Project 2

Added by Christopher Hoder, last edited by Christopher Hoder on Sep 25, 2011

Project 02 writeup:

The goal of this project was to design and implement Conway's Game of Life. This text based simulation of the game involved creating a grid where each character in the grid represented a cell that was either dead or alive. A dead cell was the ' ' (space) character while an alive cell was the '0' character. The rules of the game are that from one iteration to the next an alive cell remains alive if and only if it has either 2 or 3 alive neighbors. In all other cases an alive cell dies. A dead cell will come alive if and only if it has exactly 3 live neighbors. In all other cases the cell will remain dead.

My solution utilized 4 classes: Cell.java, Landscape.java, LifeSimulation.java and ReadFile.java. The cell.java stores the data for any given cell including its position (rows,cols) and whether it is alive or not. The tricky method in this class is the updateState method. This method will determine whether the cell will be alive or dead in the next iteration. It does this by finding its neighbors in the landscape, passed in as an input. The method then counts the number of alive neighbors and uses this to determine whether it is alive or not depending on the rules of the game.

The ReadFile class is used to read a text file. It will go to the given path, and take the given text file and save each line in the file to a string and save all of the strings as an Array of strings. This array will be return from the openFile() method. The code for this class was taken/adapted from the code on the following website: http://www.homeandlearn.co.uk/java/read_a_textfile_in_java.html

The Landscape class holds the grid of cells for the game. The landscape class has 2 constructors. In one, the number of rows and columns in the grid is input. In the other the path to a text file containing the grid to be loaded is input into the constructor. (This method to be discussed later as part of the extension). The getNeighbors method in this class, will get all the neighbor cell objects to the cell at the given row and column. The way this is done in my solution is to use 2 nested for loops. The first loop, loops through the row before the row the given cell is in to the row after. Likewise the second loop, loops through form the column before the given cell to the column after. The code then checks to see if a cell at each one of these 9 points 1st exists and 2nd is not the given cell that you are finding the neighbors of.

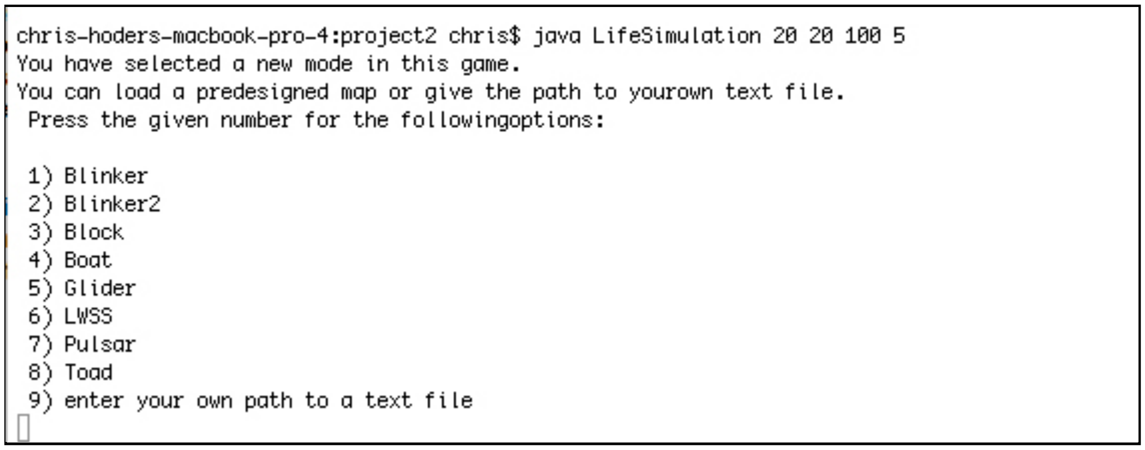
The advance method in the landscape class will create a new grid of equal size to the old grid and then loop through every cell in the grid. With each cell it will find its neighbors, and then call the updateState method on the cell to determine if it will be alive next round. The method will then create a new cell in the new grid that is either alive or dead depending on the result of the updateState method.

