**Project Plan**

**Purpose** : Our project intends to simulate Conway’s Game of Life visually on a finite, 20x20 grid . We will let the user set the initial conditions. Then, our program animate each generation following these rules for each square on the grid, with neighbors being defined as cells directly and diagonally adjacent:

1. If a the cell has less than two neighbors that are alive, it dies
2. If a cell has two or three living neighbors, it stays alive
3. If a cell has more than three living neighbors, it dies
4. If a dead cell has three neighbors, it comes back to life.

Because our grid is finite, the state of edge cells will be determined using their available neighbors.

**Examples**: (living cells will be shaded in gray, dead cells will be blue)

**General Case Examples:**  (Cells with eight neighbors (Non-edge cells))

Positions of referred cells:

0 1 2

3 4 5

6 7 8



The center cell would die because it only has 1 living neighbor, cell 2 (Rule 1)



The center cell would stay alive because it has two living neighbors, cells 1 and 2 (Rule 2)



The center cell would die because it has more than three neighbors, cells 0, 1, 2, and 3 (Rule 4)



The center cell would come to life because it has three living neighbors, cells 1, 2, and 7 (Rule 2)



The cell in the upper right corner would die because it only has 1 neighbor, cell 4.

**Special Case Examples:**



The grey cell in the corner (a corner cell) only has three neighbors, so its state will be determined by only three neighbors. Rule three would not apply to corner cells because they have less than four total neighbors, but the other rules will. In this example, the cell would die because it has no living neighbors.



The grey cell in this example is an edge cell, which only has five neighbors. Its state for the next generation would be determined by its five neighbors. In this case, it would die because it has no living neighbors. All rules governing the state of a cell would apply to the edge cells because they have a sufficient number of neighbors.

**Classes Headers, Method Headers, and Fields**

public class Cell

* private fields
  + boolean state
  + double x
  + double y
* methods
  + public boolean getState()
  + public void setState(boolean s)
  + public void drawCell(Graphics myBuffer)
  + public boolean contains(double x,double y)

public class GamePane extends JPanel

* private fields
  + private Cell[][] grid
  + private Timer t

public class GameDriver

* public static void main(String[] args)

Algorithm

* in GameDriver:
* set up JFrame container
* read location of alive cells from grid
* set content pane as game pane
* In GamePane:
* Read file
  + The file will contain 400 random 1s or 0s to create a living or dead state for each cell
* When “Start” button is pressed, start animation
* animation steps
  + store previous generation state.
    - This could possibly be implemented by creating a text file that stores the value (1 or 0) that the cell is in during the recorded generation.
    - 1 is an indication of cell death, while 0 is representative of a living cell.
  + The state of each cell must be determined using the previous generation’s state
    - In other words, newborn cells cannot affect the state of previous cells
  + for each cell in grid:
    - count total number of neighbors(number of valid indices between 0 and 5 for each row)
    - int[] neighbors = int[number of neighbors]
    - count the number of neighbors which are alive and dead( directly adjacent and diagonally adjacent cells) by adding neighbor to state
    - if number of living neighbors < 2
      * set state of cell to dead (1)
    - if 2<= number of living neighbors <=3
      * keep cell alive (0)
    - if number of living neighbors > 3
      * set state of cell to dead (1)
    - if number of living neighbors = 3 and cell is dead
      * set cell to alive position (0)
    - General Case:
      * 8 neighbors
    - Special Case:
      * Corner cells (3 neighbors), disregard rule 3
      * Edge cells (5 neighbors), no rules to disregard
      * Special cases will be evaluated using the same rules as the general case of eight neighbors unless exceptions are stated (above)

**Division of Labor:**

* Shriya
  + File read and animation panel
* Vivian
  + Animation and Cell class
* Report - both