CMPE 150 Fall 2015

Project III

Mustafa Enes ÇAKIR ***2013400105***

Project Description:

In this project, we need a board of color. It has 16x16 characters, and outside of board is O. So we can fill edges of our board with O. 18x18 character array enought for us. Each cell connected to top,bottom, left and right cells. Cells will changed according to some conditions about neighbour cells. G, B, P, and C are dark colors; W, O, Y, and L are light colors. If a DC is connected to 4 DC, it becomes to B.

If a DC is connected to 3 LC, or 2 LC and at least 1 P, or 1 LC and at least 2 P it becomes to P. If a DC is not B or P it becomes C. If a W is connected to at least 1 O it becomes O. If an O is connected to at least to 2 Y and at most 1 O, or 1 Y and at least 2 DC, or at least 2 DC and at least 1 O it becomes to Y. When last 2 board configurations are same, you should finalize your board, and change remained W with L. When you increase row number by 1, you check bottom cell; or when you decrease column number by 1, you check left cell. You need to check conditions one by one. When you relabel one cell, go to next one. Initial and explored, boards and statics should be printed.

Project Solution:

Totally I used 16 *for* loops, and 1 while loop. Mostly I used them for coming over each character on board.

I divide my program to 7 *static methods* in addition to my *main* method. Each method responses some part of the project. They make easy to debug my program.

I keep my board in an 18x18 character array. Its name is *board*. In additon to that, I keep my previous board in *oldBoard* variable, and set it each time before change board. I create them in my *main* method. First of all, I will explained my other methods, and come back my main method.

My *readFile* method reads input text and put it in to an array. But first, It fills all array with O. After that it put characters that in text file to inner part of the board. It doesn't touch outer O's, because they are represent outside. Text files contains 16x16 board, anyway our *board* array is 18x18. It read line by line with Scanner and transfer them to our program. It takes the board that we wanted to put characters in

it, and file name as a parameter.

My *readFile* method prints current board configuration; count and print number of each character with desired text to the beginning of the line. First 2 nested for loops comes over every character on our board configuration. i and j starts with 1 and ends with length-1 because edges of our board array filled with O. It represents outside. We don't need to print and count them. End of the method last for loop goes through every index on counter and chars array. If number of character is not equal to zero, prints it.

My *isDC* method checks given character is DC or not. It takes a character as parameter. It returns result as an boolean. I added that method because I need to check this condition a lot of times in my main method. Instead of writing this checking algorithm all the time, easily I used my method.

My *countConnectedDC* method calculate number of DC connected to given position and return it. It takes board and position indexs as parameters. It returns number of connected DC as an integer. It looks 4 sides of given position. If one of them is DC increase *count* variable by one. I added that method because I need to check this condition a lot of times in my main method. Instead of writing this checking algorithm all the time, easily I used my method.

My *countConnectedChar* method calculate number of the given character connected to given position and return it. It takes character that we want to count, board and position indexs as parameters. It returns number of given character as an integer. It looks 4 sides of given position. If one of them is matched with given charater increase *count* variable by one. I added that method because I need to check this condition a lot of times in my main method. Instead of writing this checking algorithm all the time, easily I used my method.

My *duplicate* method duplicates our board to new variable and return new variable. Nested for loops comes over every character on our board configuration and copy them to new board variable. I need to this method because, Equal sign(=) doesn't work with arrays. It is set to same adress of the array, not contents of them. When you use '=' with array they refer same part of memory. When you change one of them, other is changed too.

My *isEqualArray* method checks 2 board are equal or not, and return result as boolean. Nested for loops comes over every character on our board configuration and if characters on that position aren't equal, return false. If for loops didn't return false, it means boards are equal and return true. I need to this method because, Assign sign(==) doesn't work with arrays. It checks equality of memory adresses.

