

Computer Science 101: Introduction to Java and Algorithms

A pineapple with a green crown of leaves and a brown, textured body is wearing a pair of bright yellow sunglasses. The sunglasses have dark lenses that reflect the surrounding environment. The pineapple is centered in the frame.

Project 1: Game Of Life

Erudition Labs

Computer Science 101: Introduction to Java and
Algorithms

May 17, 2019

Contents

1	John Conway's Game of Life	1
1.1	Cellular Automata	1
1.2	The Rules	1

1 John Conway's Game of Life

Most of this introduction comes from https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life If you would like to just read about it there. John Conway was a mathematician who was trying to create a simulation that would fit computer scientist, John von Neumann's definition of what life is. Von Neumann was trying to do this by engineering a solution by making electromagnetic components randomly float in liquids or a gas. Another mathematician by the name of Stanislaw Ulam invented "cellular automata" to simulate what Von Neumann was trying to do. Finally, John Conway created a set of complex mathematical rules that would yield a configuration that would fit Von Neumann's definition of life. That being said, Conway was more interested in trying to be able to complex and unpredictable cellular automata that could go on forever or die out depending on the initial configuration. It's an interesting way to see patterns evolve and show the emergence of organisms as well as their self-organization with Conway's deterministic rules.

1.1 Cellular Automata

Cellular automaton consists of a grid of cells that have a finite number of states and where rules are applied to all of the cells. It is quite commonly used in simulations in the sciences like physics, chemistry etc. We will be talking about how we will do this in a bit.

1.2 The Rules

Many conclusions have been drawn from Conway's experiments. His mathematical rules can be summed up in words as the following.

We assume that we have an infinite 2d, orthogonal grid.

1. Any live cell with fewer than two live neighbours dies, as if by underpopulation.
2. Any live cell with two or three live neighbours lives on to the next generation.
3. Any live cell with more than three live neighbours dies, as if by overpopulation.
4. Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

As it turns out, these rules, when applied to cellular automaton, will create complex patterns and form groups of cells (organisms) that do things like oscillating pulsation and even move.

2 Our Project

We are going to implement this simulation using most of the concepts we have learned so far like variables, arrays, loops, if-statements and methods. We are going to translate the above descriptions and rules into working code.

2.1 Modeling

2.1.1 2D infinite Grid and Cell States

Lets think about how we can do this. The rules require an infinite 2d grid. Obviously, we cannot do the infinite part since we do not have infinite memory. However, we can still operate under the assumption that the grid is infinite.

We can use a two-dimensional array to represent our infinite 2d grid. We will need it to be fixed size, so we are going to have to break the rules a bit. When a cell is about to move past the limits of our 2d array, what are we going to do? Well one option is to wrap it around to the other side. However, I get a less infinite feeling from this. Another option, and the one we will most likely do, is to just kill the cell once it surpasses the confines of our grid.

So we have a 2d array to represent our 2d grid, and each element of our 2d array will be a “cell“. In the rules, there are only two states that a cell can be in, alive or dead. There are a number of ways that we could represent those states. Perhaps we could use boolean values. If we create a 2d array of type boolean, then we could let “true“ be “alive“ and “false“ be dead. This is probably the simplest way, but it is also limiting to two states. As we go through the course, we are going to be re-factoring this code as well as adding to it. I plan on adding to the rules and the states. So to give us more options, we are going to use an integer array. This way, we can use “0“ as dead and “1“ as alive.

So again, we are going to represent our infinite 2d grid as a 2d integer array where once a cell leaves the confines of our array, we will simply assume that it will go on forever without really affecting our immediate system, so we will just kill that cell.

2.1.2 Rules

Now how can we apply the rules that govern the states of the cells. Well we could use a switch statement or if-statements to check each of the rules. We will loop over each element in our 2d array. Keep in mind that each element is a cell that is either dead or alive. We will use the current index to calculate where the neighbors of that cell is at. Then we will index into the array where the neighbor positions are to check their states. Finally, we will use their states to decide if the current cell should live or die. Once we have iterated over the entire array, that will be considered one generation.

2.1.3 Visualization

To Keep things simple, for now, we will simply be printing all of the live cells in the standard output, aka terminal. In other words, the simulation will be text based for now.