Turing Completeness and the Game of Life

Dr S. S. Chandra

The Game of Life (GoL) simulation can be thought of as a form of 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. 1 or 0 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.

Important Notes

For this practical you will need to install/already have numpy, scipy and matplotlib on your machine. Use either Anaconda Python or WinPython to have these setup for you quickly and hassle free. See my video series on setting up Python environments on Windows for help.

For this practical, we will only accept solutions in Python and all animations must be in matplotlib (not tkinter or turtle, etc).

[This is the lab sheet for the Game of Life Demonstration. You must attempt as many tasks (ideally all tasks) AND demonstrate it to the instructor in one of your practical sessions BEFORE the due date in order to be awarded marks. Please check the ECP for the correct due date. Note that tasks are 'complete' and marks are awarded by attempting each task AND correctly answering related questions to the satisfaction of the instructor.]

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 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 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. Hint: One of the lines for the glider gun member is incorrectly alive.

(2 Marks)

d) Construct different patterns from the LifeWiki for the `conway.py` module by implementing a plaintext reader for the module as a `insertFromPlainText()` member (see stub provided). This member should accept a string of the pattern in human readable form as defined by the format as a single string (as provided by the standard Python file reader after suitably handling comments etc.). Demonstrate multiple initial patterns that are greater than 20x20 in size.

(3 Marks)

Part II – Turing Completeness of the Game of Life Simulation (7 Marks)

An initial class called "rle.py" is provided with the necessary hooks required for this part of the lab. An example test script is provided that enables the running of the relevant patterns.

[See scripts `conway.py`, `rle.py` and `test_gameoflife_turing.py`]

e) Implement a fast method for computing the weights for the rules based on convolution and run a 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)

f) Construct different patterns from the LifeWiki for the `conway.py` module by implementing run length encoded (RLE) reader for the module as a `insertFromRLE()` member (see stub provided) using the `rle.py` module provided. This member should accept a string of the pattern in run length encoded form as defined by the format as a single string (as provided by the standard Python file reader). Demonstrate multiple initial patterns that are greater than 20x20 in size.

(2 Marks)

g) Demonstrate a running GoL Turing Machine pattern by using your RLE reader from the previous section to load and run the pattern.

(1 Mark)

h) Given the Turing machine pattern runs within GoL, comment on whether GoL is Turing complete. Justify your answer by referencing the theory of Turing machines and the different components of the Turing machine pattern provided using this link.

(2 Marks)

Interesting Links

Shakes' Windows Deep Learning Python Setup Series
https://www.youtube.com/playlist?list=PLC0kkV5axv-X4OpBHlIPlIz15XNNGd3OE

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.