Project 1: Conway's Game of Life

In this project you'll be given more code than you normal get in labs. In addition, this code is documented, and uses a different style than what you're used to from labs. Professor Kaplan was nice enough to donate the code for this project. I recommend you start early, as there is a lot of code to read through.

This is not a team project. You must work and submit as an individual. See the class policy regarding high level collaboration amongst your fellow students.

You may have already coded up Conway's Game of Life for another course. That's fine, this time we're doing it using the object-oriented programming paradigm discussed in lecture.

Conway's Game of Life

In 1970, the Mathematician John Conway invited a "game" as a simplified simulation of cellular life. You can read more about the game here: https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life

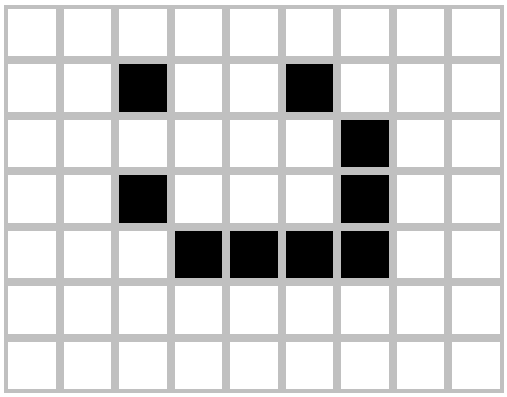
In this project, you will be filling in the final methods for an implementation of Conway's Game of Life.

Rules

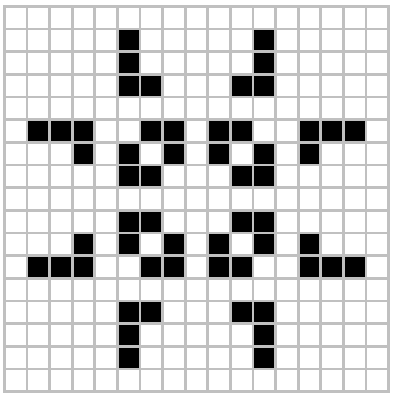
The Game of Life takes place on a grid of squares (cells). Each cell is, at any given time in one of two states: `alive` or `dead`. An initial configuration (specifying which cells are alive and which are dead) is provided. Then the grid updates itself in a series of steps (or "generations"). At each step, every cell changes its value based on the values of its neighbors (the neighbors of a cell are the 8 cells around it (above, below, left, right, and the four diagonal cells)). Cells change in each generation based on the following rules:

1. Death by Underpopulation: An `alive` cell with fewer than two living neighbors becomes `dead`.
2. Living on: An `alive` cell with two or three `alive` neighbors stays `alive`.
3. Death by Overpopulation: An `alive` cell with four or more `alive` neighbors becomes `dead`.
4. Reproduction: A dead cell with *exactly* three `alive` neighbors becomes `alive`.

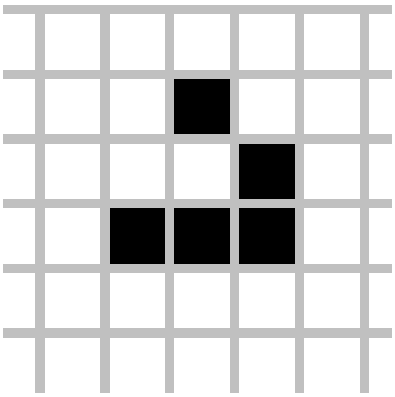
From these simple rules, very complicated behavior can arise. For example (from wikipedia):



A LightWeight Space Ship (LWSS)

