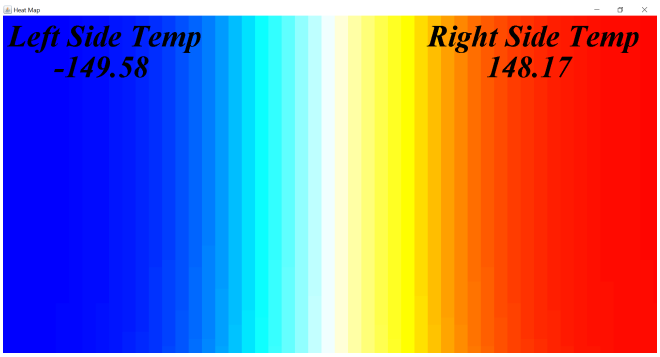
Heat Map

2D Arrays and Graphics

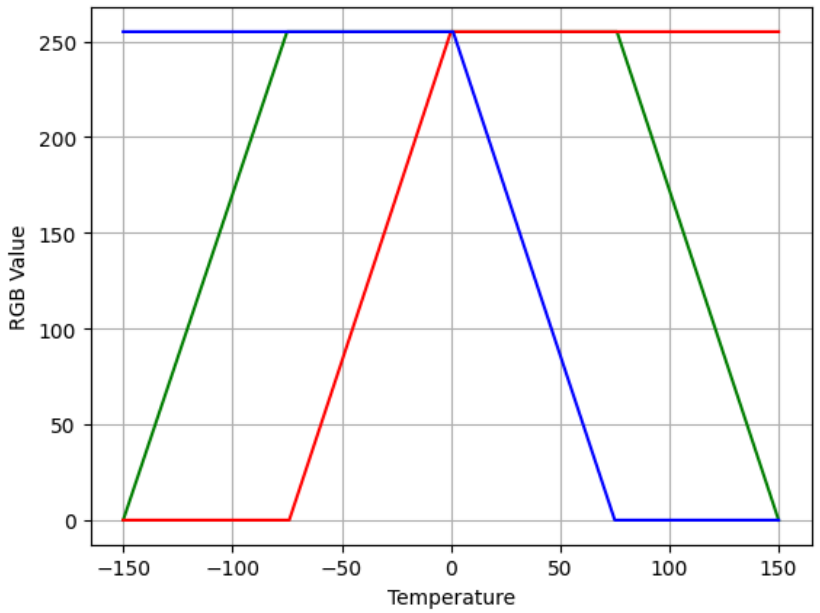
This lab models how heat is transferred within a space. If an insulated room is very cold on one side and very hot on the other side, eventually the room temperature will average out. The temperature transfer is modeled at each specific point by taking the average of all its surrounding points over a specific time period. In this lab, we'll model the heat transfer on a 2-dimensional plane/grid.



Temperature to Color mapping

To have a visual model, we'll have the max temperature represented by the color RED and the minimum temperature by the color BLUE. All temperatures within a specified range will be mapped proportionally with (RED, GREEN, BLUE) values as shown below:

In the Java Color class, each color has a red, green, and blue component with a range of 0 to 255. The following table shows how each RGB component varies within each temperature range. The max/min temperature range is 150 to -150.

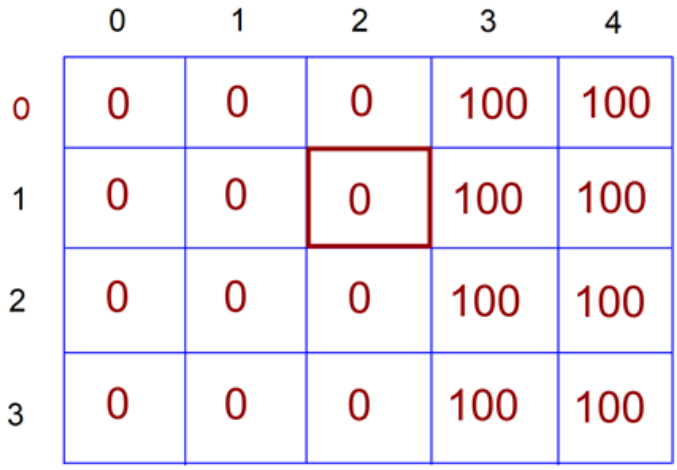


Temperature Range	Red	Green	Blue
[-150,-75]	[0,0]	[0,255]	[255,255]
[-75,0]	[0,255]	[255,255]	[255,255]
[0,75]	[255,255]	[255,255]	[255,0]
[75,150]	[255,255]	[255,0]	[0,0]

Temperature Averaging

Each “tick” of the simulation all temperature values in tempGrid[][] need to be averaged. To do so, take the average values of the neighbors to the west, north, east, south, and the cell being updated.

For example, if the temperature at index (1,2) was to be updated, the neighbor’s values at (1,1), (0,2), (1,3) and (2,2) will be averaged along with the current value at (1, 2). Then (1,2) will be updated with the value of 20.