The LifeSimulation class will run and control the game of Life game play. The game of life can be controlled based on the command line inputs. The inputs into the command line are as follows:

```
java LifeSimulation [rows] [cols] [iterations] [density/mode]
```

All of these inputs do not need to be put into the command line inorder to run the program. If nothing is input, a grid of size 20x80 will be made with a density of 0.5 and run for 100 iterations. If something is missing or a bad value it will be set to a default value. The default values are: 20 rows, 80 columns, 100 iterations and 0.5 density. You cannot however input just the rows and iterations. The 1st input will always be interpreted as the rows, 2nd as the columns etc. If more than 4 inputs are put into the command line, only the first 3 will be used to create the grid. If a density greater than 1.0 is input, this will start up a different mode of the Game of Life.

As part of an extensions I created a new constructor for the landscape class which will allow the user to load the starting grid from a text file. The format of the text file is as follows: the first line should have rows and columns as the first 2 words. The next lines should be a picture of the grid with a ' ' (space) for a dead cell and a '0' for a live cell. When the user inputs a density greater than 1 the user will be prompted to select either from a pre-saved grid or to input the path to their own. The terminal would look as follows:

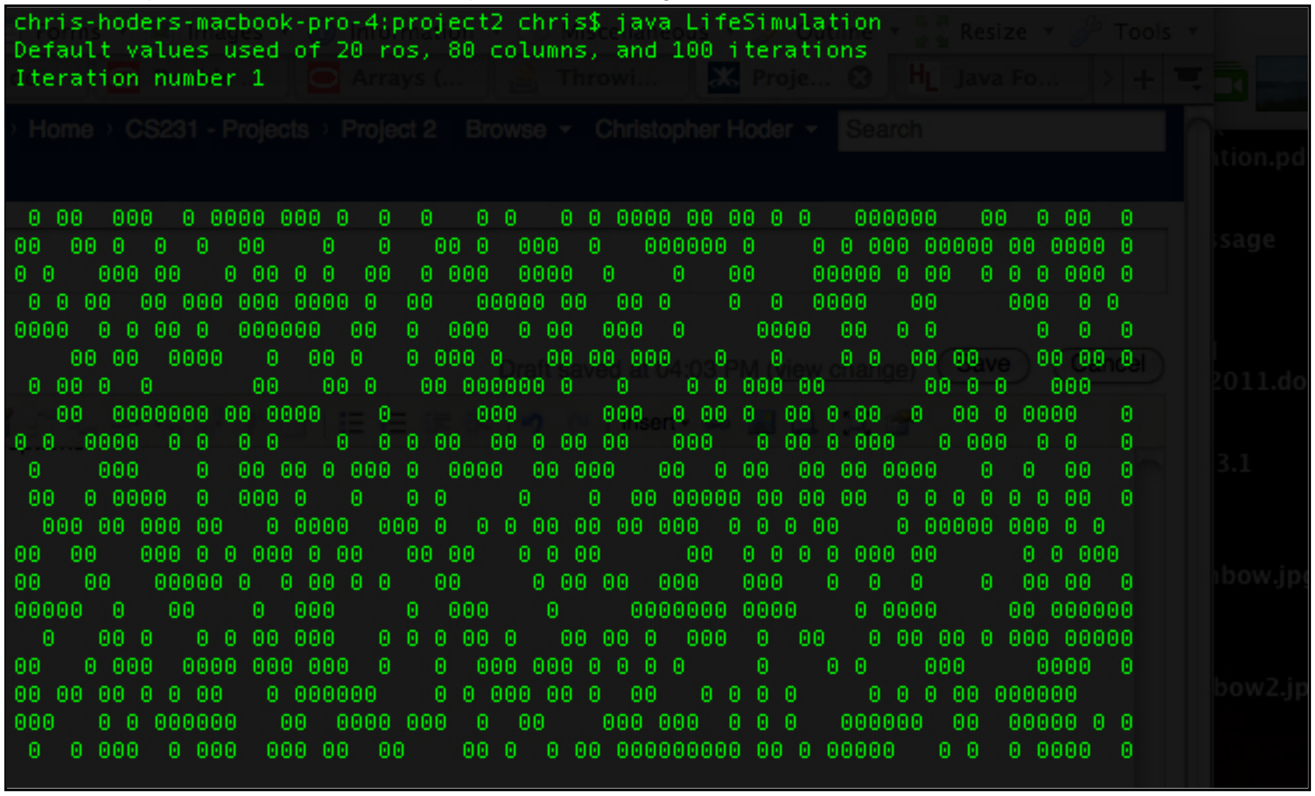


```
chris-hoders-macbook-pro-4:project2 chris$ java LifeSimulation 20 20 100 5
You have selected a new mode in this game.
You can load a predesigned map or give the path to yourown text file.
 Press the given number for the followingoptions:

 1) Blinker
 2) Blinker2
 3) Block
 4) Boat
 5) Glider
 6) LWSS
 7) Pulsar
 8) Toad
 9) enter your own path to a text file
```

