# **Array Plotter**

# **Plotting Pictures in a 2D Grid**

**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 |

|  | 0 | 1 | 2 | 3 |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| 1 |  |  |  |  |
| 2 |  |  |  |  |
| 3 |  |  |  |  |
| 4 |  |  |  |  |

**Getting Started**

1. Copy the **ArrayPlotter Student** folder to your network drive. It contains a BlueJ project which includes **ArrayPlotter.java**, the only file you will be changing in this lab. The project also contains a **class** file (**ArrayPlotterGUI.class**) and a **jar** file (**gridgridworld.jar**). These are used to create and display the *Graphical User Interface* (GUI) that you will use to see the grid of colors. The GUI also has buttons which are used to call the methods that manipulate the grid.  
     
   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.  
   1. 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.
   2. Complete the constructor to initialize the instance variables as follows:  
      1. Construct a new ArrayPlotterGUI object. Its constructor takes one parameter, the current ArrayPlotter object. Assign the reference to that ArrayPlotterGUI object to the appropriate instance variable.
      2. Assign null to the 2D array instance variable. You’ll create or re-create a 2D array object in response to GUI button clicks.
      3. Compile.
   3. Complete the initializeColorArray method. This method must assign a 2D array of the given dimensions to the colorArray instance variable. Compile.
   4. Complete the main method. All that’s required is to create a new ArrayPlotter object, no additional code is required in main. This is an event driven program. You execute methods in ArrayPlotter by clicking buttons in the Graphical User Interface (GUI).  
        
      Compile and run your project. You should see the GUI, experiment with it. When you click the **New Grid** button, your initializeColorArray method is called. Clicking this button also makes the **ClearGrid** and **RowMajorFill** buttons available for use. These two buttons execute your onClearGridButtonClick and onRowMajorFillButtonClick methods respectively. Currently, clicking either of these buttons causes a pop-up message indicating that you have not yet implemented the corresponding method.
2. Replace the body of the onRowMajorFillButtonClick 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 all elements of colorArray to true and then call the update method of the GUI with the statement gui.update(colorArray);  
     
   Compile and run your program. Create a new grid and click on the **rowMajorFill** drawing button; 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 **Adjust Speed** slider while your program is filling the grid.
3. Replace the body of the onClearGridButtonClick 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.

**Adding Additional Drawing Methods**

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 ArrayPlotterGUI update method must be called once for each colorArray element value change.
* Do not update the GUI for false elements.
* Compile and test each method as you go along.
* Buttons for new correctly written drawing methods will magically appear in the GUI. ArrayPlotterGUI 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.

1. Write the onColMajorFillButtonClickmethod, using the onRowMajorFillButtonClick 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.
2. Write an onReverseRowMajorFillButtonClick method, using the onRowMajorFillButtonClick methodas 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.
3. Write an onReverseColMajorFillButtonClick method, using the onColMajorFillButtonClick 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 onColMajorFillButtonClick.
4. Write an onMainDiagonalFillButtonClick 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.

|  |  |  |  |
| --- | --- | --- | --- |

1. Write an onMainTriangleFillButtonClick 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.

|  |  |  |  |
| --- | --- | --- | --- |

1. Write an onOtherDiagonalFillButtonClick 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.
2. Write an onOtherTriangleFillButtonClick 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 from right to left.
3. Write an onXFillButtonClick 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.

|  |  |  |  |
| --- | --- | --- | --- |

1. Your next drawing method will utilize two private helper methods. Proceed as follows:  
   1. 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. This is the first private helper method.
   2. Write a similar private void fillRowRightToLeft method that traverses the row from right to left. This is the second private helper method.
   3. Write an onSerpentineFillButtonClick 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.
2. Write an onBorderFillButtonClick 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.

|  |  |  |  |
| --- | --- | --- | --- |

*This lab was inspired by the Dots program by Richard Rasala of Northeastern University.   
   
Copyright Alyce Brady, 2003.*

*Modified and published by Robert Glen Martin with Alyce Brady’s permission.*