Let's return my *main* method. I read my *board* from text and print it as "INITIAL" with my methods. Using a while method, my programs relabel characters until last two boards are same. At the beginning of the while loop, clone *board* to *oldBoard*, and start to change current board. With 2 nested for loops comes over every character on our board configuration, and check their relabeling condition. My other

methods make easier my work. Nested If-else tree checks conditions. “at least 1 P” means in my program is countConnectedChar(‘P’, board, i, j) >= 1 or “at most one O” means countConnectedChar(‘O’, board, i, j) <= 1, or “connected to 4 other DC “ means countConnectedDC(board, i, j) == 4. As you see it is very easy. When board is finalized, nested for loops change remained W’s with L’s. End of the main method, explored board is printed as “EXPLORED“ with my method.

Implementation:

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

public class MEC2013400105 {
    /* This method is our main method, so it says to compiler what it needs execute. */
    public static void main(String[] args) throws FileNotFoundException {
        char[][] board = new char[18][18]; // board is variable that keeps board configuration.
        // Although our board is 16x16, our array is 18x18. Because it is contains 2 rows, 2 columns as outside.
        char[][] oldBoard = new char[18][18]; // oldBoard is variable that keeps previous board configuration.
        readFile(board, “input”); // Read the file
        print(board, “INITIAL: “); // Print the initial board and initial counts
        // Check to conditions until last two boards are equals
        while(!isEqualArray(board, oldBoard)){
            oldBoard = duplicate(board); // Clone board to oldBoard, and start to change current board.
            // This 2 nested for loops comes over every character on our board configuration.
            // i and j starts with 1 and ends with length-1 because edges of our board array filled with O.
            // We don’t need to check and change them
            for(int i = 1; i < board.length - 1; i++){
                for(int j = 1; j < board[i].length - 1; j++){
                    // This if-else if tree checks conditions one by one
                    if(isDC(board[i][j])){ // If a DC
                        if(countConnectedDC(board, i, j) == 4) //connected to 4 other DC
                            board[i][j] = ‘B’;
                        else if(countConnectedDC(board, i, j) == 1 //connected to 3 LC ||
(countConnectedDC(board, i, j) == 2 && countConnectedChar(‘P’, board, i, j) >= 1 ) // connected to 2 LC
and at least 1 P || (countConnectedDC(board, i, j) == 3 && countConnectedChar(‘P’, board, i, j) >= 2 )) //
connected to 1 LC and at least 2 P
                            board[i][j] = ‘P’;
                        else if(board[i][j] != ‘B’ && board[i][j] != ‘P’) // If a DC is not B and not P
                            board[i][j] = ‘C’;
                    }
                    else if(board[i][j] == ‘W’ && countConnectedChar(‘O’, board, i, j) >= 1 ) // If a W is
connected to at least one O
                        board[i][j] = ‘O’;
                    else if(board[i][j] == ‘O’ && ( //If an O
                                (countConnectedChar(‘Y’, board, i, j) >= 2 && countConnectedChar(‘O’,
board, i, j) <= 1 ) // connected to at least 2 Y and at most one O || (countConnectedChar(‘Y’, board, i, j) == 1
&& countConnectedDC(board, i, j) >= 2 ) // connected to 1 Y and at least 2 DC || (countConnectedChar(‘O’,
board, i, j) >= 1 && countConnectedDC(board, i, j) >= 2 ))) // connected to at least one O and at least 2 DC
                        board[i][j] = ‘Y’;
                }
            }
        }
        // When board is finalized, this nested for loops changed remained W’s with L’s
        for(int i = 1; i < board.length-  1; i++){
```