Depending on the choice the given grid will be loaded and run for the number of iterations, determined by the 3 input on the command line.

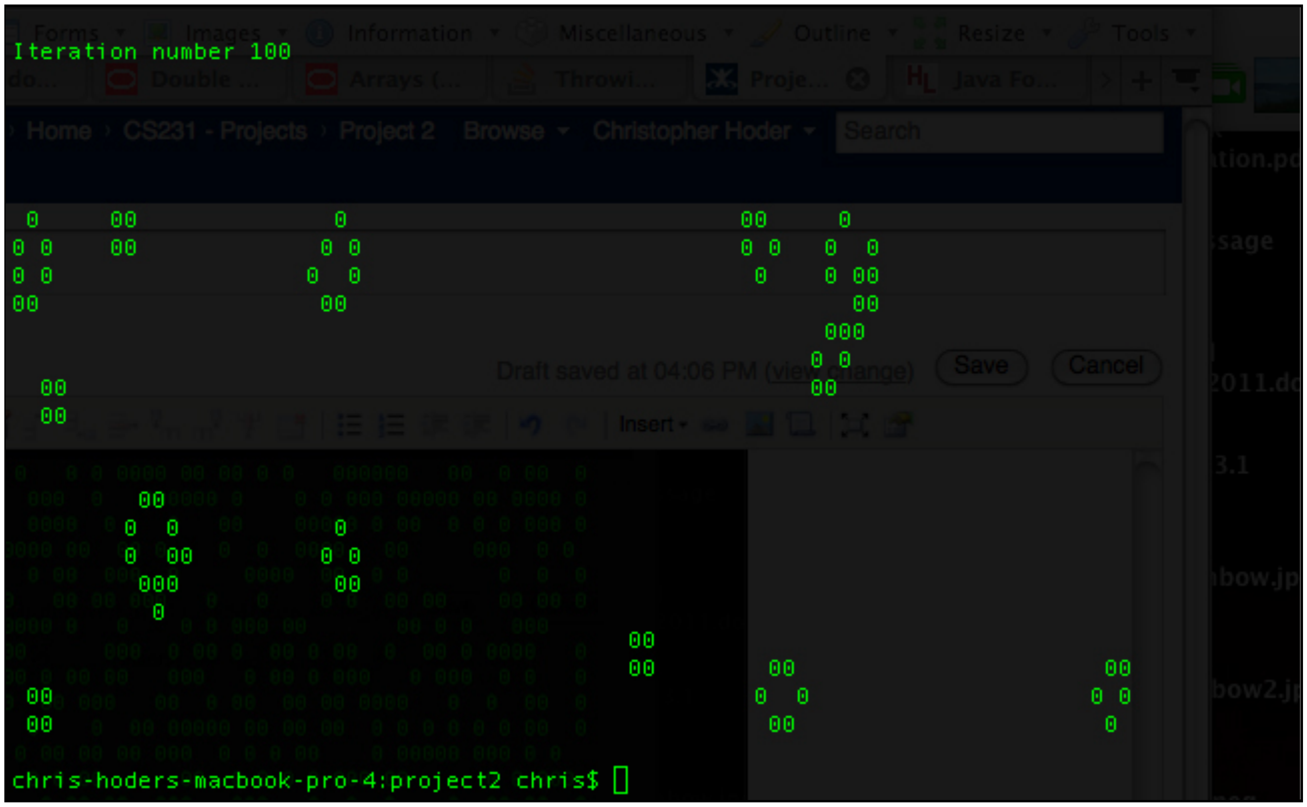
Here is a printout of when the user inputs the following into the command line



```
chris-hoders-macbook-pro-4:project2 chris$ java LifeSimulation
Default values used of 20 ros, 80 columns, and 100 iterations
Iteration number 1

 0 00  000  0 0000 000 0  0  0   0 0   0 0 0000 00 00 0 0   000000   00  0 00  0
00  00 0  0  0  00    0   0   00 0  000  0   000000 0    0 0 000 00000 00 0000 0
0 0   000 00   0 00 0 0  00  0 000  0000  0    0   00    00000 0 00  0 0 0 000 0
 0 0 00  00 000 000 0000 0  00   00000 00  00 0    0  0  0000   00     000  0 0
0000  0 0 00 0  000000  00  0  000  0 00  000  0     0000  00  0 0       0  0  0
    00 00  0000   0  00 0   0 000 0   00 00 000   0   0    0 0  00 00    00 00 0
 0 00 0  0       00   00 0   00 000000 0   0    0 0 000 00       00 0 0   000
   00  0000000 00 0000    0      000      000  0 00 0  00 0 00  0  00 0 0000   0
0 0  0000  0 0  0 0    0  0 0 00  00 0 00 00   000   0 00 0 000   0 000  0 0   0
 0    000    0  00 00 0 000 0  0000  00 000   00  0 00  00 00 0000   0  0  00  0
 00  0 0000  0  000 0   0   0 0     0    0  00 00000 00 00 00  0 0 0 0 0 0 00  0
  000 00 000 00   0 0000  000 0  0 0 00 00 00 000  0 0 0 00    0 00000 000 0 0
00  00   000 0 0 000 0 00   00 00   0 0 00      00  0 0 0 0 000 00      0 0 000
00   00   00000 0  0 00 0 0   00     0 00 00  000   000   0  0  0    0  00 00  0
00000  0   00    0  000     0  000    0     0000000 0000    0 0000     00 000000
  0   00 0   0 0 00 000   0 0 0 00 0   00 00 0  000  0  00   0 00 00 0 000 00000
00   0 000  0000 000 000  0   0  000 000 0 0 0 0     0    0 0    000     0000  0
00 00 00 0 0 00   0 000000    0 0 000 00 0  00   0 0 0 0     0 0 0 00 000000
000   0 0 000000   00  0000 000  0  00    000 000  0 0 0   000000  00  00000 0 0
 0  0 000  0 000  000 00  00    00 0  0 00 000000000 00 0 00000   0 0  0 0000  0
```

