2D Array Color Plotter

Plotting Pictures in a 2D Grid

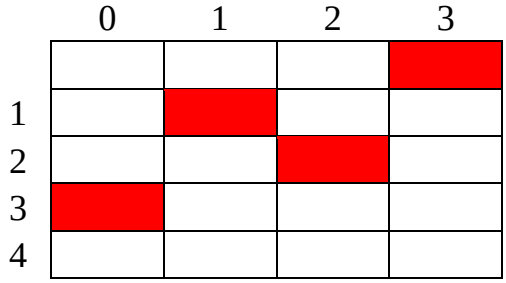
AP Computer Science A
Chapter 6 Big Java
Unit 8 Project STEM

Introduction

In this lab you will implement methods for drawing in a two-dimensional grid by changing the elements in a 2D array of `boolean` values. This will give you practice with developing algorithms for two-dimensional arrays.

You will manage a 2D array of `boolean` values named `colorArray`. This array will be used to color a two-dimensional grid of colors. If an element of `colorArray` has the value `true`, then the corresponding grid cell color will be the "drawing color". If it has the value `false`, then the corresponding grid cell color will be the "background color". See the example below in which the grid on the right is filled in with colors based on the `colorArray` array on the left. In this example, the "drawing color" is red and the "background color" is white.

	0	1	2	3
	false	false	false	**true**
1	false	**true**	false	false
2	false	false	**true**	false
3	**true**	false	false	false
4	false	false	false	false



Getting Started

1. The only file you will modify in this lab is **ArrayPlotter.java**; other files in this project help to run the GUI and menu system.

   First you will add the code in the `ArrayPlotter` class that is necessary to create an Array Plotter object and to create and display the GUI.

   a. Locate the two private instance variables in `ArrayPlotter`. **These are the only two instance variables that you will need, do not add any more.** Read the comments so that you understand their purpose.

b. Complete the constructor to initialize the instance variables as follows:

   i. Assign the reference to the `PlayGrid` object to the appropriate instance variable.
   ii. Create and assign a new 2D array of the given dimensions to the `colorArray` instance variable.

c. Complete the `clear` method. First it must assign `false` to all of the elements of `colorArray`. Then it must update the GUI. Only call `update` after the iteration is complete, don't call it inside a loop. This will result in the grid being cleared all at once instead of pausing for each cell.

   Compile and run your program. The GUI controls should now be fully operational, showing `clear` (letter 'a') as the first choice. Experiment with `clear` using various sizes, such as the following menu commands:

```
a
a 10 15
a 1 1
a 9 9
```

2. Replace the body of the `rowMajorFill` method with an appropriate implementation. This method uses a nested loop to traverse `colorArray` in *row-major* (top-down, left-to-right) order. **Use descriptive names for your loop control variables.** I suggest naming them `r` and `c`, or `row` and `col`. The body of the inner loop needs to set the appropriate element of `colorArray` to `true` and then call the `update` method of the GUI with the statement `gui.update(colorArray);`

   Compile and run your program. You should see the option to run `rowMajorFill`; you should see colors filling the locations of the grid in row-major (top-down, left-to-right) order. Note that the drawing pauses between coloring each cell. Experiment with the **speed** options (type digits 1-9) while your program is filling the grid.

Adding Additional Drawing Methods

Now it's time to write the remaining drawing methods on your own. Pay attention to the following important requirements as you write and test each of these methods.

- **Each method must fill in the grid in the order specified.**
- The `PlayGrid update` method must be called once for each `colorArray` element value change.
- Except for the `xFill` method, a method may not set a `colorArray` element to `true` and update the GUI more than once. This will cause a pause in the filling of the grid. Do not update the GUI for elements that you decide should become `false`.
- Compile and test each method as you go along.

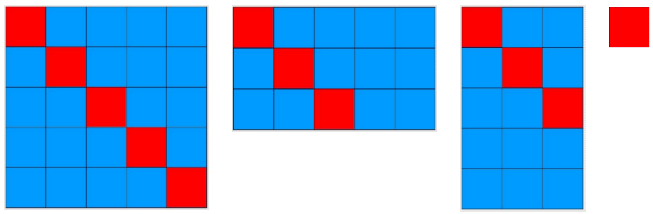
- Options for new correctly written drawing methods will magically appear in the GUI. The client code accomplishes this by using Java Reflection. Google "Java reflection" if you want to know more.
- The methods must work properly for all different grid dimensions. You should test each method on grids of the following sizes: 5 x 5, 4 x 8, 8 x 4, 1 x 1, 1 x 10, and 10 x 1.

3. Write the `colMajorFill` method, using the `rowMajorFill` method as a guide. The cells must be traversed in column-major order. A column-major order is column-by-column from left-to-right, going down each column. It first visits all the locations in column 0 top-to-bottom, then all the locations in column 1, and so on.

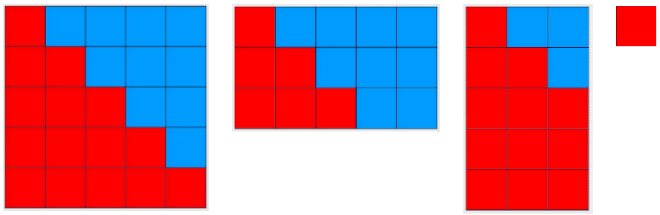
4. Write an `reverseRowMajorFill` method, using the `rowMajorFill` method as a guide. This algorithm must fill in cells bottom-up, going left-to-right across each row. In other words, the row order is reversed, but the column order is not.

