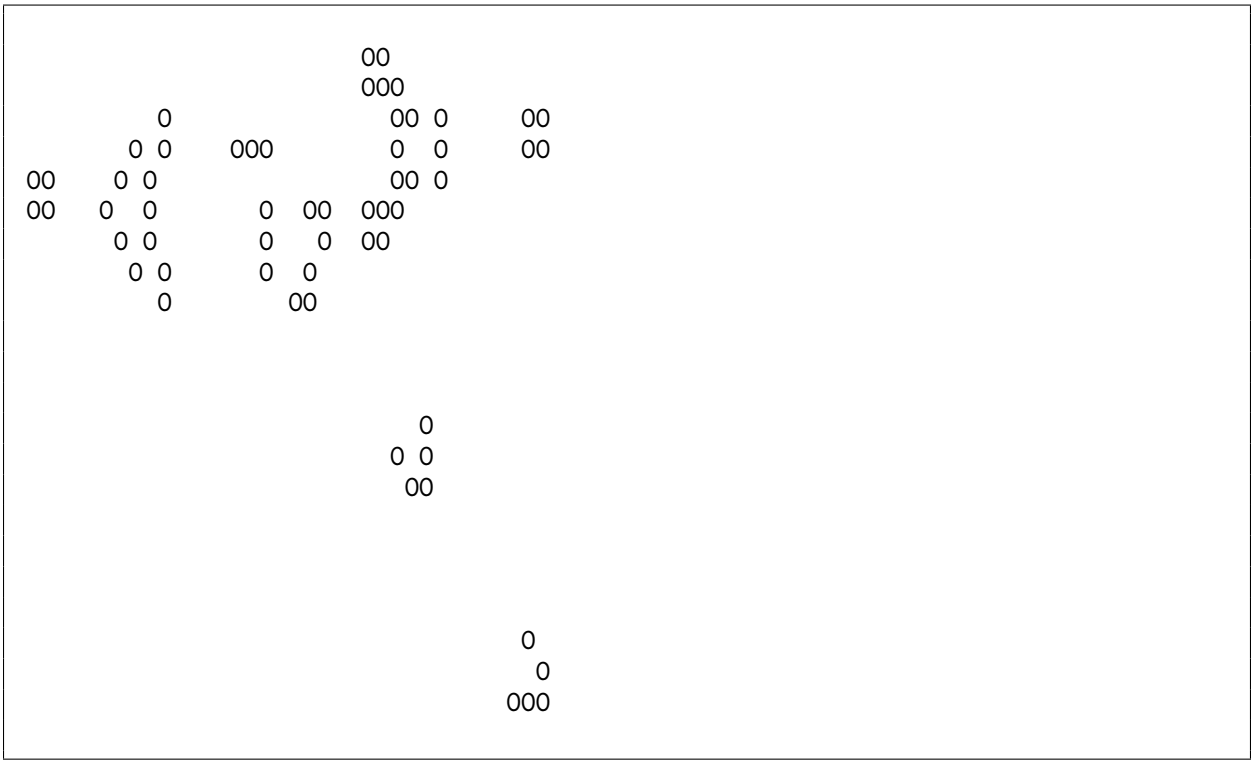
COM S 327, Spring 2017
Programming Project 0



The Game of Life

Conway's *Game of Life* is the best known of a class of mathematical games called as *cellular automata*. The mathematician John Conway invented the game in 1970. In it, the cells represent living organisms which breed and die according to specific rules. The Game of Life has been studied for decades and has had countless scholarly articles written about it (it has even had a computer implemented within it!).

Cellular automata live in arbitrary mathematical spaces. The standard rules for Life (and for most cellular automata) have it played in $\mathbb{Z}^2$ (two-dimensional, integer Euclidean space), but simple extensions of the rules can move the game into other spaces, like $\mathbb{Z}^3$. The automata interact with their space, making changes to it, and their behavior is determined by the state of the space.

Life is supposed to be played in an infinite plane, but it's also interesting on a torus, and because we have finite memory, we're going to choose the latter, which means "wrapping" coordinates ($y_{\text{min}} - 1$ becomes y_{max}, $x_{\text{max}} + 1$ becomes x_{min}, etc.) when cells reach the edges of the finite representation. If you're not familiar with the concept of a torus, imagine inscribing a Euclidean plane on a donut.

Every cell in the world is either *alive* or *dead*. Each cell has 8 *neighbors*; these are the cells adjacent to it on the plane, or, in our case, on the torus. The game is played by turns, and at each turn, for each cell in the world, one of the following rules is applied:

1. If the cell is alive and has 0 or 1 live neighbors, it dies.
2. If the cell is alive and has 2 or 3 live neighbors, it lives on.
3. If the cell is alive and has 4 live neighbors, it dies.
4. If the cell is dead and has exactly 3 live neighbors, it becomes alive.

You may refer to this Wikipedia article for more information about the Game of Life:

https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life.

Your game is to be played on an 80×24 torus (80×24 is the standard, default size of a terminal), so you'll probably want an appropriately sized 2-dimensional array to represent your world. My solution will display a space for each dead cell and 'O' for each live cell; however, you may choose to use any pair of printable ASCII characters you like. Simply printing the world after every time-step with a short pause (see below) between each frame will allow you to watch the progression like a video.

You will read a set of cell coordinates from the command line as x and y pairs. The cells at these coordinates should be initialized as *live*; all other cells are to be initialized *dead*. Your program should continue to run as long as there is at least one live cell. If it reaches a steady-state that would continue indefinitely, you can kill it by hitting control-C (you don't need to code anything for this; it's default behavior).

The following parameter set creates a *glider* and a *lightweight space ship* which travel around the torus multiple times before colliding at time 371 and achieving a steady state at time 463:

```
2 1 2 3 3 4 4 4 5 1 5 4 6 2 6 3 6 4 15 0 16 1 14 2 15 2 16 2
```

And

```
1 5 2 5 1 6 2 6 11 5 11 6 11 7 12 4 12 8 13 3 13 9 14 3 14 9 15 6 16 4 16 8 17 5 17 6 17
7 18 6 21 3 21 4 21 5 22 3 22 4 22 5 23 2 23 6 25 1 25 2 25 6 25 7 35 3 35 4 36 3 36 4
```

creates a *glider gun* which shoots itself in the figurative foot, leading to its destruction, with steady state at time 812.

Most of the code for this assignment deals with simple loops and array indexing, as well as some printing, and is very similar—or even identical—to what you'd use in Java. Those things which are different, we'll discuss explicitly in class. In particular, you'll need `atoi()` to convert the (string) command line arguments into integers, and you'll need `usleep()` to make the game run slowly enough to observe. `usleep()` sleeps for the specified number of microseconds (10^{-6} seconds); through trial and error, I've found that 12 frames per second (`usleep(83333)`) works well.

See the syllabus for information about what to turn in and submission format. In particular, you must write, use, and turn in a *Makefile*!

Extra Challenges (nothing below this line is required)

- Modify your world so that your cells live on a (simulated) infinite plane.
- Modify your world so that your cells live on a flat, non-toroidal plane. You'll need special rules to handle the edges.
- Modify your world so that your cells live on a cylinder. You'll need special rules for the unconnected edges.
- Modify your world so that edges "reflect" your live cells away.
- Modify your world—and it's rules—so that your cells live in $\mathbb{Z}^3$.
- Make the world size a command line parameter.