ISSN: 2664-9640

Development of Snake Game Based on Java Code

Zhengyang Lu

Beijing Wuzi University, Beijing, China

Abstract

With the rapid development of computer science and technology, programmers continuously explore new methods to develop exciting and entertaining applications that enhance our lives. This paper focuses on a classic game - Snake, which is simple yet challenging and has received widespread acclaim. It also delves into a popular programming language - Java. Java is widely used in the field of software development, known for its cross-platform compatibility, object-oriented features, and rich standard library, making it suitable for various application developments. The primary objective is to explore how to develop a feature-rich and distinctive Snake game based on Java algorithms and evaluate the feasibility of game development using Java. We will delve into the game development process, including game architecture, user interface design, game logic, and more. By analyzing factors such as the difficulty of implementation and entertainment value of the game, we will further examine the feasibility of Java algorithms.

Keywords

Java algorithms, Snake game, game development, feasibility.

1. Introduction

1.1. Snake Game

Computer games have continuously evolved and expanded, becoming a realm filled with creativity and innovation. The Snake Game is a classic, straightforward, and addictive game that dates back to the early days of computers, with its origins tracing back to the 1970s. In this game, players control a snake with the goal of increasing its length by consuming food while avoiding collisions with boundaries or the snake's own body. Although the gameplay of Snake appears simple, it challenges players' reaction time and strategic thinking.

1.2. Java Programming Language

Simultaneously, the Java programming language, as a versatile, cross-platform programming language, plays a vital role in the field of computer science. Java's features include object-oriented programming, memory management, cross-platform compatibility, and a powerful standard library, making it an attractive choice for game development.

In this context, the development of a Snake game based on Java algorithms has become an intriguing and challenging task. It not only involves game design and programming skills but also necessitates a deep understanding of algorithms and performance optimization to ensure smooth gameplay.

2. Overview of the Snake Game

2.1. Game Rules and Objectives

Game Rules:The player controls a snake's movement within a bounded area, consuming food to increase its length while avoiding collisions with boundaries or itself. The game's objective is to prolong the snake's length as much as possible and accumulate a high score.

(2) Entertainment Value and History

The Snake Game's popularity stems from its simple yet challenging rules. Over time, the game has become increasingly challenging, testing players' reflexes and strategies. Its straightforward and entertaining gameplay rules have established it as a classic. Furthermore, the Snake Game has a long history, cherished by players since the 1970s, making it an iconic creation in the world of gaming.

3. Game Development

3.1. Achievement of Goals

Based on our set objectives to develop a feature-rich Snake game, we need to implement the following elements:

3.1.1 Implement an initial menu for starting the game, viewing scores, and selecting difficulty levels.

3.1.2 Implement a logical loop to enable the snake's movement.

3.1.3 Control the snake's direction and manage game start, restart, etc., through key inputs.

3.1.4 Automatically generate obstacles (in this game's development, obstacles increase in number after consuming a certain amount of food) and food items.

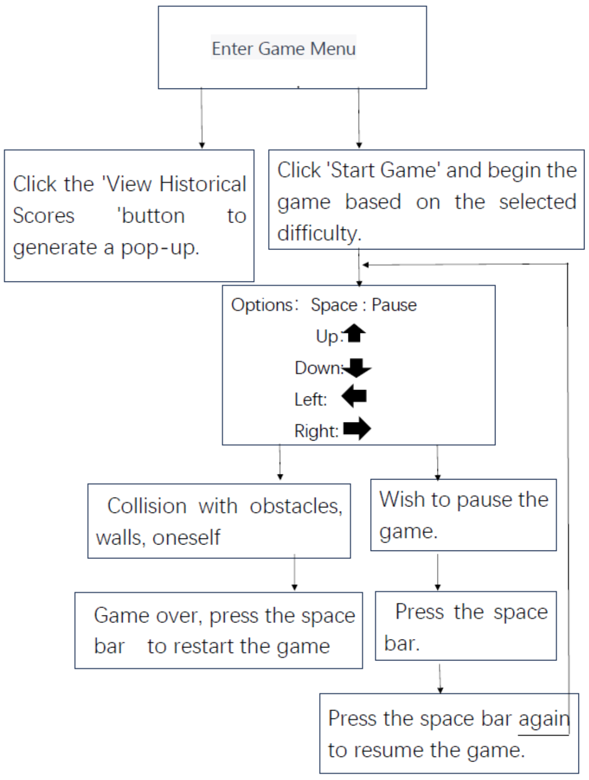
3.1.5 Include sound effects for death, food consumption, and background music.

