Santa 2048 README

How the Program Works

Our program is an implementation of the 2048 puzzle game using Java and the Swing library for the graphical user interface.
Our program is divided into two main classes: Santa2048Logic and Santa2048GUI.
The Santa2048Logic class handles the core logic for 2048 and includes a 2D array (using double arrays) representing the game board, where each cell can hold a tile with a set value. This class includes methods for sliding tiles (slideUp, slideDown, slideLeft, slideRight), combing tiles (combineUp, combineDown, combineLeft, combineRIght), and performing moves (moveUp, moveDown, moveLeft, moveRight). The methods for performing moves utilize a sequence of sliding tiles, then combining tiles if possible, and then sliding tiles again. If an arrow key is pressed, tiles will first shift to the farthest possible empty cell in the board in the corresponding row or column. Then, the program checks if any combinations can be made. Next, the program shifts any tiles combined in the given direction.
This class also includes methods to check if the game is over (isGameOver) or if the player has won (isGameWon). These methods return a boolean value that is used to determine the game state and the appearance of game end screens.
This class also includes a method to add a new tile of value 2 or 4 to a random empty cell each time a move is made (addRandomTile). Another method (resetGame) resets the game state.
In addition, the Santa2048Logic class has a scoring system that is implemented. At the start of a new game, the score is initialized to 0 in the Santa2048Logic constructor. When two tiles with the same value are merged from a move, the score increases by the value of the merged tile. For example, if two tiles with a value of 8 are merged, the player’s score increases by 8 points.
The Santa2048GUI class is responsible for the graphical representation of the game, primarily using Java Swing components.
The class extends JPanel and implements KeyListener to capture user input. User input from arrow keys triggers corresponding methods in the Santa2048Logic class, updating the GUI accordingly with repaint().
The GUI includes JPanel objects which are used to represent the title screen, an end screen, and a win screen.
The drawScore method in the Santa2048GUI class displays the current score in the window, rendered in a set color and font.
This class also includes the paintComponent method draws the title, game board, tiles, and options at different points in the game.

How to Use the Program

The program is a game GUI, so using it should be fairly simple. When you run the program, the title screen pops up. The title screen contains (1) the title, (2) how to play, and (3) the start button to play the game. The instructions tell the user to use the arrow keys on the keyboard.
To start playing the game, the start game button needs to be pressed. The user then sees a grid with two squares. These squares can be either 4s or 2s. Each arrow key shifts all the squares left, right, up, or down. In doing so, any of the same tiles along the shift are combined and

added together, creating a newer, higher tile. The objective of the game is to retrieve the 2048 tile.
If the user loses, he is greeted with the losing screen. The user then has two options: (1) return to the main menu or (2) restart the game and try again. If the user hits restart, the game is reset and they are given another chance to beat it.

Challenges

There were many, many challenges in developing this program. However, the largest challenge came from designing the logic to move tiles. When the user picks a direction to move their tiles by using the arrow keys, the game has to take into account the value of the tiles to calculate the score, where the tiles are, where they have to be shifted, and whether or not they combine with a tile in the shifting direction. Creating the logic was very difficult for these key event listeners. In the end, we landed on double int arrays. The double int arrays can store a grid of values, which is perfect for 2048, a game that takes place in a grid. We then use for loops going across columns and rows from left and right and up and down (all depending on the key) as well as conditional statements to generate a whole new board. We also use similar logic for the combination of tiles.

Another thing that was difficult was creating the logic to indicate game over. Since there are multiple factors to consider like if the board is full and if there are the same tiles next to each other (meaning there is a valid move left), translating this into logic with a double array proved to be difficult.

Concerns

There are major concerns about the fairness of the game. While the game functions almost always normally, if the user hits an arrow key to shift the tiles a certain way and the tiles are already shifted in that direction, then the program registers it as a move and generates new tiles despite there being no change in the layout. This feature is not in the original game of “2048.” While this concern is reasonable, we think that this difference in tile generation simply creates a different way of playing the game. So, while our program adds tiles when the original game doesn’t, we think it is only a difference in how the programmer wants the game to run. However, the real concern comes with all the other key presses. At the moment, our program generates tiles no matter which button you click. This is a major area for improvement in our program. While we think that a new tile generated from the arrow keys, even if it does not shift any tiles, is valid, we do not believe *every* click should generate a new tile. This could allow the user to simply generate all tiles without combining any together.