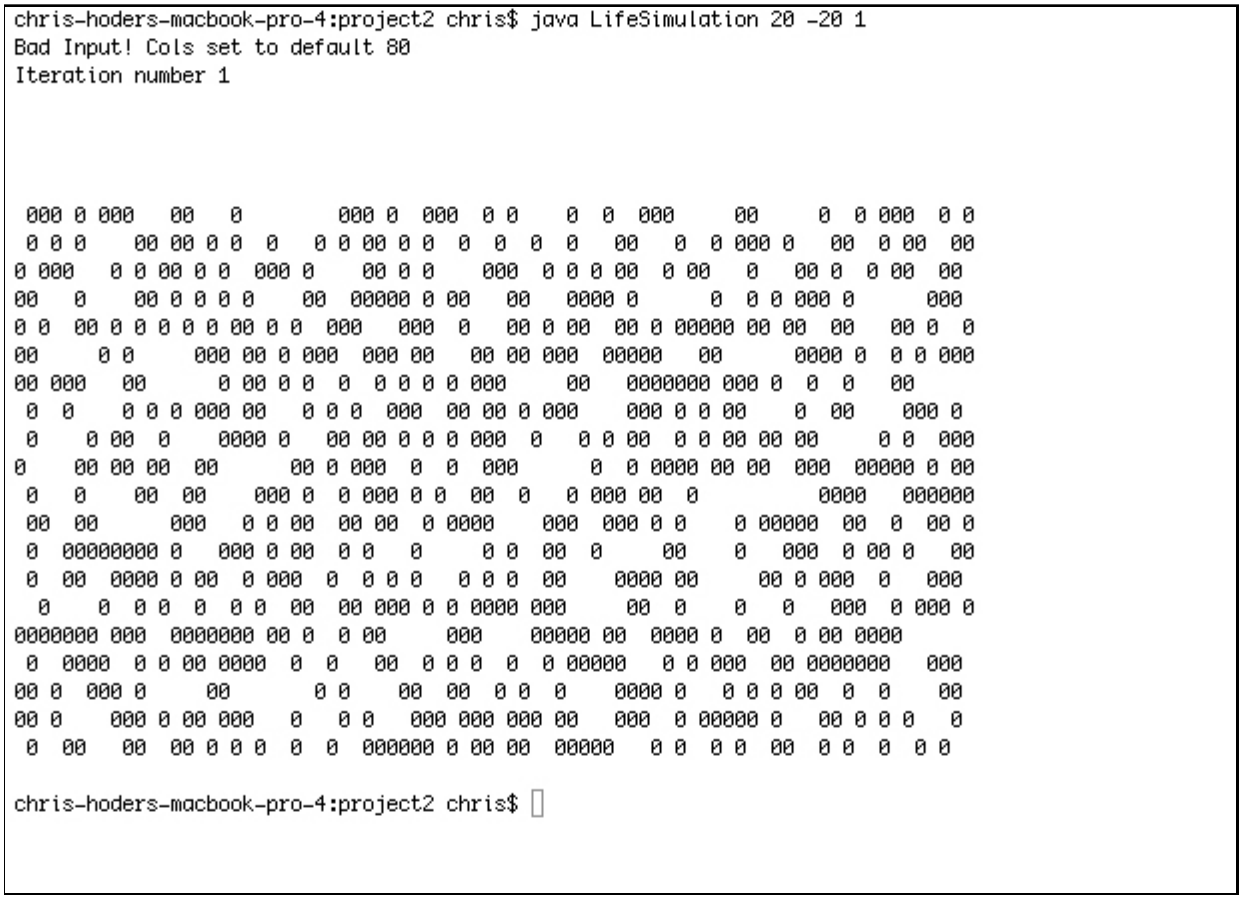
... (skip all iterations from here to 100)



```
Iteration number 100
 0     00              0                            00      0
0 0    00             0 0                         0 0    0  0
0 0                  0  0                            0    0 00
00                    00                                     00
                                                          000
                                                          0 0
  00                                                     00
  00


         00
        0  0           0
        0  00         0 0
         000           00
          0
                                            00
                                            00        00                      00
 00                                                  0  0                    0 0
 00                                                   00                       0
chris-hoders-macbook-pro-4:project2 chris$
```

Here is a screen shot showing that you can input an incorrect value and it will be set to the default:



