AMS 595.01 Fall 2020
MATLAB Homework: The Game of Life

Author: Oliver Yang

due either September 18, October 16, or November 13, at 11:59pm

(None of the linked content in this document needs to be read. They only serve as pointers for future exploration, if that interests you.)

A former college math professor of mine, John H. Conway, shown below, died from coronavirus a few months ago. In honor of his life, we will play with a cellular automaton he invented in 1970, called the Game of Life.

1 Background

1.1 Experimental Mathematics, Simulation, and Visualization

"Mathematics is not a deductive science – that's a cliché. When you try to prove a theorem, you don't just list the hypotheses, and then start to reason. What you do is trial and error, experimentation, guesswork. You want to find out what the facts are, and what you do is in that respect similar to what a laboratory technician does." – Paul Halmos

The purpose of applied math, and particularly computational math, is not only to discover and analyze numerical algorithms, but also to use them to do computational *science.* Since the invention of the computer, simulation has become a very powerful tool to test hypotheses and even discover surprising new mathematical and scientific phenomena, especially as computing power continues to increase. It is often said that in addition to theory and experiment, simulation has become a third pillar of science – theoretical experiment, which is needed when either adequate theory does not yet exist (e.g. for turbulence or superconductivity), experiments are not feasible to perform (e.g. in cosmology), they are too dangerous (e.g. with the climate), unethical (e.g. in biology), or too expensive (e.g. in finance).

Another way in which computers have aided scientific understanding is through graphics. Nowadays simulations and observations generate huge amounts of complex data, and effective visualization is often needed to interpret scientific results.

In this assignment, we will use both applications of computers. Since the mathematical and scientific background of the class is varied, we will do simulations and visualizations of a simple kind. But despite the simplicity of the subject matter, it is actually related to deep theoretical results in computability. The hope is that this assignment will help motivate you to learn to program.

1.2 Cellular Automata

Cellular automata were invented in the 1940's at Los Alamos National Lab by mathematician-physicist-computer scientists Stanisław Ulam and John von Neumann, shown below, while they were studying models of crystal growth, liquid motion, and self-replication.

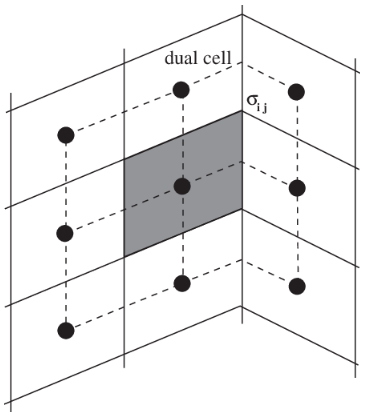
Here they are seen conversing with Feynman at LANL:

Among other applications, people have used cellular automata to generate patterns resembling those found in nature:

1.2.1 The Universe

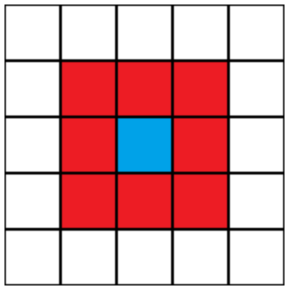
In a cellular automaton, the universe is a grid of cells, each of which assumes one of a finite number of states (e.g. white, green, cyan, magenta, blue, and purple in the grid of regular hexagonal cells on the next page).

Every grid has a cell-centered dual. Here is a rectangular example:



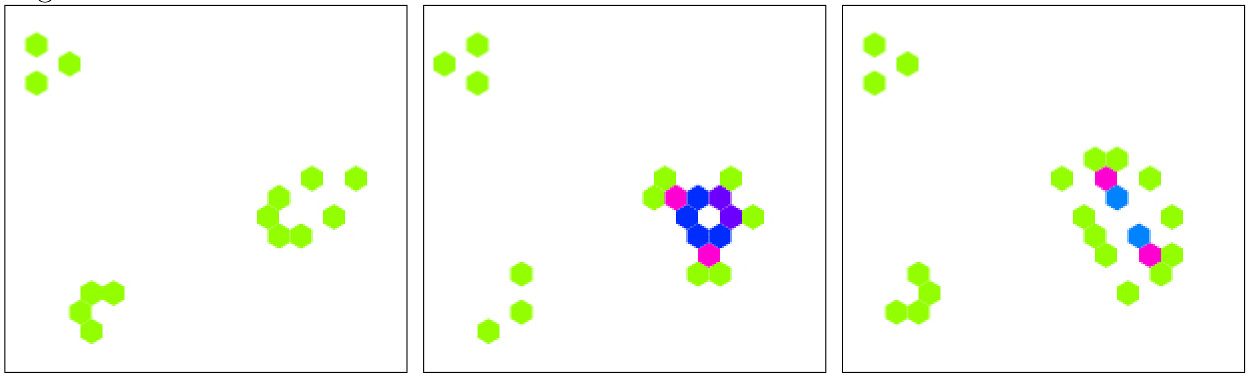
Typically, cells are considered neighbors if they share a facet (an edge in 2D, or a face in 3D), i.e. their corresponding cell-centered vertices are adjacent in the dual. So every cell in an infinite rectangular grid typically has 4 neighboring cells. This is a von Neumann neighborhood.

