Isaac Gaber

James Marshall

Intermediate Python

15 December 2023

Cellular Automaton

My conference project itself evolved from much experimentation over the course of the semester. I went from basic object-oriented grid-world simulations to exploring other programming languages, finally back onto the grid world. The final iteration of my project is a collection of various cellular automata, as will as a basic user interface and the ability to load different files from patterns

NOTE:

This project requires the Pygame framework to be installed, which is not included in the files. The latest version can be installed with “python -m pip install -U pygame==2.5.2 –user”. Further instructions can be found on pygame.org.

gui.py:

This file contains the file manager for the models, as well as a simple GUI made using entirely ASCII in the Pygame framework. Run this file to explore the different automata.

automata.py:

The “automata.py” file contains all the different automaton simulations, which are stored as classes. The methods and attributes of the different simulations are contained within. The object-

oriented structure of the project allows for easy creation of new automaton models. There are two currently two superclasses for the Automata: “Automata”, and “Automata3D”. The Automata superclasses contain methods to intialize the simulation, step the simulation, check the state of any cell’s neighbors in their Moore or Von Neumann neighborhoods, and to format and return lists of pygame surfaces to be rendered by the blitter.

The “Life” simulation is one of the simplest of the simulations, replicating the standard rules for the game of life. It is also the simulation most suited to loading in different patterns and exploring their evolution, due to the binary nature of the simulation and the plethora of patterns available for viewing online. When initialized to a random board, the game of life generally exhibits few flyers, and is on the less chaotic side of the simulations.

The “Wire-World” simulation is currently non-functional, but if it was, would allow for rudimentary simulation of electrical circuits.

The “Brian’s Brain” automaton was invented by Brian Silverman and has very similar rules to the Game of Life simulation, except that in addition to dead or alive, cells may also be dying. This simple change results in a far greater number of flyers, and a much more chaotic overall simulation

The “Seeds” automata is also credited to Brian Silverman, and results in few fliers, but very quickly fills the screen in a chaotic mess.

The “Life-ish smooth automata” is my own creation, inspired by the “Smooth life automaton” in the Netlogo model suite. The model attempts to replicate some features of the game of life continuously, and while it exhibits some spinning and pulsing patterns, it doesn’t form fliers to the same extent at all, and requires further tweaking.

The “Game of Life 3D” is a simple transposition of the Game of Life’s rules into three dimensions. As such, it behaves wildly differently than the two dimensional version, and requires further tweaking.

“Simple Fluid Model 2D” models a basic two dimensional fluid. The cells start with a random fluid quantity, then as time steps progress, pass fluid to their neighbors. Eventually, they find their own level.

Finally, “Simple Fluid Model 3D” models a very similar but slightly different fluid but in 3D. The cells also begin with a random distribution of fluid which flows to their neighbors over time. However, the amount of fluid in each cell can be compressed slightly, allowing the fluid to find its own level, something the 2D fluid does not have implemented due to the fluid’s top-down perspective.

patterns_2d.txt:

When this file is present in the same path as the automata.py file, 2d automata simulations will by default make use of this file when they are loaded. While the automata’s default behavior is to initialize the state to a random grid of set density, when this file is present the automata will instead attempt to read the text file as a set of cell states. Then, it will write those states, starting in the center of the field. No equivalent loading system exists for the 3D automata, as the nature of text documents make visualizing the final pattern extremely difficult. In addition, finding interseting patterns for 3D life models is substantially more difficult than in two dimensions.

To return the 2D automata to their default behavior, simply delete or move the patterns_2d.txt file form the directory. I would liked to have made a better GUI that allowed the user to select different preset patterns to load, but I ran into time constraints, and decided it wasn’t worth the added complexity to write.