Java Snake Game: Project Report

Team Snake Game

August 5, 2025

Contents

1 Project Overview

Our team of three developed a classic Snake Game using Java Swing as an academic project to demonstrate proficiency in object-oriented programming (OOP), GUI development, and event handling. The game involves a snake controlled by user input or an AI agent, navigating a grid to eat apples, grow in length, and avoid collisions with itself or the boundaries. This report outlines the modules used, Java imports, OOP principles applied, keyboard input mechanisms, GUI implementation, AI training model, and how these components integrate to create a functional game.

2 Modules Used

The project is organized into five Java files, each serving a specific purpose:

- SnakeGame.java: The main class that initializes the game window and launches the application.
- GamePanel.java: The core game panel that manages the game loop, rendering, user input, and AI control.
- Snake.java: Represents the snake, handling its movement, growth, and direction.
- Apple.java: Manages the apple's position and spawning logic.
- SnakeAI.java: Implements a Deep Q-Learning model to train an AI agent to play the game, using an Experience class to store state-action-reward transitions.

These modules create a modular and maintainable codebase, with each class encapsulating specific functionality.

3 Header Files (Java Imports)

The following Java packages and classes are imported to support the game's functionality:

- javax.swing.* (SnakeGame.java, GamePanel.java): Provides GUI components like JFrame, JPanel, and JButton for the game window and interface.
- java.awt.* (GamePanel.java): Supplies graphics tools (Graphics, Color, Font) and event handling (ActionListener, KeyAdapter).
- java.awt.event.* (GamePanel.java): Enables keyboard and action event handling (ActionEvent, KeyEvent).
- java.util.Random (Apple.java): Used for generating random positions for the apple.
- org.deeplearning4j.* (SnakeAI.java): Provides neural network functionality for the AI training model.
- org.nd4j.* (SnakeAI.java): Supports numerical computations for the neural network.

These imports provide the necessary tools for GUI rendering, event handling, random number generation, and AI training, ensuring the game operates smoothly.

4 Object-Oriented Programming (OOP) Implementation

We applied several OOP principles to ensure a robust and scalable design:

- Encapsulation:
  - In Snake.java, the snake's state (position arrays x, y, bodyParts, direction) is encapsulated as instance variables, with methods like move(), grow(), and setDirection() controlling access and updates.
  - In Apple.java, the apple's coordinates (x, y) are encapsulated, with spawnApple() managing their updates.
  - In SnakeAI.java, the neural network and replay memory are encapsulated, with methods like chooseAction() and train() managing AI behavior.
- Abstraction:
  - The GamePanel class abstracts the game loop, rendering, and AI control logic, exposing only high-level methods like startGame(), restartGame(), and enableAIControl().
  - The Snake, Apple, and SnakeAI classes abstract their respective behaviors, allowing GamePanel to interact with them without needing to know their internal implementations.
- Modularity:
  - Each class has a single responsibility: SnakeGame for window setup, GamePanel for game logic, Snake for snake behavior, Apple for apple spawning, and SnakeAI for AI training.
  - This separation allows easy maintenance and potential extensions, such as adding new features or modifying existing ones.
- Inheritance and Polymorphism:
  - GamePanel extends JPanel and implements ActionListener, inheriting Swing's panel functionality and overriding methods like paintComponent() and actionPerformed().
  - The inner class MyKeyAdapter extends KeyAdapter, overriding keyPressed() to handle keyboard input.

  These principles ensure the codebase is organized, reusable, and easy to understand.

5 Keyboard Input Handling

Keyboard input is handled to control the snake's direction using both arrow keys and W/A/S/D keys when not in AI mode:

- Implementation: The GamePanel class includes an inner class MyKeyAdapter that extends KeyAdapter. The keyPressed(KeyEvent e) method captures key events and maps them to snake directions:
  - Left arrow or A: Sets direction to 'L'.

    - Right arrow or D: Sets direction to 'R'.
    - Up arrow or W: Sets direction to 'U'.
    - Down arrow or S: Sets direction to 'D'.
  - Logic: The setDirection(char newDir) method in Snake ensures the snake cannot reverse directly (e.g., from left to right), preventing invalid moves.
  - Integration: The GamePanel constructor adds the key listener (this.addKeyListener(new MyKeyAdapter())). The panel is set to be focusable (this.setFocusable(true)) to receive keyboard events. After a game over, restartGame() calls requestFocusInWindow() to ensure input is captured.