3.1.6 Incorporate a score management system that allows score input and output.

3.1.7. Implement collision detection mechanisms, including snake growth upon food consumption, game over on collisions with obstacles and walls.

3.2. Game Flowchart:

In line with these objectives, we have designed a simple flowchart as follows:


Enter Game Menu
Click the 'View Historical Scores 'button to generate a pop-up.
Click 'Start Game' and begin the game based on the selected difficulty.
Options: Space : Pause
Up:
Down:
Left:
Right:
Collision with obstacles, walls, oneself
Wish to pause the game.
Game over, press the space bar to restart the game
Press the space bar.
Press the space bar again to resume the game.

4. Key Classes in Game Development

4.1. SnakeGameMenu Class:

The SnakeGameMenu class is the main menu interface of the game. It inherits from the JFrame class and is used to display the game's menu options. The game menu interface includes the "Start Game" button, "View High Scores" button, and a difficulty selection dropdown.

Presentation of Key Code:

```
//Create class SnakeGameMenu
public class SnakeGameMenu extends JFrame implements ActionListener {
//Set Main Menu Style.
    setTitle("SnakeGame"); // Set Window Title.

 setSize(400, 200); // Set Window Size.
……… // Here, some code is omitted for setting buttons and dropdowns.
// Add an event listener to the start game button.
startGameButton.addActionListener(this){
if (e.getSource() == startGameButton)// Using conditional statements to determine the code to execute based on the user's click.
   …………// Launch different classes (programs) based on the buttons clicked by the user.

}
```

4.2. Snake1 and Snake2 Class

4.2.1 When the user clicks the 'Start Game' button, create different game objects (Snake1 or Snake2) based on the selected difficulty and launch the corresponding game interface.

4.2.2.Using 'import javax.sound.sampled.*', allow the Snake game to have different sound effects for various game states, such as end, in progress, pause, etc.

4.2.3.You can also use if statements to conditionally add
relevant code, such as:

```
File deathSoundFile = new File("deathsound.wav");
// The death sound file name, this code is added when the game state is 'game over' to generate sound effects.
```

4.2.4 Using Java.awt.Graphics to set font size and font color to fulfill requirements for displaying scores and providing various hints, such as:

Press the 'Spacebar' to start the game

4.2.5 Set the initial position of the snake and use a for loop to implement snake movement. For example:

```
// When the state is 1 (game in progress), execute a for loop.
 if (started == 1) {
     for (int i = slong - 1; i > 0; i--) {
    Snakex[i] = Snakex[i - 1];
   Snakey[i] = Snakey[i - 1];
     }
```

Continuously change the coordinates, allowing the snake to move, and wait for the next command.

And use the ‘void keyPressed(KeyEvent e)’ method to allow the snake to change direction accordingly based on the user's input of the up, down, left, or right arrow keys.

