BACK

Tetris Game

SCORE: 300/300

80,354

DESCRIPTION | DRAFTS | SOLUTIONS | COMMENTS | README

Medium **Codewriting** **300**

Let's play Tetris! But first we need to define the rules, especially since they probably differ from the way you've played Tetris before.

There is an empty field with `20` rows and `10` columns, which is initially empty. Random pieces appear on the field, each composed of four square blocks. You can't change the piece's shape, but you can rotate it `90` degree clockwise (possibly several times) and choose which columns it will appear within. Once you've rotated the piece and have set its starting position, it appears at the topmost row where you placed it and falls down until it can't fall any further. The objective of the game is to create horizontal lines composed of `10` blocks. When such a line is created, it disappears, and all lines above the deleted one move down. The player receives `1` point for each deleted row.

Your task is to implement an algorithm that places each new piece *optimally*. The piece is considered to be placed *optimally* if:

- The total number of blocks in the rows this piece will occupy after falling down is maximized;
- Among all positions with that value maximized, this position requires the least number of rotations;
- Among all positions that require the minimum number of rotations, this one is the leftmost one (i.e. the leftmost block's position is as far to the left as possible).

The piece can't leave the field. It is guaranteed that it is always possible to place the Tetris piece in the field.

Implement this algorithm and calculate the number of points that you will get for the given set of `pieces`.

Example

Example

For

```
pieces = [[[".", "#", "."],
          ["#", "#", "#"]],
         [["#", ".", "."],
          ["#", "#", "#"]],
         [["#", "#", "."],
          [".", "#", "#"]],
         [["#", "#", "#", "#"]],
         [["#", "#", "#", "#"]],
         [["#", "#"],
          ["#", "#"]]]
```

the output should be

`tetrisGame(pieces) = 1`.

For this explanation, we are representing each block by the index of the piece it belongs to. After the first 5 blocks fall, the field looks like this:

```
...
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . 3 4
. . . . . . . . 3 4
. 0 . 1 . 2 2 . 3 4
0 0 0 1 1 1 2 2 3 4
```

Note that the 0^{th}, 1^{st}, and 2^{nd} pieces all fell down without rotating, while the 3^{rd} and the 4^{th} pieces were rotated one time each.

Since there is now a row composed of `10` blocks, it is deleted, and you get `1` point.

When the last piece falls, the final field looks like this:

```
...
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
5 5 . . . . . 3 4
5 5 . . . . . 3 4
. 0 . 1 . 2 2 . 3 4
```

Input/Output

- **[execution time limit] 4 seconds (js)**
- **[input] array.array.array.char pieces**

  A non-empty array of pieces in the order in which they fall. Each piece is represented as a rectangular matrix, where `'#'` represents a block and `'.'` represents an empty cell.

  Each piece consists of `4` blocks, and each block shares a common side with at least one another block. It's guaranteed that each piece contains neither empty rows nor empty columns.

  Guaranteed constraints:

  `3 ≤ pieces.length ≤ 30`,

  `1 ≤ pieces[i].length ≤ 2`,

  `2 ≤ pieces[i][j].length ≤ 4`.

- **[output] integer**

  The number of points you will have by the end of the game.

[JavaScript (ES6)] Syntax Tips

```
// Prints help message to the console
// Returns a string
function helloWorld(name) {
    console.log("This prints to the console when you Run Tests");
    return "Hello, " + name;
}
```