This approach provides a responsive and intuitive control scheme for human players.

6 GUI Implementation

The game's graphical user interface is built using Java Swing:

- Window Setup (SnakeGame.java):
  - Creates a JFrame titled "Snake Game".
  - Adds a GamePanel, sets it as non-resizable, and centers it using setLocationRelativeTo(null).
  - Uses pack() to fit the frame to the panel's preferred size (600x600 pixels).
- Game Rendering (GamePanel.java):
  - Overrides paintComponent(Graphics g) to call draw(Graphics g), which renders game elements.
  - The draw method:
    - Draws the apple as a red oval using fillOval().
    - Draws the snake, with the head in green and body in a darker green using fillRect().
    - Draws the score using drawString().
  - The gameOver method displays "Game Over" and the final score, showing a "Play Again" button.
- Interactive Elements:
  - A JButton ("Start Game") is displayed on the home screen to start the game.
  - A JButton ("Pause/Resume") toggles the game state.
  - A JButton ("Play Again") is centered at game over and triggers restartGame() when clicked.
  - A JLabel displays controls information on the home screen.

    - The panel uses a black background and a grid-based rendering system (25x25 pixel units).

The GUI provides a clear and engaging visual experience, with real-time updates driven by a Timer.

7 AI Training Model

To enhance the game, we implemented an AI agent using Deep Q-Learning (DQN) to learn how to play the Snake game autonomously. The AI is implemented in the SnakeAI class, leveraging the Deeplearning4j library for neural network functionality.

- State Representation: The game state is represented as a vector including:
    - Relative position of the apple to the snake's head (e.g., is the apple left, right, above, or below).
    - Immediate dangers (e.g., is a collision with the body or boundary imminent in each direction).
    - The snake's current direction to prevent invalid moves.
- Actions: The AI chooses from three actions (left, right, forward) relative to the current direction, avoiding direct reversal into the snake's body.
- Reward System:
    - +10 for eating an apple, triggering growth and respawning.
    - -100 for collisions with the body or boundaries, ending the game.
    - +1 for moving closer to the apple, -1 for moving away, based on Manhattan distance.
- Neural Network: A multi-layer perceptron with input layer (state size), hidden layers, and output layer (three Q-values for actions) is trained using Deeplearning4j. The network uses a learning rate of 0.001 and gamma (discount factor) of 0.99.
- Training Process:
    - Uses an epsilon-greedy strategy (initial $\epsilon = 1.0$, decaying to 0.1) for exploration vs. exploitation.
    - Stores experiences (state, action, reward, next state, done) in a replay memory (Experience class).
    - Trains on mini-batches (size 32) sampled from replay memory to update Q-values.
- Integration: The GamePanel includes an aiControlled flag. When enabled, actionPerformed() calls SnakeAI.chooseAction() to set the snake's direction, stores experiences, and triggers training.

This AI model allows the snake to learn optimal strategies over time, improving its ability to collect apples and avoid collisions.

8 How It All Works Together

The components integrate seamlessly to create a functional game with both human and AI control:

- Initialization: SnakeGame creates the JFrame and adds GamePanel, which initializes the game state (startGame()) with a new Snake, Apple, and optionally SnakeAI.
- Game Loop: GamePanel uses a Timer (100ms delay) to trigger actionPerformed(), which:
  - Calls Snake.move() to update the snake's position.
  - Invokes checkApple() to detect apple collisions, triggering Snake.grow() and Apple.spawnApple() if needed.
  - Calls checkCollisions() to detect game-over conditions.
  - For AI mode, calls SnakeAI.chooseAction() to set the snake's direction, computes rewards, stores experiences, and triggers SnakeAI.train().
  - Triggers repaint() to update the GUI.
- User Input: When not in AI mode, keyboard events are captured by MyKeyAdapter, updating the snake's direction via Snake.setDirection().
- AI Control: When aiControlled is true, the AI determines the snake's direction based on the current state and learned policy.
- Rendering: The draw method renders the snake, apple, score, or the game-over screen if the game ends.
- Restart Mechanism: The "Play Again" button calls restartGame(), resetting the game state, restarting the timer, and optionally reinitializing the AI.

This modular design ensures each component handles a specific aspect of the game, with clear interactions between classes. The use of OOP principles, event-driven programming, Swing's GUI capabilities, and the DQN-based AI results in a cohesive, playable, and intelligent game.