```
            for(int j = 1; j < board[i].length - 1; j++){
                if(board[i][j] == 'W')
                    board[i][j] = 'L';
            }
        }
        print(board, "EXPLORED: "); // Print explored board
    }
    /* This method read the text file that with given name, and put characters to board array.
    It takes 2 parameters.
        board: Our board configuration array
        fileName: Name of the file that we want read
    It returns nothing.
     */
    public static void readFile(char[][] board, String fileName) throws FileNotFoundException{
        File file = new File(fileName + ".txt"); // import file whose name is given
        Scanner scanner = new Scanner(file); // scanner is used for taking input from text file.
        // This 2 nested for loops comes over every character on our board, and fill it with O's.
        for(int i = 0; i < board.length; i++){
            for(int j = 0; j < board[i].length; j++){
                board[i][j] = 'O';
            }
        }
        // This 2 nested for loops comes over every character on our board, and filled it with input character
        // i starts with 1 and ends with length - 1 because top/bottom edges of our board array filled with O. It
represents outside.
        // We set to our array with board[i][j + 1], not with board[i][j]; because we want to keep right/left edges
as O. It represents outside.
        for(int i = 1 ; i < board.length - 1; i++){
            String line = scanner.nextLine(); // It keeps current line of input text
            for(int j = 0; j < line.length(); j++ )
                board[i][j + 1] = line.charAt(j);
        }
    }
    /* This method print current board configuration; count and print number of each character
    It takes 2 parameters.
        board: Our current board configuration
        label: The text that written at the beginning of the numbers line
    It returns nothing.
     */
    public static void print(char[][] board, String label) {
        int[] counter = new int[8]; // It keeps number of each character, It's length is 8 because we have 8
characters
        char[] chars = {'P', 'C', 'B', 'G', 'O', 'Y', 'L', 'W'}; // It keeps all character
        // This 2 nested for loops comes over every character on our board configuration.
        // i and j starts with 1 and ends with length-1 because edges of our board array filled with O. It
represents outside.
        // We don't need to print and count them
        for(int i = 1; i < board.length - 1; i++){
            for(int j = 1; j < board[i].length - 1; j++){
                System.out.print(board[i][j]); // Print the character character at this position
                // This for loop goes through every index on counter and chars array
                for(int h = 0; h < chars.length; h++){
                    if(board[i][j] == chars[h]) // If chars are matched increase corresponding index on counter
array by one
                        counter[h]++;
                }
            }
            System.out.println(); // Go to the next line at the end of the each row
```

```
        }
        System.out.print(label); // Print desired text to the beginning of the counts line.
        // This for loop goes through every index on counter and chars array
        for(int i = 0; i < counter.length; i++){
            if(counter[i] != 0) // If number of character is not equal to zero, print it
                System.out.print(chars[i] + "=" + counter[i] + " ");
        }
        System.out.print("ALL=256\n\n"); // Print the total number
    }
    /* This method checks given character is DC or not.
    It takes a character as parameter.
    It returns situation of character as an boolean.
     */
    public static boolean isDC(char ch) {
        return ( ch == 'G' || ch == 'B' || ch == 'P' || ch == 'C');
    }
    /* This method calculate number of DC connected to given position and return it.
    It takes 3 parameters.
        board: Our current board configuration
        i: row number
        j: column number
    It returns number of connected DC as an integer.
     */
    public static int countConnectedDC(char[][] board, int i, int j) {
        int count = 0; // It keeps number of connected DC
        // Check around given position. If one of them is DC increase count by one.
        if( isDC( board[i - 1][j] )) count++;
        if( isDC( board[i + 1][j] )) count++;
        if( isDC( board[i][j - 1] )) count++;
        if( isDC( board[i][j + 1] )) count++;
        return count; // Return number of connected DC
    }
    /* This method calculate number of the given character connected to given position and return it.
    It takes 4 parameters.
        ch: Character that we want to count
        board: Our current board configuration
        i: row number
        j: column number
    It returns number of given character as an integer.
     */
    public static int countConnectedChar(char ch, char[][] board, int i, int j) {
        int count = 0; // It keeps number of given character around given position
        if( board[i - 1][j] == ch ) count++;
        if( board[i + 1][j] == ch ) count++;
        if( board[i][j - 1] == ch ) count++;
        if( board[i][j + 1] == ch ) count++;
        return count; // Return number of given character
    }
    /* This method duplicates our board to new variable and return new variable
    It takes 1 parameters.
        oldBoard: Our current board configuration
    It returns board as a new 2d char array variable.
     */
    public static char[][] duplicate(char[][] oldBoard) {
        char[][] newBoard = new char[18][18]; // It keeps board in new variables
        //This nested for loops  comes over every character on our board configuration and copy them to new
board variable.
        for(int i = 0; i < oldBoard.length; i++){
```

