Homework 1: Simplified Candy Crush Game

November 27, 2023

Introduction

In this assignment, you will develop a simplified version of the popular game Candy Crush using Java. Your game will feature a grid of colored cells, where players match three cells of the same color to clear them from the board.

Task

1. **Initialize the Game Board:** Randomly fill the board with colored cells. Ensure no matches are present at the start. For this assignment, you should represent the game board as a two-dimensional array. You have the flexibility to decide the type of the array based on what you think is most appropriate for managing the colored cells. For instance, you can use a 2D array of integers, characters or Strings depending on your implementation strategy.

2. **Game Loop:**
   - Check for and clear any matches.
   - If no moves are possible, reshuffle the board.
   - Accept player input for cell selection and direction of swap.
   - Process the swap and update the board.

3. **Scoring:** Implement a scoring system based on matches and any additional criteria you choose.

4. **Game Modes:** Implement two game modes that are randomly chosen at the start of the game:
   - *Mode 1:* Collect 200 points in 15 moves.
   - *Mode 2:* Collect 100 points for each color in 20 moves.

   The game ends when the move count reaches zero, and the program should print the collected points and remaining moves after each move.

Method Descriptions

In this section, a list of suggested methods is provided to help you structure your program. However, you are not required to strictly adhere to these method signatures. Feel free to modify the parameters, change the method signatures, add more methods, or use fewer methods as per your design and implementation strategy. The goal is to create a functional game; the structure provided here is a guideline, not a strict requirement.

- **initializeBoard()**: Create and randomly fill the board ensuring no initial matches.
- **printBoard()**: Display the current state of the game board.
- **checkForMatches()**: Check and clear matches after each move.
- **isMovePossible()**: Determine if any moves are left that can lead to a match.
- **shuffleBoard()**: Reshuffle the board if no moves are possible.
- **swapCells(int x, int y, String direction)**: Swap cells based on player input.
- **updateBoard()**: Update the board after each move.
- **calculateScore()**: Update the player's score based on cleared cells.
- **checkWinCondition(int gameMode)**: Check if the player meets the win condition.
- **printGameStatus()**: Print current score and remaining moves.
- **play()**: The main method for game loop execution.

Bonus Points and Colored Output

Bonus Points for Moving Cells Down

In your implementation, include a feature where players can receive bonus points for moving upper cells to the lower positions on the board. Specifically, if a player's move causes cells to fall down and fill cleared spaces, they should receive an additional 15 points for each such move. This adds an extra layer of strategy to the game, encouraging players to plan moves that cause cascades of matches.

Colored Output for Additional Points

To enhance the visual appeal of your game, use the following lines of code to display colored cells in the console:

```
static String red = Colors.RED + "R" + Colors.RESET;
static String green = Colors.GREEN + "G" + Colors.RESET;
static String blue = Colors.BLUE + "B" + Colors.RESET;
```

When you use ‘System.out.print(red);‘, for example, the output will be colored red. Implementing this colored output feature in your game will earn you an additional 5 points. This not only makes your game more visually appealing but also demonstrates your ability to use advanced console output features in Java.

Play Example

Mode 1:

```
Welcome to the game!
Enter 's' to start the game or 'q' to quit:
s
You have to collect 200 points in 15 moves!
Good luck!
Enter the size of the matrix:
4 5
You have 15 moves to reach the goal!
|B|R|G|B|B|
|R|G|R|R|B|
|B|B|G|G|R|
|B|R|B|G|R|
Enter the cell:
1 1
Enter the direction:
up
Invalid move!
|B|R|G|B|B|
|R|G|R|R|B|
|B|B|G|G|R|
|B|R|B|G|R|
Enter the cell:
2 4
Enter the direction:
right
Clearing Board:
|B|R|G|B|B|
|R|G|R|B|R|
|B|B|G|G|R|
|B|R|B|G|R|
--------------
|B|R|G|B|G|
|R|G|R|B|R|
|B|B|G|G|G|        // seen another match
|B|R|B|G|B|
--------------
|B|R|B|R|G|      // new generated colors
|R|G|G|B|G|
|B|B|R|B|R|
|B|R|B|G|B|
You have collected 20 points and you have 14 moves left!
```

Mode 2:

```
Welcome to the game!
Enter 's' to start the game or 'q' to quit:
s
You have to collect 100 points for each color in 20 moves!
Good luck!
Enter the size of the matrix:
4 5
You have 20 moves to reach the goal!
|B|R|G|B|B|
|R|G|R|R|B|
|B|B|G|G|R|
|B|R|B|G|R|
Enter the cell:
1 1
Enter the direction:
up
Invalid move!
|B|R|G|B|B|
|R|G|R|R|B|
|B|B|G|G|R|
|B|R|B|G|R|
Enter the cell:
2 4
Enter the direction:
right
Clearing Board:
|B|R|G|B|B|
|R|G|R|B|R|
|B|B|G|G|R|
|B|R|B|G|R|
--------------
|B|R|G|B|G|
|R|G|R|B|R|
|B|B|G|G|G|        // seen another match
|B|R|B|G|B|
--------------
|B|R|B|R|G|      // new generated colors
|R|G|G|B|G|
|B|B|R|B|R|
|B|R|B|G|B|
You have collected 10 points for red, 10 points for green, 0 points for blue and
you have 19 moves left!
```

This process continues until all the moves are played or the player completes the mission. At the end, the program should announce whether the player won the game or not.

Submission

Submit your source code with comprehensive comments explaining your implementation.