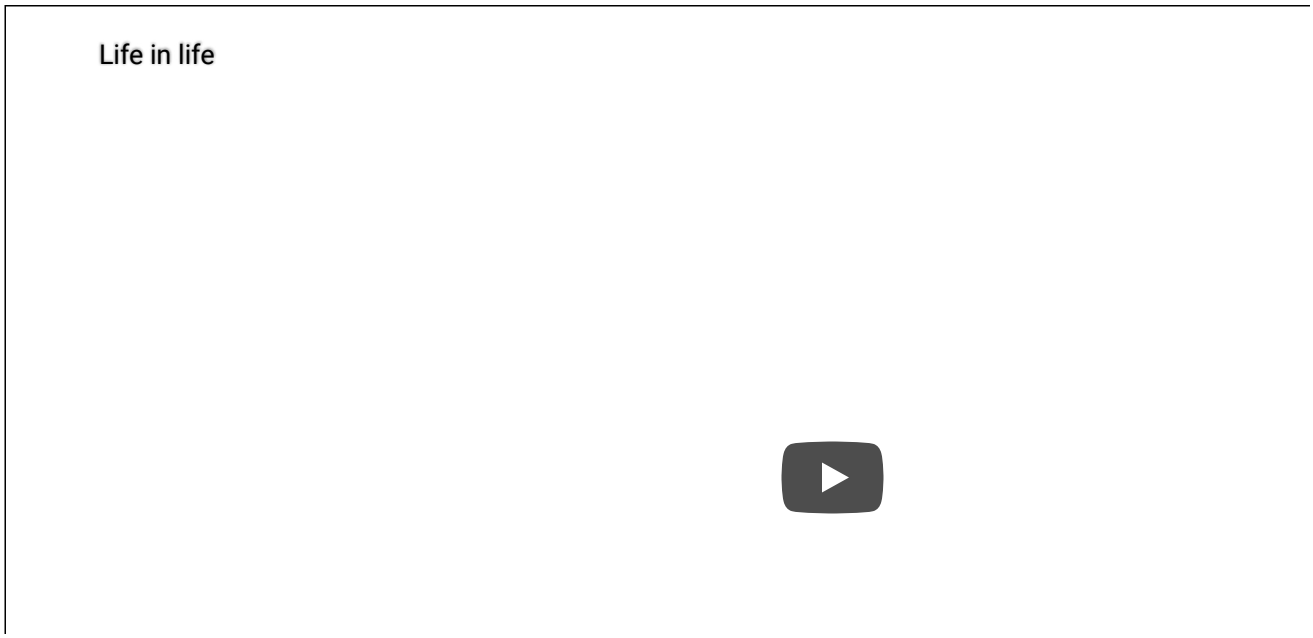


A Pulsar



A Glider

For a sense of just how complicated a behavior these rules can lead to, see:



Code:

To begin, download and unzip the following file:

project-1
Archive

Failed to load Dropbox embed preview.

https://www.dropbox.com/s/kkzn4nwcfv7deff/battlesnake.ai?dl=0

First, read through the java files to understand how the program is structured.
Then, to compile, type:

```
javac *.java
```

(The `*` is a special character (called a wildcard) which, in this context, lets us compile all the `.java` jfiles at once.)
To run the program, type:

```
java Life [some init file] [number of generations to run] [Graphic or Text]
```

For example:

```
java Life simple.init 10 Graphic
```

will run the problem using the initial configuration specified in `simple.init` for `10` generations.

Note about graphics

The `nographics` flag may be helpful in debugging.

What you need to implement

For this assignment, you will be writing 5 methods:

1. In `Game.java`: `evolve()`
2. In `Game.java`: `getPopulation()`
3. In `Cell.java`: `countLiveNeighbors()`
4. In `Cell.java`: `evolve()`
5. In `Cell.java`: `advance()`

Look at the comments within the code, as well as how each method is called, to see what each method should do. You should not change any other `.java` file.

Tips:

1. Start early. You should read through the code to understand what is going on at a high level before you dive in and start writing code. It can take time to grok what's happening, so start early.
2. `simple.init` and `X-pattern.init` are fairly simple initialization configurations. Just because your solution works on those doesn't mean it works in general. You should perform your own, additional testing by creating other initialization files. The file line of an `.init` file contains the width and height of the grid. The rest of the lines specific the locations of all `alive` cells.
3. Be methodical when debugging. Make `.init` files which are simple, and for which you know what should happen. When you make a change to your code, check that you *still* pass tests you used to pass.

To Submit Your Work

You will submit 3 files.

Submit `Game.java` and `Cell.java` on Moodle.
Remember to also fill out the survey: https://cosc112s22.page.link/Project01Survey

Finally, submit a file `my**config.ini**t` which specifies a initial configuration which leads to interesting behavior when run for 100 generations. Note: There will be some subjective grading for this, but for the most part, anything non-trivial will result in full credit, and I reserve the right to grant extra credit for particularly interesting submissions.