INTRODUCTION TO COMPUTER SCIENCE II
PROJECT 1
The Game of Life

1 A game that isn't really a game

John Conway's *Game of Life* is not really a game to be played. It is a *cellular automaton*, but that doesn't tell you much, either. It is an evolving environment—a grid of *cells* that live or die, from one generation to the next, based on a simple set of rules. From these mindless, superficial rules comes some startlingly complex behavior. For an extreme example of such behavior, watch this little movie (although you should do so with the sound off, because it's goofy). We will discuss what's happening with these automata a bit in class. For now, our goal is to implement this so-called *game*.

2 Getting started

If you are doing your work on `remus/romulus`, login with Remote Desktop, open a terminal window, create a directory for your work, and change into it. Next, grab the initial source code for this assignment:

- **On `remus/romulus`:** Copy the source code with the following command...

```
$ cp ~sfkaplan/public/COSC-112/project-1/*.java .
$ cp ~sfkaplan/public/COSC-112/project-1/*.init .
```

- **On your own computer:** Download the source code by clicking here. Note that it is a *zip file*, so you will need to extract its contents.

Examine the files you just copied. There's a good bit there, and you should expect to spend significant time simply grasping the relationship between the files. Here is a description of what's there:

- `Life.java`: This simple class contains the `main()` method that gets the program started. It creates a `Game` object and then calls `play()` on that object to get the program moving. **You should not change this class.**

- `Game.java`: A `Game` is the high-level director of this cellular simulation. It reads some not-so-simple work of reading an *initial grid file* (see below) and creating the `Grid` of `Cell` objects described therein. It then is responsible for *evolving* the cells for the number of *generations* request by the user, displaying the grid at each through a `UserInterface` object.

  There are two methods in this class that you must write: the `evolve()` method; and, the `getPopulation()` method. Both are described in their comments.

- `Grid.java`: A `Grid` is a two-dimensional container of `Cell` objects. `You should not change this class.`
- `Cell.java`: A `Cell` is either *dead* or *alive*. It additionally determines, based on the cells around it—its *neighborhood*—whether it should live or die in the next generation.

  There are three methods in this class that you must write: the `countLiveNeighbors()` method; the `evolve()` method; and, the `advance()` method. All three are described in their comments.
- `UserInterface.java`: Provides the ability to print the current state of the game. **You should not change this class.**
- `Support.java`: A handy utility method or two. **You should not change this class.**
- `simple.init`: A simple *initial grid file*. It contains pairs of integers such that the first line provides the size of the grid, while all subsequent lines provide the coordinates of initially live cells. Taken together, these form starting state of the game in generation 0.
- `X-pattern.init:` Another *initial grid file*. It specifies a modestly larger grid with a more interesting pattern of initially live cells.

3 Your assignment

Write the methods needed in the `Game` and `Cell` classes. You should initially debug your code with the `simple.init` and `X-pattern.init` files, but you are encouraged to make more complex initial grid files of your own.

4 How to submit your work

Use the CS submission systems to submit your work. Specifically, you will need to submit your `Game.java` and `Cell.java` files. **Each time you submit, please submit both files together.** You may use either of the following two methods, while connected to `remus` or `romulus`, to use the submission system:

- **Web-based:** Visit the submission system web page.
- **Command-line based:** Use the `cssubmit` command at your shell prompt, like so:
  `$ cssubmit Game.java Cell.java`

This assignment is due on Sunday, Feb-26, 11:59 pm.