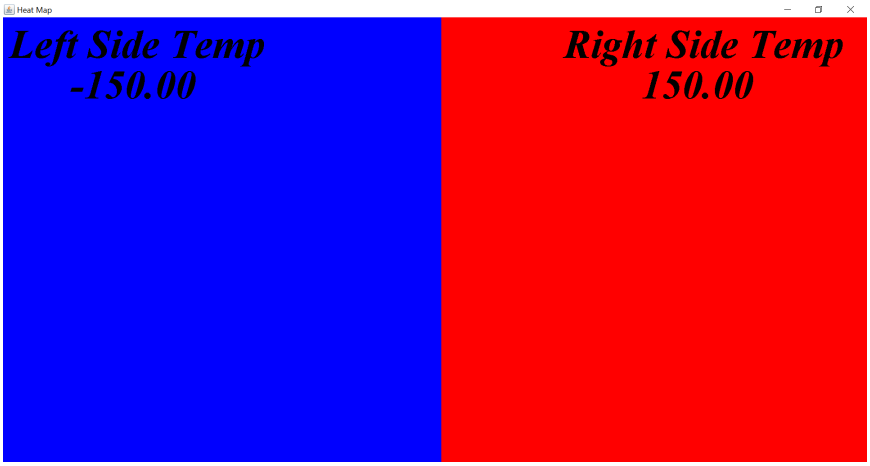
`newTemp = (0 + 0 + 100 + 0 + 0) / 5 = 20`

If a neighbor is out of bounds, that corresponding value is not used in the average, so you'll need to maintain a count. For example, if (0,0) is being updated, the surrounding neighbors are (1,0) and (0,1). You would then divide their sum by only 3.

Todo Instructions:

Part 1

1. Download the `Heatmap.java` shell code file. You will only see a black screen when you run the code until you complete all steps in Part 1.
2. Complete the `getRed, getGreen, and getBlue` methods. See the "Temperature to Color Mapping" section above. Run the program and note the outputs in the console to verify that your methods are working correctly. The `paintComponent` method (completed for you) uses these methods when drawing the `tempGrid` array onto the graphics window. *see example output on last page
3. Add code to the top of the constructor (under the `TODO` comment) that initializes the `tempGrid` array to half cold half hot (-150 on the left and 150 on the right).
4. Uncomment `t.start()` in the constructor and you should see the following:



Part 2

1. Now you need to add code that allows the heat to disperse throughout the room. This will go inside of the `actionPerformed` method so that the calculations are repeated constantly by the `timer`.
2. The new temperature in each cell (new value of each element in the 2D array) will be equal to the average of that cell and the cells above, below, left, and right of it. **Beware of edge (literally) cases!**
3. Use a standard nested for loop to traverse through the 2D array. You can start at (0, 0) and work left to right. For each element in the array calculate the new temperature and update the element.
4. See the "Temperature Averaging" section above for more detail.

Part 3

Now we will use the mouse as a hot and cold source. If you press the mouse's left button, it will set the temperature at that location to 10 times the maximum temperature. If you press the right button, it will set it to 10 times the minimum temperature. The mouse will stay that temperature until you release the button.

Note: the methods that get the Color values for red, green, and blue should be capped at the range of 0 to 255, so if the temperature exceeds the maximum or minimum, overall color should be capped at red or blue.

To support this feature, you will need to modify `mousePressed` and `mouseReleased` methods. `mousePressed` is called when you press the button down on the mouse and

`mouseReleased` is called when you release the button. Note that `mousePressed` will not be continuously called if you hold the button down.

Both of these methods are called with a reference to a `MouseEvent`, which has the following methods of interest:

- `getX()` – returns the x coordinate of where the button was pressed
- `getY()` – returns the y coordinate of where the button was pressed
- `getButton()` – returns which button was pressed. BUTTON1 is the left button and BUTTON3 is the right.
    - ex. `event.getButton() == MouseEvent.BUTTON1`

1. When the mouse is pressed you will need to do a few things:
    a. Convert the (x, y) coordinates of the click to corresponding row and column values*. Remember that the upper left hand corner of the Java graphics window (0, 0). Set the global variables `clickedRow` and `clickedCol` accordingly.
    b. Check which button was pressed and set the global variable `clickedTemp` accordingly.
2. When the mouse is released, reset clickedRow and clickedCol to -1.
3. In **`actionPerformed`** add an if statement. Check the value of `clickedRow` (or `clickedCol`) to see if the mouse is currently being pressed. If it is set the grid temperature at `[clickedRow][clickedCol]` to `clickTemp`. This will ensure that the cell is repeatedly set to the high (or low) temperature until the mouse is released.

*There are several ways to convert (x, y) coordinates to row and column values. Take a look at the code provided in `paintComponent`. If you can understand what it is doing you can do something similar in your `mousePressed` method.

Extensions:

1. This is not an accurate approximation because we are updating the matrix AS we calculate the values. We should really be updating a NEW matrix based on the OLD values. Make 1 new 2D array to hold these values then copy it over to the heatmap matrix when all are updated.
2. We're still not perfect because we're basing the average on a "cross" around a pixel. (If you click hot then cold in the same position you'll see a checkerboard appear because of this). A "square" would be better. Use a weighted average where corner points count 1 time and "cross" points count 2 times in the average.

*Sample Color Values Output:

This is an example of values returned by your RGB methods. Yours should be similar but need not be exact.

```
 -150:    0,    0,  255
 -140:    0,   34,  255
 -130:    0,   68,  255
 -120:    0,  102,  255
 -110:    0,  136,  255
 -100:    0,  170,  255
  -90:    0,  204,  255
  -80:    0,  238,  255
  -70:   17,  255,  255
  -60:   51,  255,  255
  -50:   85,  255,  255
  -40:  119,  255,  255
  -30:  153,  255,  255
  -20:  187,  255,  255
  -10:  221,  255,  255
    0:  255,  255,  255
   10:  255,  255,  217
   20:  255,  255,  183
   30:  255,  255,  149
   40:  255,  255,  115
   50:  255,  255,   81
   60:  255,  255,   47
   70:  255,  255,   13
   80:  255,  241,    0
   90:  255,  207,    0
  100:  255,  173,    0
  110:  255,  139,    0
  120:  255,  105,    0
  130:  255,   71,    0
  140:  255,   37,    0
```