Creating A Snake Game In Java

Lecturer:

D4017_Jude Joseph Lamug Martinez, MCS

Arranged by:

2702296672_Aldika Fama Reynanda

Object Oriented programming

Computer Science Faculty

Binus International University 2023/2024

Abstract

This project aims to recreate the classic Snake game using Object-Oriented Programming (OOP) principles in Java. The Snake game is a timeless classic that challenges players to control a snake as it navigates a grid, consuming food and growing longer with each bite, while avoiding collisions with itself and the walls. The *IDE* that this project is using is Intellij IDE.

Chapter 1 - Project Description

1.1 Introduction

In this project, I am recreating the classic Snake game using Java and Object-Oriented Programming (OOP). With Java, I organize the game into smaller parts that work together. This helps me to learn important concepts like creating different pieces of the game and how they interact. and I combine old-fashioned fun with modern ways of making computer games. This project especially helps me learn while having a great time.

1.2 Idea inspiration

The idea why i made this Snake game because i am new to the Java environment and i want to learn Java but the amount of material that i need to learn is wide ranging to easy to the hardest and it demotivate me, so i have a thought that i should made a project to challenge myself to create something while learning something and that is how i decide i want to make Snake game because i played it in the past it was an iconic game and it's not a boring project to make.

1.3 How it works

This Java Project creates a simple Snake game using Swing for the GUI and sound effects. The `App` class initializes a `JFrame` window and adds an instance of `SnakeGame`. The `SnakeGame` class extends `JPanel` and implements `ActionListener` and `KeyListener` to manage game updates and user input. The game initializes the snake and food positions, sets a timer for game updates, and handles key events to control the snake's direction. The game loop checks for collisions with food or the snake's body and updates the display accordingly. Eating food grows the snake and plays a sound effect, while background music loops during gameplay and stops upon game over, triggered by collisions or boundary breaches.