In this assignment, however, we will not use von Neumann neighborhoods, but will consider cells to be neighbors if they share a vertex. So the blue cell shown below has the 8 red cells as its neighbors. This is a Moore neighborhood:



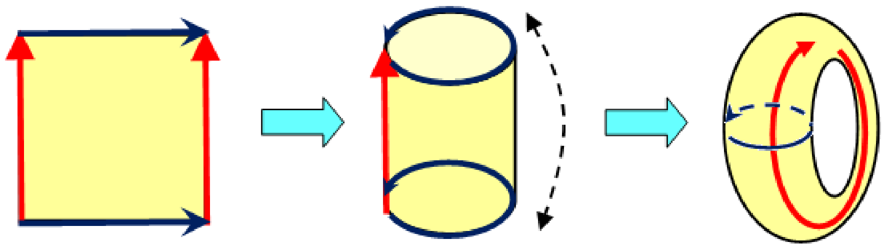
1.2.2 Evolution

The universe is seeded with an initial configuration of states in all its cells. From the current configuration, a set of rules is then used to determine a new configuration of states in all the cells. The new state of a cell depends only on its current state and the current states of its neighbors. In this way, the universe evolves though successive generations indefinitely. Here is an example of three generations:



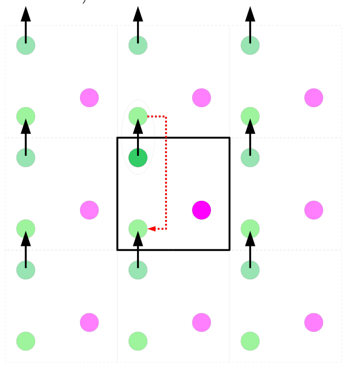
1.2.3 Periodic Boundary Conditions

At the moment, cells at the boundary of a finite rectangular universe have fewer than 8 neighbors. To make the adjacency matrix symmetric, we will identify opposite edges of the universe as the same, wrapping the square into a 2-torus, $\mathbb{T}^2 = \mathbb{R}^2/\mathbb{Z}^2$:



In topology, this operation is called cutting-and-pasting, or surgery.

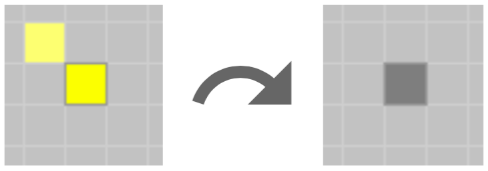
This has the effect that moving upward out of the top edge of the square means you come back in through the bottom, and vice versa. Moving out of the right edge of the square means you come back in through the left, and vice versa. In this way, cells at the boundary will have a full set of 8 neighbors as well. In differential equations, these are known as periodic boundary conditions:



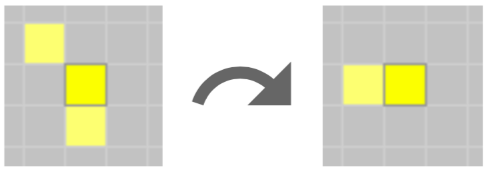
1.3 The Game of Life

The Game of Life is a 2D cellular automaton on a square grid. Cells in the grid can only assume two states: dead or alive (think, for example, of bacteria). It is traditional to color the dead cells white and the alive cells black. After much experimentation, Conway chose the following simple rules that allow small seed configurations to produce complicated behavior, but without exponential growth. These same rules apply to all cells and do not change from generation to generation:

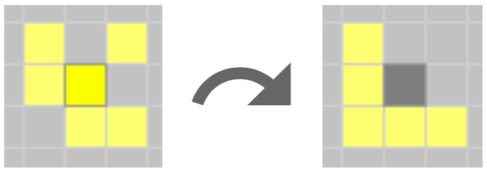
1. A cell that is alive, with exactly 0 or 1 neighbors that are also alive in the current generation, is dead in the next generation (death from underpopulation):



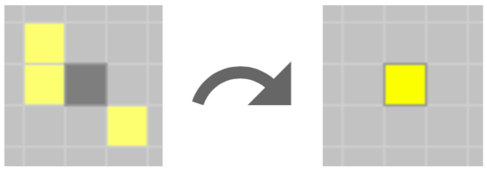
2. A cell that is alive, with exactly 2 or 3 neighbors that are also alive in the current generation, is alive in the next generation. (survival):



3. A cell that is alive, with 4 or more neighbors that are also alive in the current generation, is dead in the next generation (death from overpopulation):



4. A cell that is dead, with exactly 3 neighbors that are alive in the current generation, is alive in the next generation (birth by reproduction):



And, of course, a cell that is dead, with either fewer or more than 3 neighbors that are also alive in the current generation, is also dead in the next generation.

