Offline Coding Task – A Popular Game

1. Preliminaries and rules

Please proceed in an **incremental way** using **baby steps**! **Ensure** that at any point in time, your solution **can be executed** and **add small features** one by one. In this way, the time limit will not hinder you to deliver at least a partial solution – which is very acceptable! Other requirements:

- The chosen programming language has to be Typescript.
- Backend - The game engine has to be written inside a Node.JS API using Express.JS. framework
- Share the code over a git repository of your choice.
- The purpose of the challenge is to evaluate generic coding skills and it's up to the developer how much time he wants to invest to show these skills.

2. Definition of Done

The definition of done for any feature is:

- Unit tests are given that demonstrate the correctness of your code
- The code adheres to high quality standards (as few lines of code as possible), best practices and a clear structure.
- The resulting software is functional and can be tested by us.
- The game engine works as expected

3. Requirements

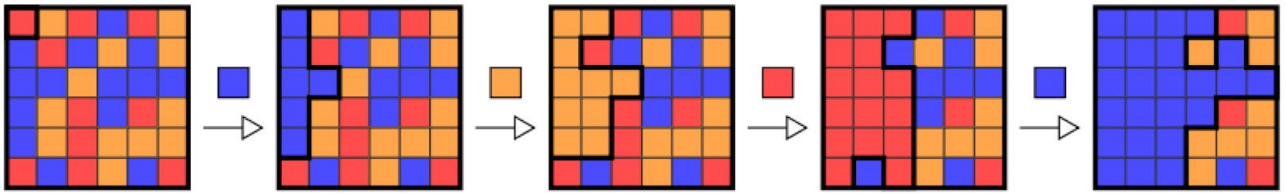
Today we are going to implement a popular one player game. The player is given an $n \times n$ board of tiles where each tile is given one of m colors. Each tile is connected to up to four adjacent tiles in the North, South, East, and West directions. A tile is connected to the origin (the tile in the upper left corner) if it has the same color as the origin and there is a path to the origin consisting only of tiles of this color. A player makes a move by choosing one of the m colors. After the choice is made, all tiles that are connected to the origin are changed to the chosen color. The game proceeds until all tiles of the board have the same color. The goal of the game is to change all the tiles to the same color, preferably with the fewest number of moves possible.

It has been proven that finding the optimal moves is a very hard computational problem. It has also been shown that finding the minimum number of flooding operations is NP-hard for $m > 3$. This even holds true when the player can perform flooding operations from any position on the board. However, this variant can be solved in polynomial time for the special case of $m = 2$. For an unbounded number of colors, even this variant remains NP-hard for boards of a dimension of at least $n = 3$ and is solvable in polynomial time for boards of dimension $n = 2$.

For your solution, you will implement a very simple greedy strategy to solve it:

- for each move, choose the color that will result in the largest number of tiles
- connected to the origin;
- if there is a tie, break ties by choosing the color that has the lowest rank among the colors.

The following figure depicts a possible sequence of moves together with the chosen color on a 6 x 6 board initially filled up by 3 distinct colors.



Please implement the game and an automated player that determines the color choice for each move. We want to determine the amount of moves and the sequence of the colors chosen by the player over the course of a game. Please use unit tests to prove that your code is working properly!

If you can implement the simple greedy algorithm quickly, you might provide an improved solution using a modified A* algorithm.

4. Front-End - Optional point

Ideally the game should have an UI where the player can play by choosing every time a colour. At the end of the game, the player can compare his steps with the “AI player”.

The UI can be a very minimalistic table of maximum 10x10 and a colour picker and the display of history of moves at the end of the game, comparing the moves provided by the “AI player”

Depending on the time spent on the previous points, it is also fine to discuss the approach during the on-premise interview.