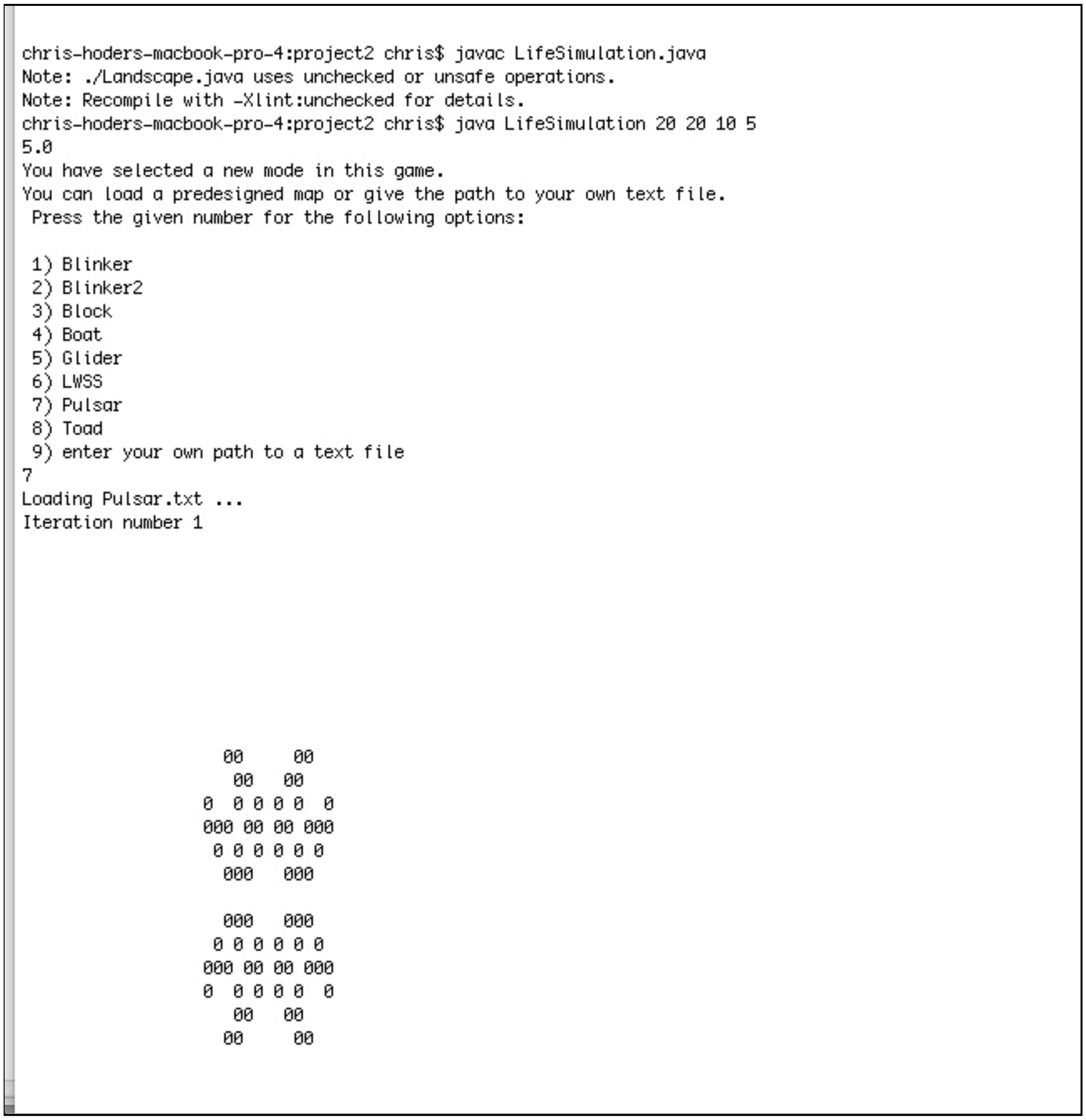
```
chris-hoders-macbook-pro-4:project2 chris$ java LifeSimulation 20 -20 1
Bad Input! Cols set to default 80
Iteration number 1




 000 0 000   00   0        000 0  000  0 0    0  0  000     00     0  0 000  0 0
 0 0 0    00 00 0 0  0   0 0 00 0 0  0  0  0  0   00   0  0 000 0   00  0 00  00
0 000   0 0 00 0 0  000 0    00 0 0    000  0 0 0 00  0 00   0   00 0  0 00  00
00   0    00 0 0 0 0    00  00000 0 00   00   0000 0      0  0 0 000 0      000
0 0  00 0 0 0 0 0 00 0 0  000   000  0   00 0 00  00 0 00000 00 00  00   00 0  0
00     0 0     000 00 0 000  000 00   00 00 000  00000   00      0000 0  0 0 000
00 000   00      0 00 0 0  0  0 0 0 0 000     00   0000000 000 0  0  0   00
 0  0    0 0 0 000 00   0 0 0  000  00 00 0 000     000 0 0 00     0  00     000 0
 0    0 00  0    0000 0   00 00 0 0 0 000  0   0 0 00  0 0 00 00 00      0 0  000
0    00 00 00  00       00 0 000  0  0  000      0  0 0000 00 00  000  00000 0 00
 0   0    00  00    000 0  0 000 0 0  00  0   0 000 00  0           0000   000000
 00  00      000   0 0 00  00 00  0 0000    000  000 0 0    0 00000  00  0  00 0
 0  00000000 0   000 0 00  0 0   0     0 0  00  0      00     0   000  0 00 0   00
 0  00  0000 0 00  0 000  0  0 0 0   0 0 0  00     0000 00     00 0 000  0    000
  0    0  0 0  0  0 0  00  00 000 0 0 0000 000     00  0     0    0   000  0 000 0
0000000 000  0000000 00 0  0 00     000    00000 00  0000 0  00  0 00 0000
 0  0000  0 0 00 0000  0  0   00  0 0 0  0  0 00000   0 0 000  00 0000000   000
00 0  000 0     00       0 0    00  00  0 0  0     0000 0   0 0 0 00  0  0    00
00 0    000 0 00 000   0    0 0   000 000 000 00   000  0 00000 0   00 0 0 0   0
 0  00   00  00 0 0 0  0  0  000000 0 00 00  00000   0 0  0 0  00  0 0  0  0 0

chris-hoders-macbook-pro-4:project2 chris$
```