(You should verify for yourself that in each of the four diagrams above, each of the 8 neighboring cells, and not just the center cell itself, indeed evolves according to these rules.)

The Game of Life is implemented in MATLAB's `life` function. In each simulation, the universe is seeded with a different random configuration. You should run several simulations to observe the qualitative characteristics of how the universes evolve. There is also a web app in which you can specify your own seed configuration, along with a long list of patterns that have been discovered. It is preloaded with a seed configuration that we will use in the assignment. Note that MATLAB uses periodic boundary conditions, while the web app does not.

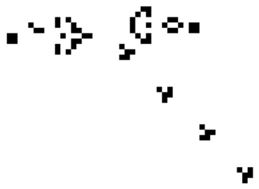
1.4 Mathematical Questions

There are many interesting mathematical questions that can be asked about the Game of Life as a dynamical system.

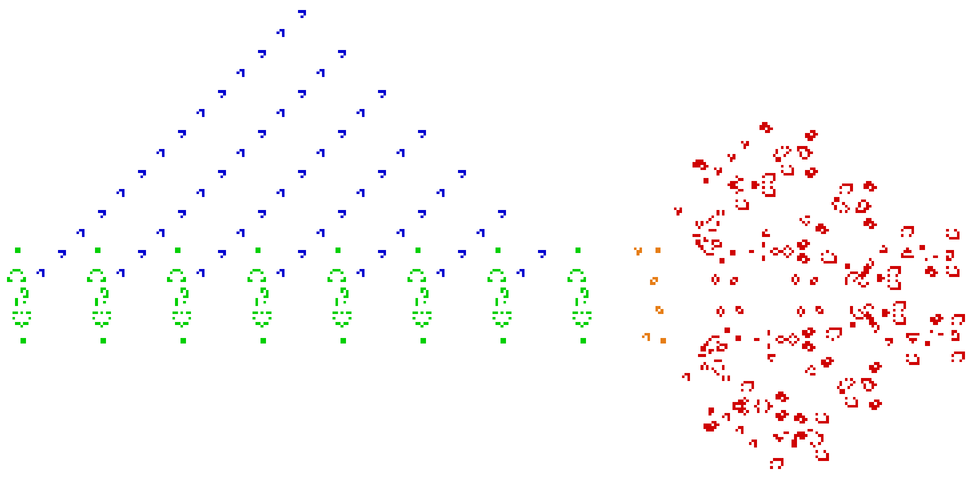
From our MATLAB experiments, we have seen that many seed configurations evolved into a state that is either stationary, or more often, periodic. (Both are considered stable.) Some seed configurations took fewer generations to reach the steady-state, and some had a longer period of chaos before reaching steady-state.

Sometimes there were also isolated patterns that moved across the universe in a certain direction, and at a finite speed. In a larger universe, they would have continued to travel farther without encountering any obstacles. This shows that the diameter (i.e. the diameter of the smallest circle that encloses all the living cells) can grow (linearly) without bound.

Let us consider the entire plane as an infinite universe. It is then natural to wonder about the following: 1) Given a finite seed configuration, can the *number* of living cells (not just their diameter) grow without bound? Conway originally conjectured that the living population was always bounded from above. A counterexample was soon found in the form of a gun:



Note that with the gun as a seed configuration, the number of living cells grows linearly as a function of the number of generations. 2) Is it possible for the growth to be faster? The answer is also affirmative. People constructed breeders, like the trail-leaving puffer train below, that achieve quadratic growth. (This can be seen from the area covered by the patterns shot from the guns.)



