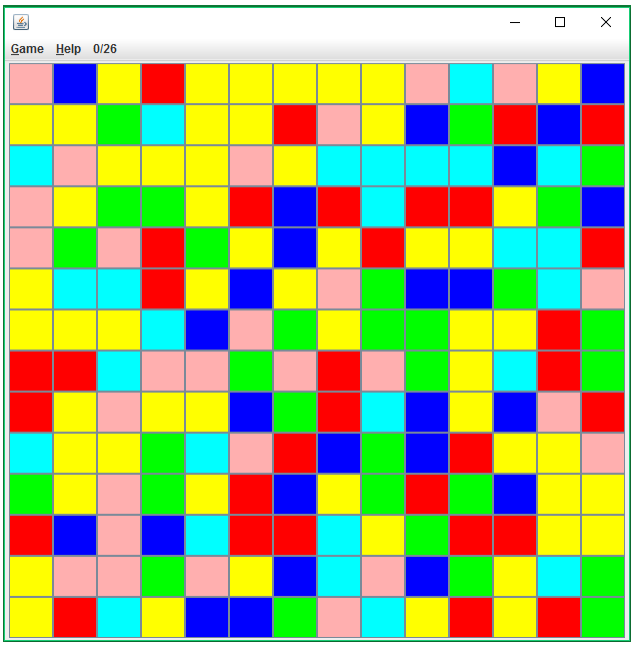
C343 Project - Flood-It!

1 The Game

The goal of the player is to *flood* the entire board with a single color within a given number of moves, e.g. for a 14x14 board, the player wins the game if they flood the board within 26 moves. The player can choose a color for the flooded region by clicking a tile which displays the color.



2 Your Task

We have written most of the game but we need your help in finishing it. We would like you to write the `flood` function in the `FloodFunction` class. The `flood` function takes two parameters:

- `color_of_tiles` - a `Map` that maps from the tile coordinates(of the `Coord` class) to their colors(of the `Integer` class). The `Coord` class contains two member values `x` and `y`, with 0 and 0 representing the upper left corner. The x coordinates increase as you go to the right and the y coordinates increase as you go down.

- `color` - an `Integer` of the color index, ranging from 0 to 5. For more details about the intger-color mappings, check the `Constants` class.

You are to store the flooded tiles in the member value `flooded_list` of the `FloodFunction` class, which by default is a LinkedList. The `flooded_list` intially contains the tile at the upper left corner.

We say that a tile is *adjacent* to another tile if it is directly above, below, left, or right, that is, sharing a side with the other one. The `FloodFunction` class contains some helpful

functions : the functions named `up`, `down`, `left`, and `right` compute the coordinates of the adjacent tiles; the function `in_bound` tells you whether a coordinate is on the board.

An *X-colored region* is a set of tiles defined as follows:

- A tile of color X is an X-colored region.
- If tile T is color X and adjacent to a tile in an X-colored region R, then $T \cup R$ is an X-colored region.

Given a `flooded_list` whose tiles are of color X, the flood function should add every X-colored region to the `flooded_list`, provided the region contains a tile that is adjacent to a tile in the `flooded_list`.

3 Analysis

After implementing and debugging your `flood` function, run `FlootIt.Driver test` in batch mode, which produces a graph of the execution time (y-axis) versus the size of the board (x-axis). Look at the graph. What function (roughly) fits that graph? (Hint: possibilities to think about are $f(n) = n, f(n) = n^2, f(n) = n \lg n$.)

4 Logistics

Create a new directory in your github at IU C343 repository named `floodit`. Download the `flood.zip` file from the course web page, put it's contents into your `floodit` directory, and add all of these files to your github repository.

Place the answer to the analysis question and a one-paragraph description of your flood implementation in the `README.md` file within `floodit`.

5 Measuring the Execution Time of Flood

In this part of the project you are to measure the execution time of your `flood` function on boards of varying sizes to see whether your algorithm scales up nicely (linearly) or whether the execution time grows at a faster rate. The `Driver` class includes support for running the game in a non-interactive "batch" mode that measures the execution time. To measure the execution time of your `flood` function, in the `bin` directory, run `Driver` with the argument `test`, for example:

```
java -classpath . Driver test
```

This will create a png file named "result.png".

Once you've created the graph, compare it to your prediction. Does the graph look like what you expected? What is the rate of growth of the execution time? Include the graph that you've created in what you turn in (upload it to github). What parts of your algorithm do you think are contributing the most to the execution time?

Try to improve the scalability of your flood function. Write new flood functions flood`x`, and graph the execution time by running `Driver` with the argument `testn`, where `x` stands for your x-th flood function and `n` stands for the total number of your flood functions. For Example, if your have `flood`, `flood1`, and `flood2`, run the following:

```
java -classpath . Driver test 3
```

Feel free to change everything in the `FloodFunction` class, but make sure that the new flood functions take the same types of inputs as the default one. By default, the efficiency test runs up to size 50; each size repeats 5 times to reduce the effect of noise. Turning down the corresponding parameters in `Constants` class may save you time when debugging.

6 Turn-in

Your `floodit` directory should include all of the files from the zip file and at least two versions of your `flood` functions. Your `floodit` directory should also contain the generated graph, `"result.png"`. Finally, you should answer all of the questions in this document in the README.md for `floodit`.