Green Vs Red

1. Introduction

The task was to write a program for a game (the rules you can find below), that accepts:

- the size of the grid – x and y (x being the width and y being the height)
- **y** lines containing strings created by 0s and 1s – each string represents a row from the grid
- x1 and y1 **–** coordinates of a given apple
- N – generations count

The program must calculate how many times the cell with coordinates x1 and y1 was green from Generation Zero until Generation N.

Two classes are realized – class GenerationMatrix and class Apple. Every time the user enters the input a matrix with y rows and x columns is created. For each cell the program generates an object from class Apple (the color is determined by the string – every character from the string is converted to int and represents the color of the given apple).

2. Classes

class Apple

This class is realized to store the color of each cell (apple). The class has one attribute - `int color;`. The color is determined by the string the user inputs.

class GenerationMatrix

The matrix is represented by two 2-dimentional arrays of Apples. To calculate how many times the cell with coordinates x1 and y1 was green from Generation Zero until Generation N is realized method `void generations(int N)`.

- Attributes

```
Apple** generationMatrix;
Apple** temp;
```

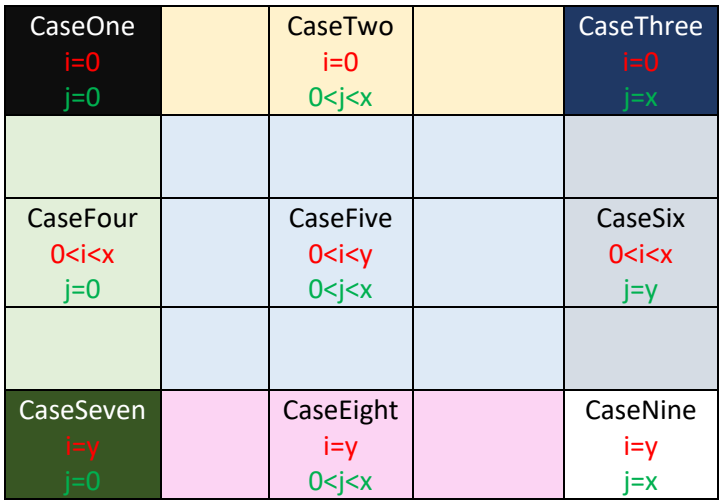
To realize the method `void generations(int N)` are used two 2-dimentional arrays. The second array (temp) is necessary because each time the method is called the cells in `generationMatrix` are modified. Using the two matrices, each time after the color of a cell is changed (`void changeAppleColor(int i, int j, int greenNeighboursCount)`) the `generations` method calls the `void switchMatrices()` method where the `generationMatrix` is equated to `temp` and the next time we call `changeAppleColor` (for the next cell) the program works with the current matrix (current matrix=current generation). After the last cell is modified by `generations` all of the cells are now modified and the current generation is over.

- Methods
  - `void generations(int N);`

The method uses 2 helper methods - `void changeAppleColor(int i, int j, int greenNeighboursCount)` and `void switchMatrices()`. Two loops are used to modify each cell depending on their coordinates. The integer `greenNeighboursCount` stores the green neighbors count. To find the count of the adjacent cells with green apples we need to check all of the cells surrounding the current one. Nine helper methods are used – each method is called depending on the coordinates of the cell.

```
int caseOne(int i, int j) const;
int caseTwo(int i, int j) const;
int caseThree(int i, int j) const;
int caseFour(int i, int j) const;
int caseFive(int i, int j) const;
int caseSix(int i, int j) const;
int caseSeven(int i, int j) const;
int caseEight(int i, int j) const;
int caseNine(int i, int j) const;
```

CaseOne i=0 j=0	CaseTwo i=0 0<j<x	CaseThree i=0 j=x
CaseFour 0<i<y j=0	CaseFive 0<i<y 0<j<x	CaseSix 0<i<x j=x
CaseSeven i=y j=0	CaseEight i=y 0<j<x	CaseNine i=y j=x



- `void changeAppleColor(int i, int j, int greenNeighboursCount);`

This method checks if a cell must change its color – if it is green and has 0, 1, 4, 5, 7 or 8 green adjacent cells, the cell turns red. Otherwise it stays green. If the cells is red and has 3 or 6 green adjacent cells, the cell turns green. Otherwise it stays red. The counter is increased each time a cell stays or becomes green.

Green vs. Red

'Green vs. Red' is a game played on a 2D grid that in theory can be infinite (in our case we will assume that **x** <= **y** < **1 000**).

Each cell on this grid can be either green (represented by 1) or red (represented by 0). The game always receives an initial state of the grid which we will call 'Generation Zero'. After that a set of 4 rules are applied across the grid and those rules form the next generation.

Rules that create the next generation:

1. Each red cell that is surrounded by exactly 3 or exactly 6 green cells will also become green in the next generation.

2. A red cell will stay red in the next generation if it has either 0, 1, 2, 4, 5, 7 or 8 green neighbours.

3. Each green cell surrounded by 0, 1, 4, 5, 7 or 8 green neighbours will become red in the next generation.

4. A green cell will stay green in the next generation if it has either 2, 3 or 6 green neighbours.

Important facts:

- Each cell can be surrounded by up to 8 cells. 4 on the sides and 4 on the corners. Exceptions are the corners and the sides of the grid.
- All the 4 rules apply at the same time for the whole grid in order for the next generation to be formed.

Your Task:

Create a program that accepts:

The size of our grid - x, y (**x** being the **width** and **y** being the **height**)

Then the next **y** lines should contain strings (long **x** characters) created by **0s** and **1s** which will represent the 'Generation Zero' state and help us build the grid.

The last arguments to the program should be coordinates (x1 and y1) and the number N.

(x1 and y1) will be coordinates of a cell in the grid. We would like to calculate in how many generations from Generation Zero until generation N this cell was green. (The calculation should include generation Zero and generation N)

Print your result in the console.

Special requirement: Write your game in a way that uses several classes. This will show OOP knowledge and will account for more points during the evaluation. Comments, good naming convention and documentation are also recommended.

Example1:

```
# 3x3 grid, in the initial state, the second row is all 1s, how many times will the cell [1, 0] (top center) become green in 10 turns?
3, 3
000
111
000
1, 0, 10
# expected result: 5
```

Example2:

```
# 4x4 grid. Input:
4, 4
1001
1111
0100
1010
2, 2, 15
# expected result: 14
```