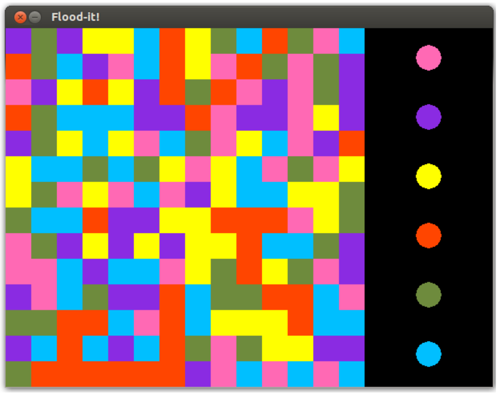
C343 Project 2 - Flood-It!

Due 11:59pm, September 12, 2014

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 25 moves. The player can choose a new color for the flooded region by clicking one of the circles to the right of the board or by pressing the following keys:



- a: pink
- s: violet
- d: yellow
- z: red
- x: olive
- c: blue

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. The `flood` function takes two arguments:

- `color_of_tile` - a Python dictionary that maps a tile coordinate to its current color. A coordinate is represented as a pair (a 2-tuple) containing the x and y values, with (0,0) representing the upper left corner. The x coordinates increase as you go to the right and the y coordinates increase as you go down. The coordinates are given in terms of pixels and each tile is 32×32 pixels.

- `flooded_list` - a Python list (an array) of coordinates for the flooded area. These tiles will always have the same color.
- `screen_size` - a pair of integers that specify the horizontal and vertical size of the screen in pixels.

We say that a tile is *adjacent* to another tile if it is directly above, below, left, or right, that is, if the two tiles share a side. You will find some helpful functions in a file named `utilities.py`: the functions named `up`, `down`, `left`, and `right` compute the coordinates of the adjacent tiles. There is also a function named `in_bounds` in `utilities.py` that 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 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 `floodit` in batch mode, which produces a graph of the execution time (y-axis) versus the size of the board (x-axis, number of tiles). 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 project directory in your `C343Fall2014` github at IU repository named `project2`. Download the `project2.zip` file from the course web page and put it's contents into your `project2` directory. The `flood` function is in the file named `flood.py`. (Not to be confused with `floodit.py`, the main program.)

You need to download and install pygame version 1.9, which you can obtain at `www.pygame.org`. Note that there are different download files of pygame for different versions of Python. They do not have Python 3 versions for many operating systems (such as Mac), so everyone needs to use the version of pygame for Python 2.7 (look for `py2.7` in the name of the pygame download file). If Python 2.7 is not already installed on your computer, you will need to install that as well.

If you have multiple version of python installed on your computer, pygame will install itself into one of them but not the others. You may have to try running `flootit.py` with each of your versions of python to find out which one has pygame.

Place the answer to the analysis question and a one-paragraph description of your `flood` implementation in the `README.md` file within `project2`.

5 Measuring the Execution Time of Flood

In this part of the project you will 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 `floodit.py` program includes support for running the game in non-interactive "batch" modes that either test your `flood` function for correctness or that time the execution time.

You're now ready to measure the execution time of your `flood` function. Run `floodit.py` with the command-line argument `time`:

```
python floodit.py time
```

This will create a file named `times.csv`. Each rows lists the number of tiles in the board and the execution time for `flood`. You can import this data into a spreadsheet such as Microsoft Excel or Google Docs and then graph it, or can you use the `plotter.py` script to create a graph. To use `plotter.py` you'll need to install `matplotlib`. For Mac users, there's a good set of instructions for this at:

`http://fonnesbeck.github.io/ScipySuperpack/`

Once you've created the graph, compare it to your prediction. Does the graph look like 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. After changing the code for `flood`, time it and graph it again as above. Describe what changes you made inside `flood` and describe the rate of growth in the new graph. Did you improve the growth rate? Include the new graph in what you turn in.

6 Turn-in

Your `project2` directory should include both versions of your `flood` function: put them in new files `flood1.py` and `flood2.py`. Your `project2` directory should also contain two graphs: the before and after execution times for flooding. Finally, you should answer all of the questions in this document in the README.md for `project2`.