Maddy Placik Project2 CS 231

Overview

For this project, we were to simulate Conway's Game of Life. This game is based on mathematical functions determining a visual representation of "cells" that are either dead or alive. The state of the cell is determined based on various methods and functions, taking into account the state of the neighboring cells.

In order to achieve this goal, we used Cell, Landscape, LandscapeDisplay and LifeSimulation classes.

The Cell class was responsible for holding the state of the given cell (whether it is dead or alive). The state of the cell was contained as a boolean. To retrieve a tangible value, we created a toString method that returned 0 when the cell was dead and 1 when the cell was alive. This class also held a draw method that was used to create a visual display of dead and alive cells.

The Landscape class created a scape in which the cells would be represented. This was created as a grid, and a cell was placed in each position of the specified grid. In this class, we created a getNeighbors method which was able to return a list of the neighbors for a given cell.

The LandscapeDisplay class was written for us but contained a reference to a draw method in both the Landscape and Cell classes. It was our responsibility to create these draw methods.

In the LifeSimulation class, we simulated Conway's game of life using the previous three classes.

Cell Class

In order to represent the states of given cells numerically, we created a toString method which returned either 0 or 1 depending on the state of the given cell. Live cells were assigned a value of "1" and dead cells were assigned a value of "0".

```
public String toString(){
    if (this.state == true){
        return "1";
    }
    else{
        return "0";
    }
}
```

In order to draw given cells, we implemented a draw method. This method used a Graphics object and drew live cells as red rectangles and dead cells as white rectangles.

```
if (this.getAlive() == true){
    g.setColor( Color.red );
    //cell is in (x,y) position with size (scale x scale)
    g.fillRect( x , y , scale , scale );
}
//if the cel is dead, it will be drawn as a white square
if (this.getAlive() == false){
    g.setColor( Color.white );
    //cell is in (x,y) position with size (scale x scale)
    g.fillRect( x , y , scale , scale );
}
```

This class also included an "updateState" method which was responsible for changing the state of the specified cell based on its neighbors' states. This method was responsible for determining the "rules" of the game.

Landscape Class

This class also included a toString method, representing a grid of cells as 0's and 1's. In order to display these numbers clearly, we used a for loop, looping through the given cells and creating a new string line when appropriate.

```
//represents Lanscape object as string
public String toString(){
    String output = "";

    for ( int i = 0 ; i < this.rows ; i++){

        for ( int j = 0 ; j < this.cols ; j++){
            output = output + grid[i][j].toString() + " " ;
        }
        output = output + "\n" ;
    }

    return output;
}
```

The Landscape classifier used Cell objects named "grid" which were allocated to each node in an array of a specified number of rows and columns.

```
/*sets the number of rows and columns to the specified
values and allocates the grid of cell references
*/
//then it should allocate a cell for each location in the grid
public Landscape( int rows , int cols ){
    this.rows = rows;
    this.cols = cols;

    //creates Cell object named "grid" with specified number of rows and columns
    this.grid = new Cell[this.rows][this.cols];

    //for loop creating Landscape of cells
    for ( int i = 0 ; i < this.rows ; i++){
        for( int j = 0 ; j < this.cols ; j++){
            Cell cell = new Cell();
            grid[i][j] = cell;
        }
    }
}
```

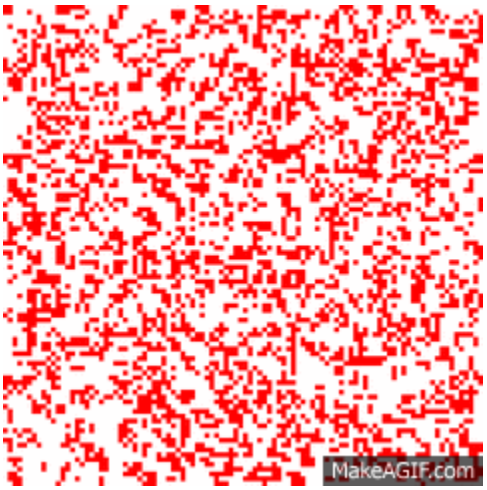
The "getNeighbors" method of this class created an ArrayList cell objects. In order to create this method successfully, we needed to take into account cells with less than 8 neighbors (cells on first and last rows/columns).

```
public ArrayList<Cell> getNeighbors( int row , int col ){
ArrayList<Cell> neighbors = new ArrayList<Cell>();

    for ( int i = row - 1 ; i <= row + 1 ; i++ ){
        for (int j = col - 1 ; j <= col + 1 ; j++){
            if ( i == row && j == col ){
                continue;
            }
            else if ( i < 0 || i >= this.rows || j < 0 || j >= this.cols){
                continue;
            }
            else {
                neighbors.add( grid[i][j] );
            }
        }
    }
    return neighbors;
}
```

LifeSimulation

Using our final LifeSimulation class, we were able to create the following series of images representing Conway's Game of Life:



Extension # 1

For my first extension, I decided to play around with the visualization of the game. In order to to this, I created a random color generator which allocated a random color to the alive cells.

```
//creates random object
Random rand = new Random();

//random primary color objects
float r = rand.nextFloat();
float gr = rand.nextFloat();
float b = rand.nextFloat();

//create random color by using random r,g,b values
Color randomColor = new Color(r,gr,b);

// if cell is alive, it will be drawn as an oval in a random color
if (this.getAlive() == true){
    g.setColor( randomColor );
    //cell is in (x,y) position with size (scale x scale)
    g.fillRect( x , y , 200 , 200 );
}
```

This modification made the LifeSimulation look like the gif below:

Extension #2

For my second extension, I decided to allow for standard input arguments for the LifeSimulation class. The first two numbers in the input represent the dimensions of the Landscape and the third number represents the number of iterations.

```
//creates standard input arguments
int lx = Integer.parseInt(args[0]);
int ly = Integer.parseInt(args[1]);
int iter = Integer.parseInt(args[2]);
```

The command line argument appears as follows to create a simulation of 3 iterations and of dimensions 300 x 300:

```
Madelines-MacBook-Air:Project2 Maddy$ java LifeSimulation 300 300 3
```

Conclusions

By completing this project, I was able to learn a lot more about Array objects and boolean values. Boolean values allow for the execution of many tasks using very little memory. They are extremely efficient ways to represent certain objects. Arrays provide for an organized method of storing and retrieving data. Our Landscape array allowed us to retrieve neighbors of a given cell in an efficient and clear manner.

To complete this project, I had help from TA's Kyle and Matt as well as from my tutor, Liwei.