Game of Life Lab

Dr S. S. Chandra

The Game of Life (GoL) simulation is a cellular automation originally developed by John Conway. The game involves a set of cells within an $N \times N$ grid whose state is either alive or dead (i.e. 0 or 1 respectively). The grid effectively represents the 'universe' that will be simulated and the alive cells the life within it. The game is governed by a set of simple rules that dictate the next state of each cell in this universe depending on its current state and the states of its neighbours. The rules of GoL depend on the 8-connected neighbours of a cell as follows:

1. **Underpopulation:** A live cell that has < 2 live neighbouring cells will die
2. **Survival:** A live cell that has 2-3 live neighbouring cells will remain alive
3. **Overpopulation:** A live cell with more than 3 live neighbours will die
4. **Reproduction:** A dead cell with exactly 3 live neighbours will become alive

The game begins with an initial state of the universe with a pattern of live cells. The universe is evolved by applying the above rules to each cell of the universe to determine the next iteration of the simulation. The evolution of the universe is observed by continual computing the next iteration of the universe. See chapter 7, section 7.6.4 of (Moore and Mertens, 2011) for more theoretical details.

In this laboratory, you will create a simulation of the GoL using Python based on an initial class provided. In the following parts of the lab, you will be required to code up the algorithms related to the computation, importation and evaluation of the GoL.

Part I – Game of Life Simulation (8 Marks)

An initial class called "conway.py" is provided with the necessary hooks required for this part of the lab. An example test script is provided that enables the animation of the simulation. Another script is also provided without animation for debugging purposes, especially for implementing the GoL rules.

[See scripts `conway.py`, `test_gameoflife_glider_simple.py` and `test_gameoflife_glider.py`]

a) Implement the four GoL rules as mentioned above in the relevant parts of the `conway.py` module and test your simulation on the 'blinker' initial pattern. You may use the 'simple' script first to ensure your algorithm is working correctly.

(2.5 Marks)

b) Change the initial pattern to the glider (already implemented in `conway.py`) and run the animation to verify that the rules are working correctly. How can you tell your code is working correctly?

(1/2 Mark)

c) Change the initial pattern to the glider gun (already implemented in `conway.py`) and run the animation. What should you get and what is wrong? Fix the glider gun pattern so that its runs correctly.

(1 Mark)

d) Construct different patterns from the LifeWiki for the `conway.py` module by implementing a plaintext or run length encoded (RLE) reader for the module. Demonstrate multiple initial patterns that are greater than 20x20 in size.

(2 Marks)

e) Implement a fast method for computing the weights for the rules based on convolution and run a really large simulation ($N > 1024$) with an appropriately large pattern (at least of the order of $N/4$ or one that is acceptable to your demonstrator).

(2 Marks)

Part II – Langton's Ant (7 Marks)

Langton's Ant is a cellular automation that is Turing complete and is capable of chaotic behaviour. In this universe, a single ant is allowed to roam a space consisting of $N \times N$ squares according to a set of rules. There are various rules known, but one of the simplest ones is the following:

1. **If on a white square**, toggle the colour of the square and move to the square on the right.
2. **If on a black square**, toggle the colour of the square and move to the square on the left.

By toggling, we mean change the colour to the next colour in a given sequence. For a binary colour system, this is simply toggling between black and white.

[No initial code is provided]

[You will also need to submit your code on Turn-it-in to receive any marks for this part of the lab]

a) Implement Langton's Ant in Python either with your own code or re-using some of the code in the GoL part of this lab.

(3 Marks)

b) Show that your ant is capable of demonstrating the following scenarios:
   i. Chaotic behaviour
   ii. The highway

(2 Marks)

c) Improve the implementation of your ant with colours for various states on squares.

(2 Marks)

Interesting Links

Video of an 8-bit Programmable computer in GoL!
https://www.youtube.com/watch?v=8unMqSp0bFY

Various GoL and Langton's Ant videos on the course Computation YouTube Playlist
https://www.youtube.com/playlist?list=PLC0kkV5axv-X3JOXeHGoedTMCXGakoYmt

References

Moore, C., Mertens, S., 2011. The Nature Of Computation. Oxford University Press.