600, 25);

    }


    public void spawn(int width, int height, int unitSize) {

        x = random.nextInt((int) (width / unitSize)) * unitSize;

        y = random.nextInt((int) (height / unitSize)) * unitSize;

    }


    public int getX() {

        return x;

    }
    public int getY() {
        return y;
    }
}
```

## User Interface

The user interface is graphical, built using Java Swing. It provides a game window where the snake is controlled by the arrow keys. The game starts with a predefined snake length, and the user navigates the snake to eat the food that randomly appears on the screen.

### Example Interaction

1. Launch the game.
2. Use the arrow keys to move the snake.
3. The snake grows longer each time it eats the food.
4. The game ends if the snake collides with the wall or itself.
5. A "Game Over" message is displayed upon losing.

## Challenges Faced

### 1. Real-Time Game Loop

Implementing a real-time game loop that updates the game state at regular intervals was challenging. This was achieved using a `Timer` object that triggers `ActionListener` events.

### 2. Collision Detection

Detecting collisions with the wall or the snake's own body required careful calculations and checks. Ensuring these checks were efficient and accurate was crucial for the game's functionality.

### 3. Responsive Controls

Handling keyboard input to change the snake's direction while avoiding quick, successive changes that could cause the snake to collide with itself was a significant challenge. This was managed by setting conditions on direction changes.

## Conclusion

The Snake Game Application successfully recreates the classic snake game using Java. It demonstrates the use of object-oriented programming, graphical user interfaces with Java Swing, and real-time game mechanics. The challenges faced during implementation provided valuable insights into game development and Java programming. Future enhancements could include adding levels, more obstacles, and improving the graphical interface for a richer user experience.