Chapter 2 - Use Case Diagram

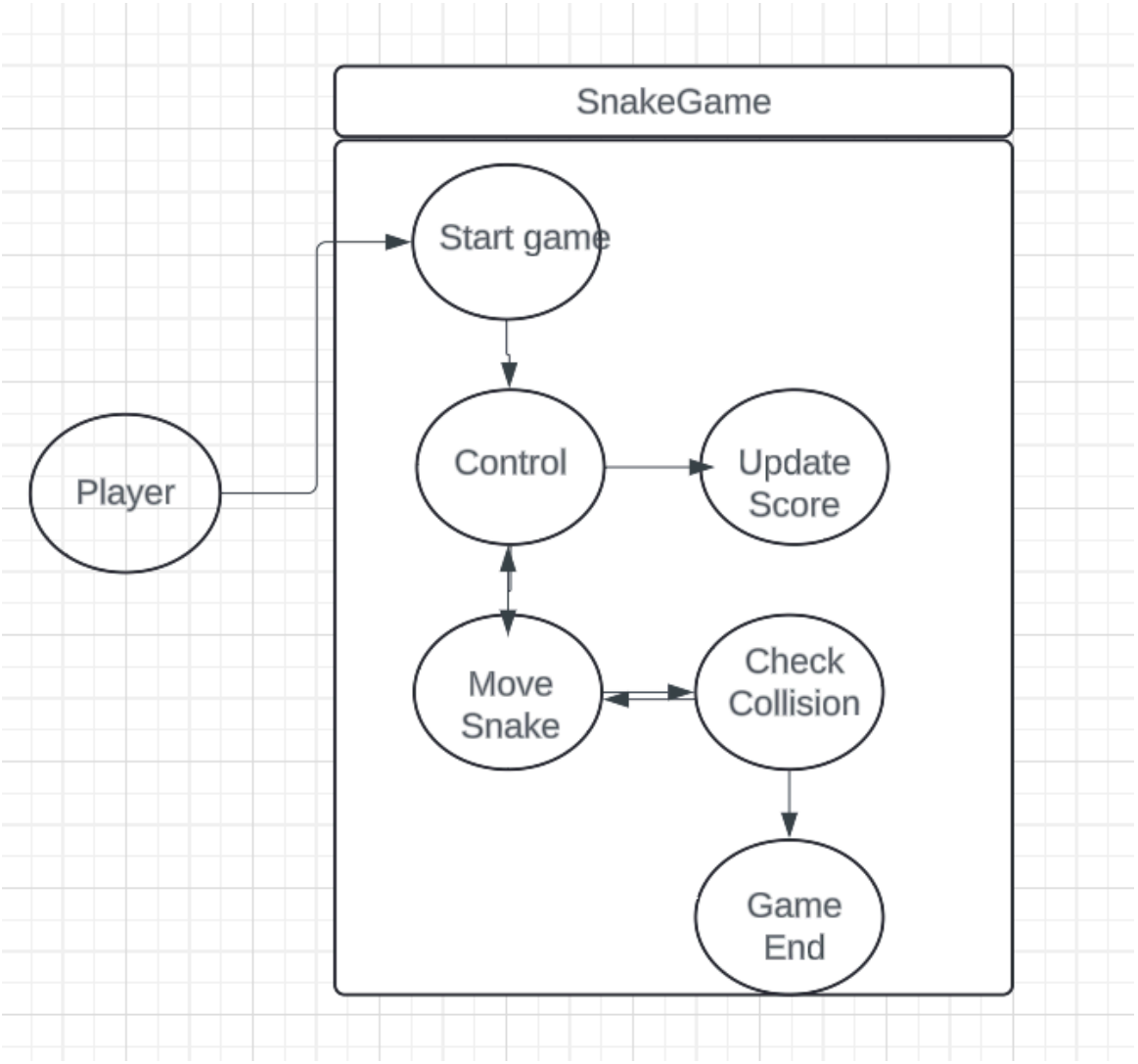


Figure 2.1

Chapter 3 - Activity Diagram

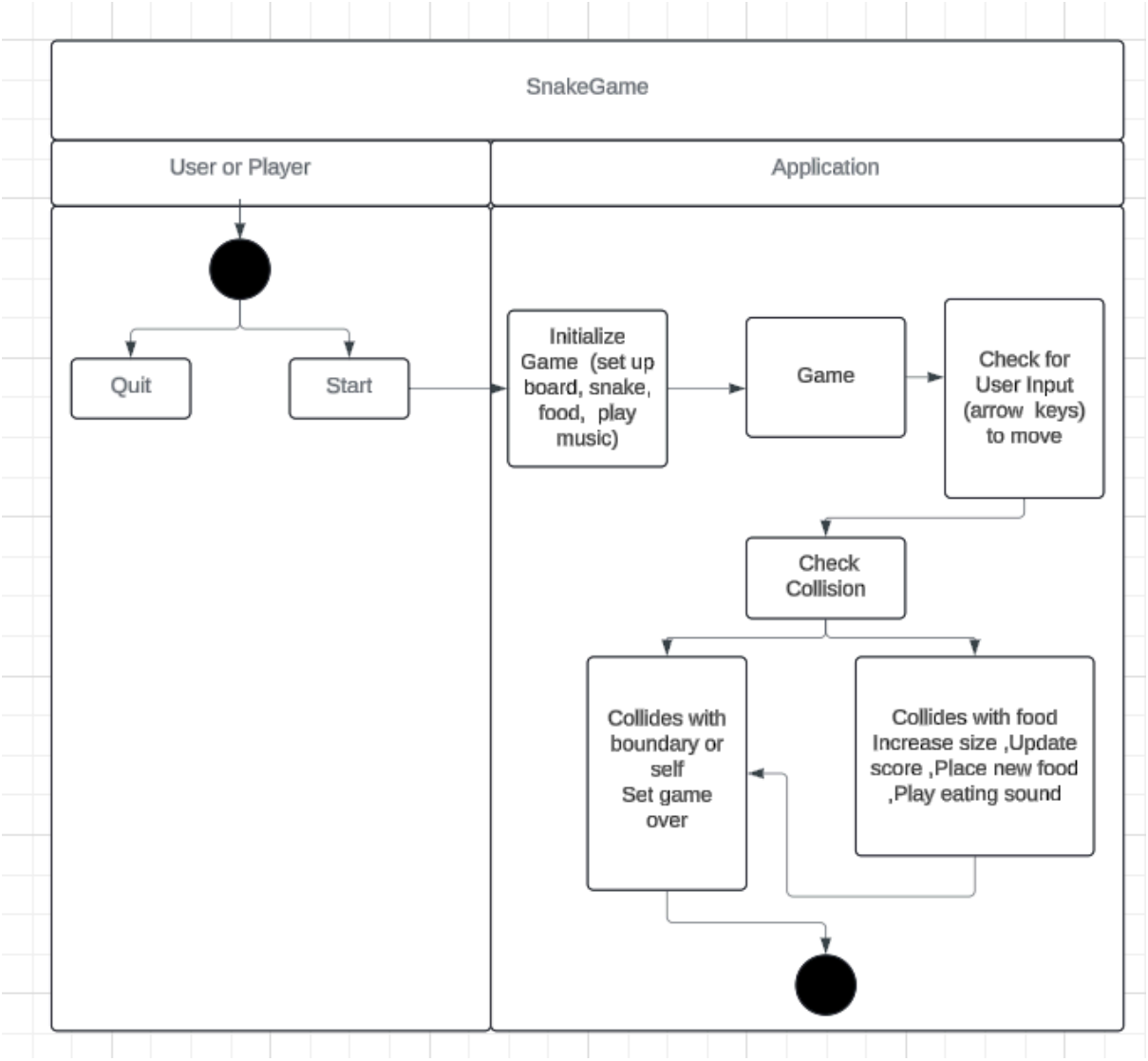


Figure 3.1

Chapter 4 - Class Diagram

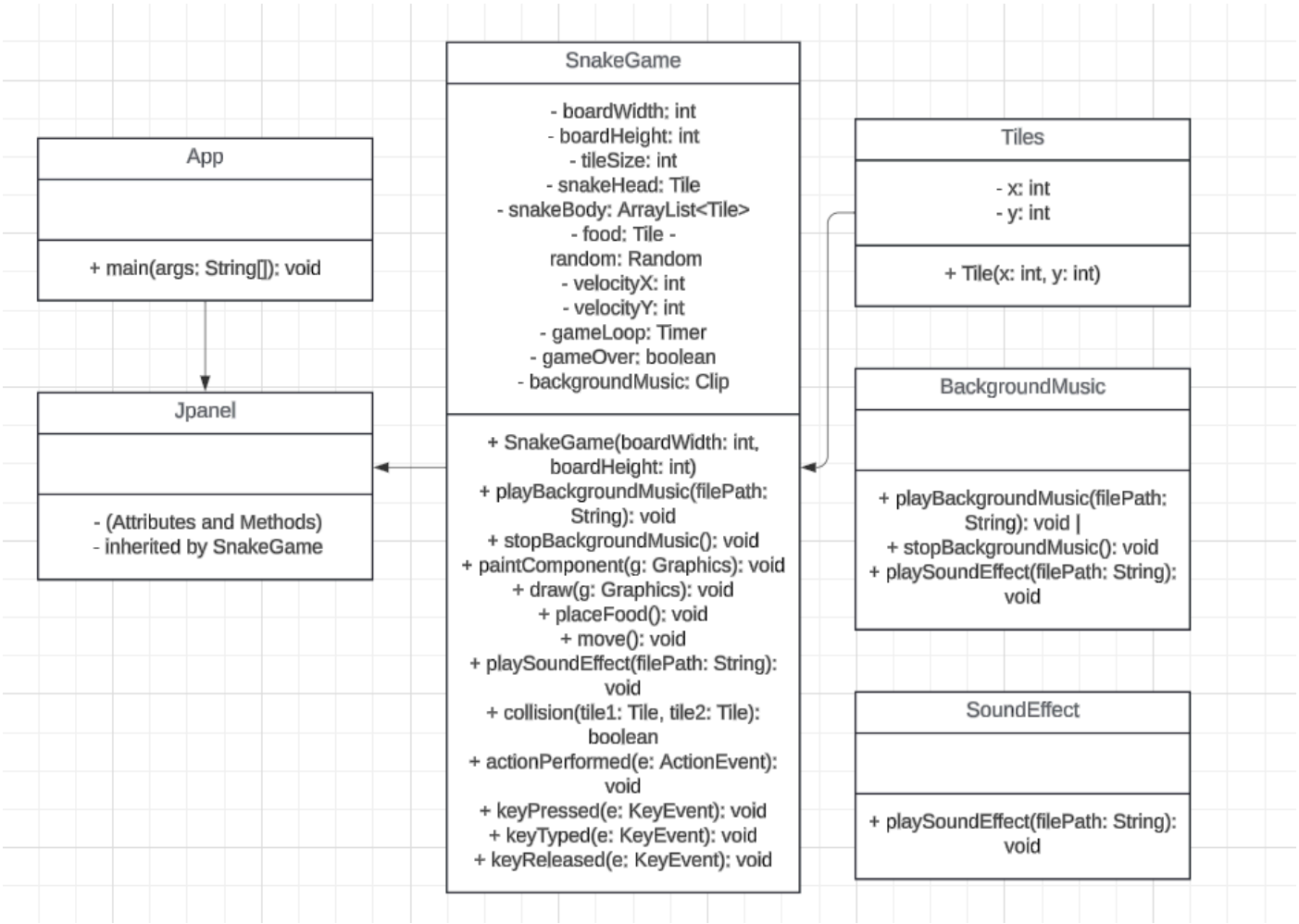


Figure 4.1

Chapter 5 - Modules Used

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.ArrayList;
```

Figure 5.1

SnakeGame.java

- javax.swing: For creating GUIs (e.g., JFrame, JPanel, JButton).
- java.awt: Provides basic graphics and GUI components (e.g., Graphics, Color, Font).
- java.awt.event: Handles events like button clicks and key presses (e.g., ActionListener, KeyListener).
- java.util.ArrayList: Resizable array for storing dynamic data (e.g snake body parts).
- java.util.Random: Generates random numbers (e.g., for placing food).
- javax.sound.sampled: For playing and manipulating sound files (e.g., AudioInputStream, Clip, AudioSystem).
- java.io.File: Represents file and directory pathnames (e.g., loading sound files).

Chapter 6 - Essential Algorithm

The essential algorithm in this SnakeGame class revolves around the game logic, including movement, collision detection, and updating the game state.