```
            for(int j = 0; j < oldBoard[i].length; j++)
                newBoard[i][j] = oldBoard[i][j];
        }
        return newBoard; // return new board variable
    }
    /* This method checks 2 board are equal or not, and return result
    It takes 2 boards as parameter, and check their equality
    It returns result of the equality as boolean .
     */
    public static boolean isEqualArray(char[][] board1, char[][] board2) {
        //This nested for loops  comes over every character on our board configuration
        for(int i = 0; i < 18; i++){
            for(int j = 0; j < 18; j++)
                if(board1[i][j] != board2[i][j])
                    return false; //If characters on that position aren't equal, return false
        }
        return true; // If for loops didn't return false, boards are equal and return true
    }
}
```

Output of The Program:

```
WWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWW
GGGGGGWWWWWWWWWW
GGWWGGWWWWWWWWWW
GGGWGGWWWWWWWWWW
GGWWGGWWWGGGWGWW
WGGGGGGGGGGGGGWW
WWWWWWWGGGWWWGGW
WWWWWWWGGGWWWGGW
WWWWWWWGGGGWWGGW
WWWWWWWWGGWWWGGW
WWWWWWWWWGWWWWWW
WWWWWWWWWGWWWWWW
WWWWWWWWGGGWWWWW
WWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWW
INITIAL: G=61 W=195 ALL=256

OOOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOO
CCCCCCOOOOOOOOOO
CCLLCCOOOOOOOOOO
CBPLCCOOOOOOOOOO
CCLLCCYYYCCCYPOO
YCCCCCCCCBCCCCYO
OOOOOOYCBCYYYCCO
OOOOOOOCBCYOOCCO
OOOOOOOCBBPOOCCO
OOOOOOOYCCYOOCCO
OOOOOOOOYPYOOOOO
OOOOOOOOYPYOOOOO
OOOOOOOOPPPOOOOO
OOOOOOOOOOOOOOOO
OOOOOOOOOOOOOOOO
EXPLORED: P=8 C=47 B=6 O=173 Y=17 L=5 ALL=256
```

```
GGGGGWWWGWWGGGGG
WGWGWGWGWWGGGGWW
GWWWWWGGGWGWGGWW
GGGWGGWWGWWWWGWG
WGWGWWWWGGWWWGWW
GWWWGWWGGGGGGGWG
WGGGGWWGWWWGWGWG
WGWWWGWWGWWGGWGW
WGWWGGWWGWGGGGGW
GGWWWWWWGWGGGGGW
WGWGWGGWWWWWWGWW
WWWWGGWGGWWGGWWG
WGWWWWWWGWGWGWGW
WGWWGGGGGWWGGGWW
WWWGWGGWWGWGWGWW
WGGGWGGGGGWWWWWW
INITIAL: G=120 W=136 ALL=256

PPPPPYYYCYYCCCPP
YPLPLCYPYYPCBCYO
PLLLLLPPPYPYCCYO
PPPLPPYYPYYYYCYC
YPLCYOOYCCYYYCYY
CLLLPOOPCCCCCCYP
YPPPPYOPYYYCLPYP
YPYYYPOYPYYCCLPY
YPOOPPOOPYCBBCCO
PPOOYYOOPYCCCBCO
YPYCYPPYYYYYYPYO
OYYYPPYPPYYPPYYC
OPYYYYYYPYCLPYCY
OPYYPCCPPYYPPPYO
OYYPYCCLLPYPYPOO
OPPPYCCPPPOOOOOO
EXPLORED: P=74 C=42 B=4 O=33 Y=86 L=17 ALL=256
```

Conclusion:

I think I solved problem with convient way for given tools. It was clean work. Dividing my program to small portion makes my work easy. With my methods, I was able to check conditions easily.