This is an example of loading a pre loaded text



```
chris-hoders-macbook-pro-4:project2 chris$ javac LifeSimulation.java
Note: ./Landscape.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
chris-hoders-macbook-pro-4:project2 chris$ java LifeSimulation 20 20 10 5
5.0
You have selected a new mode in this game.
You can load a predesigned map or give the path to your own text file.
 Press the given number for the following options:

 1) Blinker
 2) Blinker2
 3) Block
 4) Boat
 5) Glider
 6) LWSS
 7) Pulsar
 8) Toad
 9) enter your own path to a text file
7
Loading Pulsar.txt ...
Iteration number 1









                    00     00
                     00   00
                  0  0 0 0 0  0
                  000 00 00 000
                   0 0 0 0 0 0
                    000   000

                    000   000
                   0 0 0 0 0 0
                  000 00 00 000
                  0  0 0 0 0  0
                     00   00
                    00     00
```

In conclusion this project taught me alot about dealing with tricky edge cases when I was coding the getNeighbors method. Additionally, as part of my extension I learned alot about loading a text file and the syntax for that which was really confusing and I ended up using some online help pages to read about which packages I needed to load inorder to this. Being able to load a given grid however made it easier to debug some parts of my code and to test that certain methods were working well because I could easily create a grid that I wanted and knew what would happen (Such as the pulsar above). I did not have time to go in and code safety checks to make sure that the user inputs an actual text file.

Labels

cs231f11project2