6.1 Action Performed

```
public void actionPerformed(ActionEvent e) {
    move();
    repaint();
    if (gameOver) {
        gameLoop.stop();
        stopBackgroundMusic();
    }
}
```

Figure 6.12

This method is invoked by the Timer at regular intervals. It's the core of the game loop. move() updates the game state by moving the snake, checking for collisions, and updating the score. repaint() refreshes the graphical representation of the game board. If gameOver is true, the game loop stops, and the background music is stopped.

6.2 KeyPressed

```
public void keyPressed(KeyEvent e) {
    if (e.getKeyCode() == KeyEvent.VK_UP && velocityY != 1) {
        velocityX = 0;
        velocityY = -1;
    } else if (e.getKeyCode() == KeyEvent.VK_DOWN && velocityY != -1) {
        velocityX = 0;
        velocityY = 1;
    } else if (e.getKeyCode() == KeyEvent.VK_LEFT && velocityX != 1) {
        velocityX = -1;
        velocityY = 0;
    } else if (e.getKeyCode() == KeyEvent.VK_RIGHT && velocityX != -1) {
        velocityX = 1;
        velocityY = 0;
    }
}
```

Figure 6.21

This method handles user input for controlling the snake's direction. It's triggered when a key is pressed. Depending on the pressed key (up, down, left, right), the snake's velocity is updated accordingly. The snake cannot reverse its direction instantly (e.g., if it's moving up, it cannot immediately move down).

6.3 Move

```
public void move() {
    if (collision(snakeHead, food)) {
        snakeBody.add(new Tile(food.x, food.y));
        placeFood();

        // Play the eat sound effect
        playSoundEffect( filePath: "C:\\Users\\USER\\OneDrive\\Documents\\Binus
    }

    for (int i = snakeBody.size() - 1; i >= 0; i--) {
        Tile snakePart = snakeBody.get(i);
        if (i == 0) {
            snakePart.x = snakeHead.x;
            snakePart.y = snakeHead.y;
        } else {
            Tile prevSnakePart = snakeBody.get(i - 1);
            snakePart.x = prevSnakePart.x;
            snakePart.y = prevSnakePart.y;
        }
    }

    snakeHead.x += velocityX;
    snakeHead.y += velocityY;

    for (Tile snakePart : snakeBody) {
        if (collision(snakeHead, snakePart)) {
            gameOver = true;
        }
    }
```

Figure 6.31

