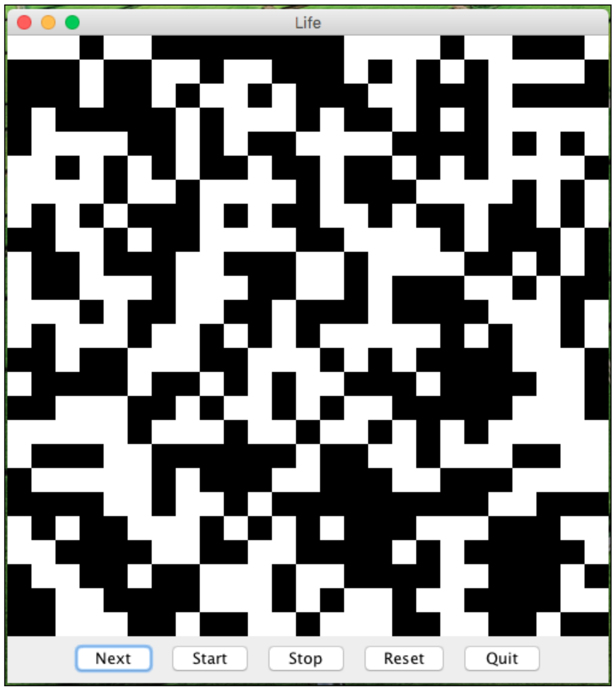
CS161: Introduction to Computer Science
Homework Assignment 9
due 4/26 by 11:59pm

Conway's Game of Life

In this assignment, you'll be implementing *The Game of Life* – the cellular automata game formulated by mathematician John Conway in the 1970s.

The game is represented using a two-dimensional array of cells. Each cell can either be dead or alive. In the picture below, the black squares are the living cells. Depending upon the state of its neighbors, a cell may either die or come to life at each generation.



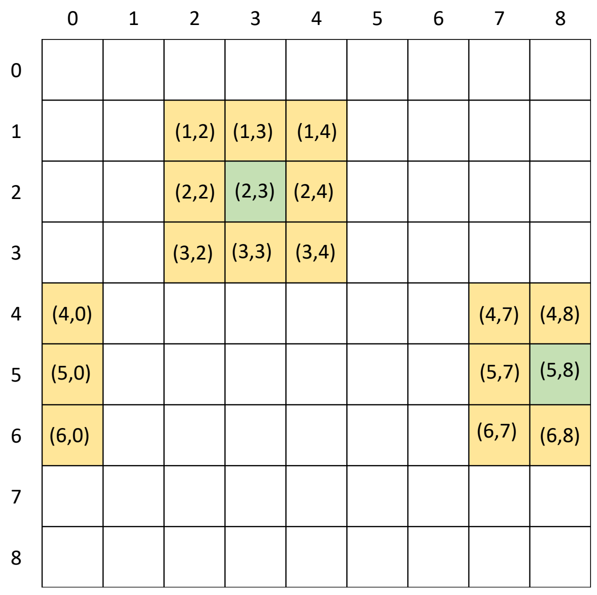
Rules of the Game

The Game of Life begins by randomly initializing each cell on the board to be either dead or alive. The game is then run for a number of iterations (also known as "generations"). For each iteration, the following rules are used to determine whether a cell remains alive, dies off, or comes back to life:

- A dead cell with exactly three living neighbors becomes a live cell
- A dead cell with less than or more than three living neighbors stays dead
- A live cell with 0 or 1 living neighbors dies (due to loneliness)
- A live cell with 2 or 3 living neighbors remains alive
- A live cell with 4 or more living neighbors dies (due to overpopulation)

Counting A Cell's Neighbors

A cell's neighbors are the cells that are horizontally, vertically, or diagonally adjacent. In other words, a cell's neighbors are the 8 other cells that it touches at edges or corners.



We consider the board to be wrapped around so that the cells at the edges still have 8 neighbors. That is, the left edge is wrapped around to touch the right edge (think about rolling a sheet of paper) and the top is wrapped around to touch the bottom.

For example, in the picture above the neighbors of the cell at `(2,3)` are `(1,2)`, `(1,3)`, `(1,4)`, `(2,2)`, `(2,4)`, `(3,2)`, `(3,3)`, `(3,4)`.

Contrast that with the neighbors of the cell at $(5, 8)$ which are `(4,7)`, `(4,8)`, `(4,0)`, `(5,7)`, `(5,0)`, `(6,7)`, `(6,8)`, `(6,0)`.

The `Life` Class

The starter code contains 3 classes:

- The `Life` class maintains the actual game board and implements the rules of the game
- The `LifePanel` class creates the panel (i.e. the window you see when you run the program)
- The `LifeFrame` class adds the interactive functionality – i.e. the buttons and responding when you push the buttons

Your job is to implement the methods in the `Life` class. Please do not change the public methods in the `Life` class because the `LifePanel` class relies upon these methods. You may, however, add as many private methods as you would like.

To run the program, right click on `LifeFrame` and select `new LifeFrame()`. This will bring up a window that shows the game board and has 5 buttons. The "Next" button will perform 1 iteration (i.e. it computes the next generation). The "Start" and "Stop" buttons will start and stop an animation showing successive generations. The “Reset” button will reset the board to a new random configuration. And the “Quit” button exits the program.

Here are some helpful hints for this assignment:

- The board should be a two-dimensional array of booleans where `true` means the cell is alive and `false` means the cell is dead.
- When you’re debugging, it will be easier to work with a smaller board. To change the number of rows and columns in the board you can change the `BOARD_WIDTH` (i.e. columns) and `BOARD_HEIGHT` (i.e. rows) variables in the `LifePanel` class.
- Don’t overwrite the cells in the board while you’re determining which cells should die or come to life in the next generation. You need to be able to reference the old board to determine the new board and, if you’re simultaneously overwriting the old board, wacky things happen.
- Although it’s random, if your board always stops changing after only 4-5 generations, something is probably wrong. My solution consistently either never converges to a steady state or takes over 20 generations to converge.

Style Guide

Before you submit your assignment, go through the checklist below and make sure your code conforms to the style guide.

Checklist

☐ All unused variables are deleted
☐ All instance variables are used in more than one method (if not, make them local)
☐ All instance variables are declared private
☐ All instance variables are initialized in the constructor
☐ Javadoc comment for all classes
☐ All methods have Javadoc comments (except for the `main` method)
☐ Good use of private methods
☐ Proper capitalization of variables (final and non-final variables), methods, and classes
☐ All numbers have been replaced with constants (i.e. a final variable)
☐ Use white space to separate different sections of your code
☐ Code is correctly indented to improve readability

See the “Style Guide” (under “Resources” on the course website) for more detailed information.

Submitting your homework assignment

You should submit your `hw9` folder with your all of your code via Moodle.