**Mini Project: Snake Game Made Using Java**

**By Group no. 2 – AIML (C)**

|  |  |
| --- | --- |
| 1. Shreya Shahi | 6. Pawan Shetty |
| 2. Aayan Shaikh | 7. Rishith Shetty |
| 3. Ayan Shaikh | 8. Aryan Shinde |
| 4. Saaem Shaikh | 9. Rohit Shinde |
| 5. Amankumar Sharma | 10. Anmol Shukla |

**An Innovative Examination (IE) Report**

**Submitted for the**

**Subject of**

**Professional Skills**

**Under the Guidance of**

**Ms. Priya Salunke**

**Asst. Professor**

**A.Y. 2024-25**

**Abstract:**

**The Snake game is a classic arcade game where the player controls a snake that grows longer as it consumes food, while avoiding collisions with the walls or itself. This project involves the development of the Snake game using Java, focusing on object-oriented principles, event handling, and GUI (Graphical User Interface) components. The game includes features such as score tracking, dynamic difficulty levels, and game-over conditions. This report outlines the design, development, and testing of the game.**

**Introduction:**

**The Snake game has been a popular game since its inception in the late 1970s. With the advancement of technology, it has transitioned from basic cell phones to more complex versions with enhanced graphics and interactivity. The aim of this project is to recreate a simple version of the Snake game using Java to demonstrate knowledge of Java programming, graphical user interface design, and event-driven programming.**

**In this report, we will describe the design decisions, functionality, and challenges encountered while developing the game.**

**Background:**

Java provides a robust framework for creating interactive applications, including games. In this project, we utilized Java’s Swing library for the graphical user interface, along with basic object-oriented design principles for organizing game logic. The game involves a snake that grows longer as it eats food, and the player must avoid hitting the walls or the snake’s body.

**System Description:**

**Block**

* **Game Interface**: Handles the display and user input (e.g., arrow keys).
* **Game Logic**: Controls the snake's movement, collision detection, score tracking, and game state.
* **Main Game Loop**: Continuously updates the game state (e.g., snake’s position, food) and renders the updated frame.

**Module**

* **Snake Class**: Represents the snake’s body, movement, and growth.
* **Food Class**: Represents the food that the snake consumes to grow.
* **Game Board Class**: Manages the game’s grid, collision detection, and rendering.
* **Game Class**: Controls the main game loop, user input, and overall game state.

**Results and Discussions:**

After implementing the Snake game in Java, several important features were tested and verified:

* **Movement**: The snake moves in four directions (up, down, left, right) based on user input. The movement is smooth and responsive.
* **Game Over**: The game ends when the snake collides with the wall or itself.
* **Score Tracking**: The score increases as the snake eats food, and it is displayed on the screen.
* **Dynamic Difficulty**: The snake’s speed increases gradually as the score increases.

**Screenshots**: Included below with the anexures

* Main game screen with the snake and food
* Score display
* Game over screen with final score

**Conclusion and Future Scope:**

The Snake game project was successfully implemented using Java, demonstrating key concepts in game development, such as event handling, object-oriented design, and GUI design. The game is simple yet engaging, providing a foundation for future enhancements.

**Future Scope**:

* **Multiplayer Mode**: Adding a multiplayer mode where two players can control their snakes on the same board.
* **Improved Graphics**: Enhancing the visual appearance with better graphics and animations.
* **Sound Effects**: Implementing sound effects for actions like eating food, collisions, or game-over.

**Acknowledgment:**

**We extend our gratitude to Ms. Priya Salunke for her invaluable guidance in the study of programming using Java. Her insights and challenging questions significantly enhanced this work. Special thanks to Thakur College of Engineering & Technology for providing access to research resources and facilities essential for this project.**

**Annexures (Code):**

The size of each of the joints of a snake is 10 px. The snake is controlled with the cursor keys. Initially, the snake has three joints. When the game is finished, the "Game Over" message is displayed in the middle of the board.

First, we will define the constants used in our game.

**A black background with white text

Description automatically generated**

The B\_WIDTH and B\_HEIGHT constants determine the size of the board. The DOT\_SIZE is the size of the apple and the dot of the snake. The ALL\_DOTS constant defines the maximum number of possible dots on the board (900 = (300\*300)/(10\*10)). The RAND\_POS constant is used to calculate a random position for an apple. The DELAY constant determines the speed of the game.



These two arrays store the x and y coordinates of all joints of a snake.

A screenshot of a computer program

Description automatically generated

In the loadImages() method we get the images for the game. The ImageIcon class is used for displaying PNG images.

A screen shot of a computer program

Description automatically generated

In the initGame() method we create the snake, randomly locate an apple on the board, and start the timer.

A computer screen with white text

Description automatically generated

If the apple collides with the head, we increase the number of joints of the snake. We call the locateApple() method which randomly positions a new apple object.

In the move() method we have the key algorithm of the game. To understand it, look at how the snake is moving. We control the head of the snake. We can change its direction with the cursor keys. The rest of the joints move one position up the chain. The second joint moves where the first was, the third joint where the second was etc.

A black background with white text

Description automatically generated

This code moves the joints up the chain.

A black background with white text

Description automatically generated

This line moves the head to the left. In the checkCollision() method, we determine if the snake has hit itself or one of the walls.

A black background with white text

Description automatically generated

If the snake hits one of its joints with its head the game is over.

A black background with white text

Description automatically generated

The game is finished if the snake hits the bottom of the board.

A screenshot of a computer program

Description automatically generated

This is the main class.



The setResizable() method affects the insets of the JFrame container on some platforms. Therefore, it is important to call it before the pack() method. Otherwise, the collision of the snake's head with the right and bottom borders might not work correctly.

A screenshot of a computer

Description automatically generated

**References**

* **Java Documentation (**[**https://docs.oracle.com/javase/8/docs/**](https://docs.oracle.com/javase/8/docs/)**)**
* **"Java: A Beginner's Guide" by Herbert Schildt**
* **Stack Overflow (for troubleshooting and solution implementation)**