Code example:

```
public void keyPressed(KeyEvent e) {
   int key = e.getKeyCode();

   if (key == KeyEvent.VK_SPACE) {
      //Handling the operation of the space bar, including starting the game, pausing the game,
or restarting the game.} else if (key == KeyEvent.VK_LEFT) {
      // Handling the left arrow key operation, such as changing the snake's movement direction.}
else if (key == KeyEvent.VK_RIGHT) {
      //Handling the right arrow key operation.} else if (key == KeyEvent.VK_UP) {
      // Handling the up arrow key operation.} else if (key == KeyEvent.VK_DOWN) {
      // Handling the down arrow key operation.}
}
```

4.2.6 Implement a collision detection mechanism. The snake moves by changing its coordinates, and you can achieve this by restricting the range of coordinate movement, implementing boundary collision detection. In other words, define that the snake's coordinates cannot cross a certain threshold in any direction, or else it will be considered a collision with the boundary, leading to the end of the game. For example:if (Snakex[0] < 0 || Snakex[0] > 780 || Snakey[0] < 80 || Snakey[0] > 580)

```
{
      started = 2;}// Game over when hitting the boundary.
```

Collision detection with oneself, obstacles, and food can be implemented by iterating through the coordinates of the snake's head and comparing them one by one with the coordinates of obstacles and food. This allows the snake to grow when it hits food and ends the game when it collides with itself or obstacles. The basic code framework is as follows:

```
// Example of collision detection method in the game logic module.
public boolean checkCollision() {
// Implementation methods for collision detection, including boundary collisions, self-
collisions, food collisions, and obstacle collisions.
  // Returning true indicates a collision has occurred, resulting in either the end of the game or
an increase in score.
   // Returning false indicates no collision.
}
```

Based on this, write the corresponding code.

4.3. ScoreManager Class

This class is used to set up a scoring system, implementing input and output of scores. At the end of the game, it records the current length of the snake and subtracts the initial length to calculate the score for the current game. Externally, you can set up a text file or database, and then use Java's input and output methods to write the score into it for future retrieval.

5. Development Results

5.1. Overview of the Overall Architecture:

- The game is implemented with a graphical interface based on Java Swing.

- The game has a main menu interface `SnakeGameMenu` for starting the game and viewing high scores.
- The game logic is primarily implemented in the `Snake1` class, including game element initialization, collision detection, and timed updates.
- The game includes sound effects and background music.
- The game has different implementations for different levels of difficulty, and the specific game logic to run is determined by the selected difficulty.

5.2. Data Structure Design

5.2.1. `Snake1` and `Snake2` Classes: These two classes represent the core logic of the Snake game, including snake movement, food, obstacles, and more. Within these classes, `snakex` and `snakey` arrays represent the snake's x and y coordinates, `foodx` and `foody` represent the food coordinates, and `obstacleXArray` and `obstacleYArray` represent the obstacle coordinates.

5.2.2 `SnakeGameMenu` Class: This class represents the game menu, which includes elements such as the "Start Game" button and a dropdown for selecting difficulty levels. It may utilize data structures to manage the components in the menu, such as JButton and JComboBox.

5.2.3 `SnakeGamePanel` Class: This class represents the game interface, including the snake's state, food, obstacles, and more. It employs data structures to represent elements within the game interface, such as the snake's coordinates, food coordinates, etc. Additionally, it may use a timer (Timer) to control the game's refresh rate.

5.2.4 Other Data Structures: In addition to the data structures within the mentioned classes, the code likely utilizes standard Java data structures such as arrays, strings, integers, etc., to store and process game state information, scores, and other related data.

5.3. User Interface Design

Game Menu Interface Design

5.3.1 In the game menu, we focus on user choices for game difficulty, starting the game, and viewing high scores.

5.3.2 Title: Display the game title "Snake Game" at the top of the window.

5.3.3 Start Game: Add a "Start Game" button that allows players to initiate the game. Below the button, provide a dropdown for selecting difficulty levels, such as "Easy" or "Hard."

5.3.4 View High Scores: Include a "View High Scores" button that, when clicked, displays historical score records. You can use a text box or a pop-up dialog to show this information.

5.4. Game Interface Design

The game interface is the primary area where players interact with the game, and it includes the game canvas and score display.

Game State and Score: Display the game state, such as "Game in Progress" or "Game Over," along with the current score, at the top or bottom of the game canvas.