5. Write an `reverseColMajorFill` method, using the `colMajorFill` method as a guide. This algorithm must fill in cells right-to-left, going up each column from the bottom. In other words, both the row and column orders must be the reverse of `colMajorFill`.

6. Write an `mainDiagonalFill` method. This algorithm must fill in cells along the diagonal from the upper-left corner towards the lower-right corner. It will end up in the lower-right corner only if the grid is square. If it is not square, the algorithm steps down and to the right until it comes to the last column or the last row, depending on whether the grid is taller than it is wide or wider than it is tall. The diagrams below show the result for a 5 x 5 grid, a 3 x 5 grid, a 5 x 3 grid, and a 1 x 1 grid.



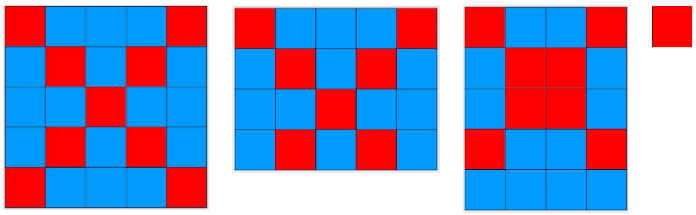
7. Write an `mainTriangleFill` method. This algorithm must fill in all the cells on and below the main diagonal. **It must fill in these cells in row-major order.** The diagrams below show the behavior for 5 x 5, 3 x 5, 5 x 3 grid, and 1 x 1 grids.



8. Write an `otherDiagonalFill` method. This algorithm must fill in cells along the other diagonal from the upper-right corner towards the lower-left corner. It will end up in the lower-left corner only if the grid is square. If it is not square, the algorithm steps down and to the left until it comes to the last column or the last row, depending on whether the grid is taller than it is wide or wider than it is tall.

9. Write an `otherTriangleFill` method similar to the first triangle fill method, but going from the upper-right corner towards the lower-left corner. (Again, whether it actually reaches the lower-left corner depends on whether or not the grid is square.) It must fill all the cells on and below the diagonal in row-major order.

10. Write an `xFill` method that draws an 'X' in the grid. **Reuse existing methods as appropriate.** Your implementation must consist of exactly two statements, with no explicit loops. The main diagonal must be drawn first. If the two diagonals share a common cell, then the corresponding `colorArray` element should be set to `true` and the GUI should be updated twice. This will cause a warning message when it draws the overlap. The diagrams below show the results for 5 x 5, 4 x 5, 5 x 4, and 1 x 1 grids.



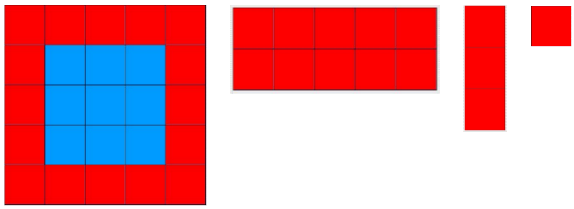
11. Your next drawing method will utilize two private helper methods. Proceed as follows:

    a. Write a `private void fillRowLeftToRight(boolean[] row)` method that traverses the `row` array from left to right. For each element in `row`, it must set that element to `true` and call the GUI update method with the `colorArray` parameter.

    b. Write a similar `fillRowRightToLeft` method that traverses the `row` from right to left.

    c. Write an `serpentineFill` method that traverses each row from top to bottom. Even indexed rows (0, 2, …) are traversed from left to right, and odd indexed rows (1, 3, …) are traversed from right to left. This method must utilize your two helper methods and must only have one loop.

12. Write an `borderFill` method that draws a border around the grid's perimeter. Your drawing must start in the upper-left corner and proceed in a clockwise fashion. You code must not reimplement any code that is already available in the methods you have written. Your method will consist of two private helper method calls and two loops. Make sure that the GUI doesn't pause unnecessarily as it fills the grid. The diagrams below show the results for 5 x 5, 2 x 5, 3 x 1, and 1 x 1 grids.



Extra Credit Options

1) Create a method `checkerboardFill` which fills the array in a checkerboard pattern in row-major order, such that on even-indexed rows (rows with even index numbers) the even-indexed columns only get filled, and on odd-indexed rows the odd-indexed columns only get filled.

2) Create a method `cornerFill` which fills the topmost row from right to left, then the unfilled cells in the leftmost column from top to bottom, then go one column to the right and fill the unfilled cells from bottom to top, then fill the unfilled cells in the second row from left to right, then go to the third row (index 2) and fill the unfilled cells from right to left which begins the sequence again until all cells are filled. Your code must fill all cells with no duplication (since that causes the double-entry error dialog), and cannot check the current value of any cell in the array before filling it.

3) Create a method `spiralFill`. Spiral IN from the outside to the middle. Start at the top left cell, then go right, then down, then left, then up. Do not fill any cells twice. Stop when you can no longer fill any cells. RESTRICTION: Don't use any if statements to check the existing state of cells: your algorithm must understand where it needs to fill without having to test.