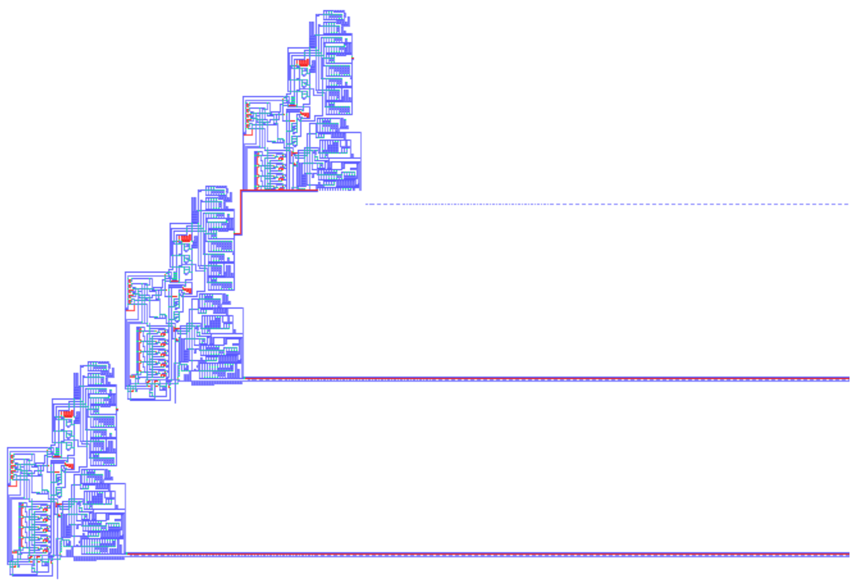
Quadratic growth of the living population is actually optimal. Think about why – it is a consequence of the fact that the rules are local (cells that are alive in the current generation only affect their neighbors in the next generation) and that the universe is 2D.

In the case of both guns and breeders, the configuration evolves in an ordered manner. 3) Are there configurations that continue to remain chaotic forever? The answer is again affirmative.

One then wonders if there is some way to predict the asymptotic behavior of the universe solely from its initial configuration. 4) Given a (finite) seed configuration, is there a general algorithm that can determine if it will eventually become stationary, periodic, ordered, or chaotic? The answer here is unfortunately negative. The Game of Life is undecidable.

We now return to the original theoretical questions that motivated von Neumann to invent cellular automata: 5) Can any computational algorithm be simulated in the Game of Life? The answer, which was discovered about twenty years ago, is, perhaps surprisingly, affirmative. A finite-state machine can be simulated within the Game of Life (guns can simulate electrical signals), so it is computationally universal, or Turing complete.

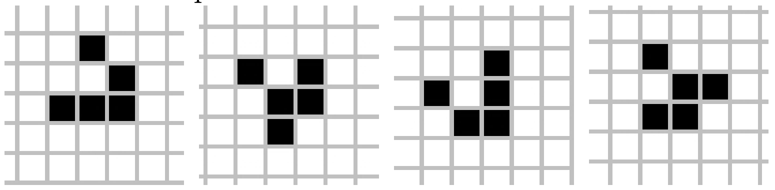
6) Is there a pattern in the Game of Life that can reproduce itself? Research over the past decade has shown that the answer is indeed also affirmative. There are von Neumann universal constructors:



Properties (5) and (6) are von Neumann's criteria for an entity to be *alive.* Do you think the Game of Life is alive because it computes and replicates, just as the entities people usually consider to be biological life on earth? It has the same processing power as any computer, and some believe, the same processing power as the human brain. If don't think it is alive, do you think humans are also not alive? If you do think it is alive, do you think the entire physical universe that we live in is also alive? The Game of Life is a simulation. Are we also living in a simulation? I will let you ponder those philosophical questions for yourselves.

2 The Assignment

For this assignment, you will implement the Game of Life for a 7×7 universe with periodic boundary conditions. As a seed configuration, you should use the glider, shown below. It is an example of a spaceship, a pattern that changes shape and moves across the universe, but returns to its original shape after a certain number of generations. The glider has a period of 4 (the period being the smallest number of generations for a shape to reoccur). You should run your simulation for 56 more generations and produce an animation in the Figure Window. You should display each generation for 0.1 seconds. The glider should move diagonally to the southeast, and wrap around the square and return to its initial position two times.



This programming problem naturally involves arrays, random numbers, loops, comparisons, and plots. In order to give you practice with functions as well, I will force you to write your program in a certain way: you must write and call a function `advance` that takes in two copies of a universe of arbitrary size, and passes them back out so that one configuration is the child (the next generation) of the other configuration. Numbers like 7×7, 56, and 0.1 will, of course, have to be hard-coded outside the function.

Your implementation will be graded only based on its correctness, and not on its elegance or efficiency, although I would encourage you to follow good programming practice, as well as optimize your code for memory usage and running time. You are, however, strongly encouraged to document your code. I will give partial credit for incomplete code or code producing incorrect results, but if does not contain any comments, and it is difficult for me to understand, then you may end up losing a much larger number of points than you otherwise would.

Please place all of your code in a single script file and name it in the form `lastname_firstname.m`, e.g. `yang_oliver.m`. The script should not require any modifications by me, and it should produce the animation when run as is. You do not need to create or submit a video file.