Game Over Information: When the game ends, show a message at the top, informing the player of their final score.

5.5. User Input Handling

The user input handling module is a core component of the Snake game, responsible for receiving, interpreting, and responding to the user's keyboard input. This module implements the KeyListener interface to listen for keyboard events.

5.6. Game Logic

The game logic module is the core of the Snake game, responsible for managing the game's execution, rules, and state. This module controls the snake's movement, food generation, score calculation, and game state transitions.

5.7. Module Functions

The main functions of the game logic module include:

5.7.1 Game Initialization: At the start of the game, the module initializes the game state, including the snake's initial position, food spawn location, score, and game state settings.

5.7.2 Snake Movement Control: Based on user input and game rules, the module controls the snake's movement, including updating the snake's position, collision detection, and boundary checks.

5.7.3 Food Generation: The module is responsible for generating food on the game canvas and ensuring that it does not collide with the snake's body or obstacles.

5.7.4 Score Calculation: Based on the number of food items consumed by the snake, the module calculates the player's score and displays it on the game interface.

5.7.5 Game Over Detection: The module checks if the game is over by evaluating whether game over conditions have been met, such as collisions with boundaries, the snake's own body, or obstacles.

5.7.6 Game State Management: Based on the game's progress, the module manages transitions between game states, including the game running, paused, or ended.

5.8. Collision Detection

The collision detection module is one of the core components of the Snake game, responsible for monitoring and determining whether collisions occur between different objects in the game. These objects include the snake and the boundaries, the snake and itself, the snake and food, as well as the snake and obstacles.

The primary functions of the collision detection module include:

5.8.1 Boundary Collision Detection: Checking if the snake's head collides with the game's boundaries. If the snake's head goes beyond the boundaries, the game is considered a failure, triggering the end of the game.

5.8.2 Self-Collision Detection: Monitoring whether the snake's head collides with any part of its own body. If the snake's head overlaps with its body, the game ends.

5.8.3 Food Collision Detection: Determining if the snake's head collides with food, indicating that the snake has consumed it. This triggers an increase in the score and the regeneration of food.

5.8.4 Obstacle Collision Detection: Inspecting whether the snake's head collides with obstacles within the game. If the snake's head hits an obstacle, the game ends.

The collision detection module accomplishes its tasks by iterating through the coordinates of objects, including the snake's head, the snake's body, food, and obstacles, as well as boundary conditions to identify collisions. When a collision is detected, the module responds accordingly, triggering actions like ending the game or increasing the score.

5.9. Conclusion

This paper discussed the development process of a Snake game based on Java algorithms, including game architecture, user interface design, game logic, collision detection, and other key components. By using the Java programming language, we were able to create a cross-platform and entertaining Snake game, providing players with entertainment and challenges. Although there were some challenges during the development process, such as collision detection and game state management, these issues could be resolved through proper design and programming practices. The difficulty was not very high, and all the elements needed for the game were successfully implemented.

Snake is considered a relatively complex game, and using it as an example, it is evident that the existing Java libraries are sufficient for developing somewhat complex games. At least 2D games can be easily implemented, and they can meet the players' requirements for entertainment. This demonstrates that developing games using the Java programming language is feasible.

Future work could include further optimizing game performance, adding more game elements and features, improving the user interface, and adapting the game for different platforms.

References

[1] Li Zhenjun, Cheng Liangyu. Analysis and Implementation of the Method for Java Mobile Game Development Based on MIDP. Computer Applications, 2004(S1): 3. DOI: CNKI:SUN:JSJY.0.2004-S1-094.

[2] Li Bin. Characteristics of the Java Language. Management Observation, 1998(3): 1.

[3] Su Zhitong, Shi Shaokun, Li Jinhong. Research on the Architecture of Mobile Game Development. Computer Engineering and Design, 2010(7): 4. DOI: CNKI:SUN:SJSJ.0.2010-07-064.