DM Programmation Avancée

Coding Game : Number of paths between two points

Nom Etudiant : Nicolas Antunes

Pseudonyme CodingGame : Astuna

Github :
https://github.com/Nestuna/LG_Prog_Avancee/blob/master/DM_NombreDeChemins/Solution.java

Code :

```
import java.util.*;
import java.io.*;
import java.math.*;

/**
 * Auto-generated code below aims at helping you parse
 * the standard input according to the problem statement.
 **/
class Solution {

    public static void afficheMatrice(int[][] matrice)
    {
        // Formatage d'un affichage compréhensible de la matrice
        for(int i = 0 ; i < matrice.length ; i++)
        {
            for (int j = 0 ; j < matrice[i].length ; j++)
                System.err.format("%6d", matrice[i][j]);
            System.err.println();
        }
    }

    public static int[][] initMatrix(int[][] matrix) {
        // On remplit les bords de la matrice, en prennant soin de vérifier
        // qu'il n'y a pas de mur (1)
        // S'il y a un mur, on met la valeur 0 (ce n'est pas un chemin possibl
e),
        // sinon 1 (c'est un chemin possible)

        matrix[0][0] = 1;
        for(int i = 1; i < matrix.length; i++) {

            if(matrix[i][0] == 0 && matrix[i - 1][0] != 0) {
                matrix[i][0] = 1;
```

```
            }
            else if(matrix[i][0] == 1) {
               matrix[i][0] = 0;
            }
        }

        for(int j = 1; j < matrix[0].length; j++){
            if(matrix[0][j] == 0 && matrix[0][j - 1] != 0) {
              matrix[0][j] = 1;
            }
            else if(matrix[0][j] == 1) {
              matrix[0][j] = 0;
            }
        }

        return matrix;
     }

    public static int nbOfPaths(int[][] matrix) {
        // Utilise la même logique que le Triangle de Pascal
        // Calcule le nombre de chemins possible d'un bord opposé à l'autre
        //
        // parametre : une matrice du parcours
        // return : le resultat de la case d'arrivée

        // On initialise les bords de la matrice
        matrix = initMatrix(matrix);

        // On calcule maintenant les autres chemins,
en s'inpirant de la formule de Pascal:
        // S'il y a un mur, on met 0 (ce n'est pas calculé comme un chemin pos
sible)
        // Sinon on applique la formule, en additionnant la case au dessus et
la case de gauche:
        // m[n][m] = n[n][m-1] + m[n-1][m]
        //


        for(int i = 1; i < matrix.length; i++) {
            for(int j = 1; j < matrix[0].length; j++) {
                if(matrix[i][j] == 1) {
                    matrix[i][j] = 0;
                }
                else if(matrix[i][j] == 0) {
                    matrix[i][j] = matrix[i][j-1] + matrix[i-1][j];
                }
            }
        }
```

```
        // DEBUG : Affiche la matrice des calculs de chemins
        System.err.println("Matrice pascalée : ");
        afficheMatrice(matrix);

        return matrix[matrix.length - 1][matrix[0].length - 1];
    }

    public static void main(String args[]) {
        Scanner in = new Scanner(System.in);
        int M = in.nextInt();
        int N = in.nextInt();
        if (in.hasNextLine()) {
            in.nextLine();
        }

        int[][] matrix = new int[M][N];

        // On lit chaque ligne et on les caste pour les affecter à une matrice
        for (int i = 0; i < M; i++) {
            String ROW = in.nextLine();
            for(int j = 0; j < N; j++) {
              matrix[i][j] = Integer.parseInt(String.valueOf(ROW.charAt(j)));

            }
        }


        // Write an answer using System.out.println()
        // To debug: System.err.println("Debug messages...")

        // DEBUG : Affiche la matrice du parcours
        System.err.println("Matrice du parcours");
        afficheMatrice(matrix);

        // RESULTAT !
        System.out.println(nbOfPaths(matrix));
    }
}
```