Snake Game Documentation

John Doe

May 20, 2023

Contents

1 Introduction

This documentation provides an overview and detailed explanation of the Snake Game implemented in Java using the Processing 4.2 IDE. The game consists of several components, including the Snake class, Controller class, and the main sketch that controls the game environment. Each component has specific functionalities and methods to facilitate the game mechanics.

2 Installation

To run the Snake Game, you need to have the Processing 4.2 IDE installed on your system. Processing is an open-source graphical library and integrated development environment built for creating visual art, animations, and interactive applications.

To install Processing 4.2, follow these steps:

1. Visit the Processing website at `https://processing.org/`.
2. Download the appropriate version of Processing for your operating system.
3. Install Processing by following the installation instructions provided on the website.

Once you have installed Processing, you can run the Snake Game by opening the provided Java code file in the Processing IDE and clicking the "Run" button.

3 Code Structure

The Snake Game code consists of three main parts: the Snake class, the Controller class, and the main sketch.

3.1 Part 1: Snake Class

The Snake class represents the snake object in the game. It handles the snake's movement, growth, collision detection, and rendering.

3.1.1 Properties

The Snake class has the following properties:

- `pos`: An array of `PVector` objects representing the positions of the snake's body segments. Each segment is a vector with x and y coordinates.
- `vel`: A `PVector` object representing the current velocity of the snake. It determines the direction in which the snake moves.
- `total`: An integer representing the total number of food items eaten by the snake.
- `justAte`: A boolean indicating whether the snake has just eaten a food item. It is set to `true` when the snake eats food and is reset to `false` after updating the snake's position.
- `controller`: An instance of the `Controller` class for controlling the snake's movement and search algorithms.

3.1.2 Methods

The Snake class has the following methods:

- `update()`: Updates the position of the snake based on its velocity and checks for collision with the boundaries or itself. It moves the snake forward by adding the velocity vector to each body segment's position.
- `render()`: Renders the snake on the screen. It draws rectangles representing each segment of the snake's body.
- `dir(int x, int y)`: Sets the velocity of the snake based on the given direction. It takes two integers (-1, 0, or 1) as parameters to set the x and y components of the velocity vector.
- `eatFood()`: Checks if the snake has eaten a food item by comparing its head position with the food position. If the positions match, it increments the `total` count and returns `true`; otherwise, it returns `false`.
- `search()`: Performs the search algorithm to find the shortest path to the food item. It calls the search methods of the `Controller` class to determine the search path.

3.2 Part 2: Controller Class

The Controller class handles the control and logic of the game.

3.2.1 Properties

The Controller class has the following properties:

- `mainSearch`: An array of `PVector` objects representing the main search path. It stores the positions of the cells visited during the search algorithm.
- `longestPath`: An array of `PVector` objects representing the longest path. It stores the positions of the cells in the longest path from the snake's current position to the food.
- `inLongestPath`: A boolean indicating whether the snake is in the longest path mode. It is set to `true` when the main search path is empty, indicating that the snake is trapped and needs to find the longest path.

3.2.2 Methods

The Controller class has the following methods:

- `renderMainSearch()`: Renders the main search path on the screen. It draws lines connecting the positions of the cells in the main search path.
- `dijkstraSearch()`: Performs Dijkstra's search algorithm to find the shortest path from the snake's current position to the food. It uses a priority queue to explore the neighboring cells and determines the shortest path by keeping track of the minimum distances.
- `longestPathSearch()`: Finds the longest path from the snake's current position to the food. It uses a modified depth-first search algorithm to explore all possible paths and chooses the one with the maximum length.

3.3 Part 3: Main Sketch

The main sketch represents the game environment and controls the game flow.

3.3.1 Variables

The Main Sketch has the following variables:

- `fps`: An integer representing the frame rate of the game.
- `horSqrs`: An integer representing the number of horizontal squares on the game board.
- `verSqrs`: An integer representing the number of vertical squares on the game board.
- `scl`: An integer representing the scale factor for rendering the game elements.
- `bgcol`: A `color` object representing the background color of the game.
- `gridcol`: A `color` object representing the color of the grid lines.
- `snakecol`: A `color` object representing the color of the snake.
- `foodcol`: A `color` object representing the color of the food.
- `searchcol`: A `color` object representing the color of the search path.
- `shortpathcol`: A `color` object representing the color of the shortest path.
- `longpathcol`: A `color` object representing the color of the longest path.
- `notRenderSearchKey`: A boolean indicating whether to show the search path.
- `renderingMainSearch`: A boolean indicating whether the main search path is being rendered.
- `gamePaused`: A boolean indicating whether the game is paused.
- `justDijkstra`: A boolean indicating whether to use only Dijkstra's search.
- `snake`: An instance of the `Snake` class representing the snake object.
- `food_pos`: A `PVector` object representing the position of the food item.

3.3.2 Functions

The Main Sketch has the following functions:

- `settings()`: Sets the size of the game window.
- `setup()`: Initializes the game environment, including the grid, snake, and food item.
- `draw()`: Updates and renders the game state.
- `grid(color col)`: Renders the grid on the game board with the specified color.
- `updateFood()`: Updates the position of the food item on the game board.
- `renderFood()`: Renders the food item on the game board.

- `isOutsideWorld(PVector pos)`: Checks if the given position is outside the game boundaries.
- `keyPressed()`: Handles key press events for controlling the game, including toggling search modes, pausing the game, and adjusting the frame rate.

4 Conclusion

The Snake Game implemented in Java using the Processing 4.2 IDE provides a fun and interactive gaming experience. By understanding the code structure and the functionalities of each component, you can further customize and enhance the game according to your preferences.