```
if (snakeHead.x < 0 || snakeHead.x >= boardWidth / tileSize ||
        snakeHead.y < 0 || snakeHead.y >= boardHeight / tileSize) {
    gameOver = true;
}
```

Figure 6.32

This method updates the game state by moving the snake and handling collisions. If the snake collides with food, it grows, and new food is placed. The snake's body is moved

segment by segment, with each segment taking the position of the segment before it. The snake's head is moved based on its velocity. Collisions with the snake's body or the wall result in a game over.

Chapter 7 - Screenshot of The Project

1. The Gameplay

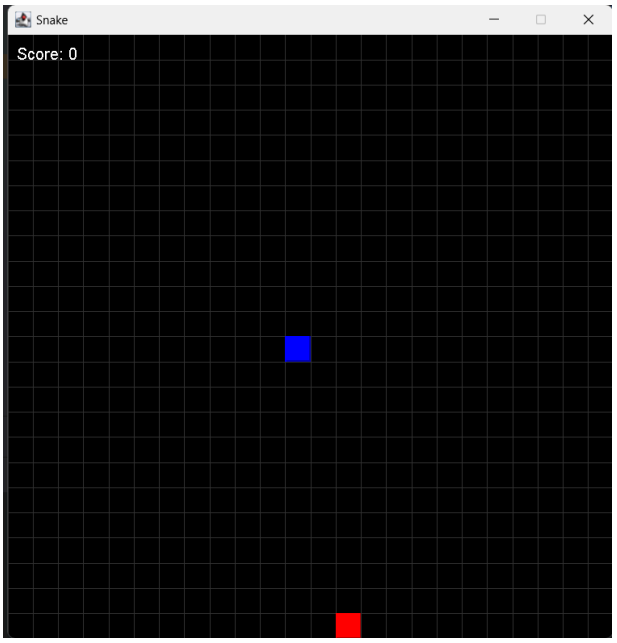
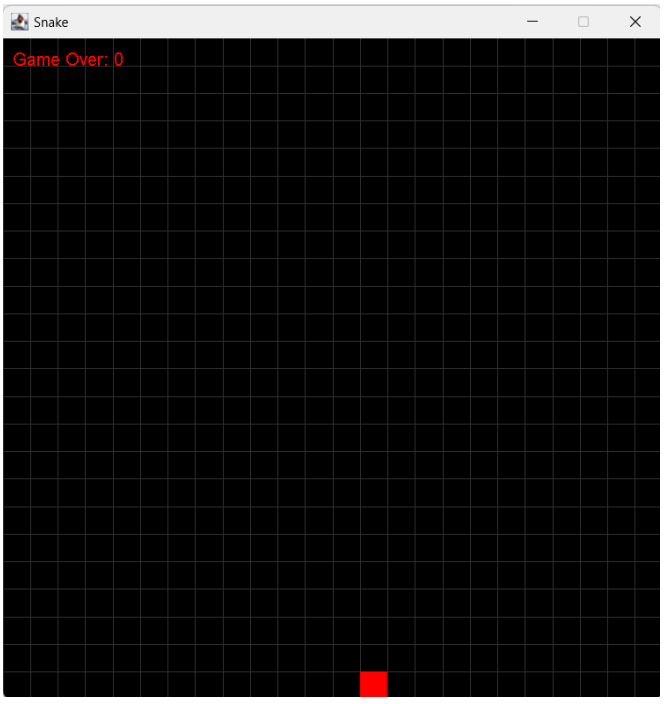


Figure 7.1

2. Game Over



Chapter 8 - Lesson Learned

Implementing the Snake game project provided valuable lessons. Modular design and code organization were crucial for managing complexity, while event-driven programming highlighted the importance of responsiveness. Handling collision detection and game state management emphasized robust error handling and thorough testing. Overall, the project underscored the significance of structured design, user interaction, and error management in software development.

Chapter 9 - Source Code and Poster

Githhub = https://github.com/rlaxNstdy/OPP_FinalProject

Poster =

https://www.canva.com/design/DAGHvU4r5tg/vcyG9_7-8YSlF0PqFbpEzw/edit?utm_content=DAGHvU4r5tg&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton