

Cellular Automata with NetLogo

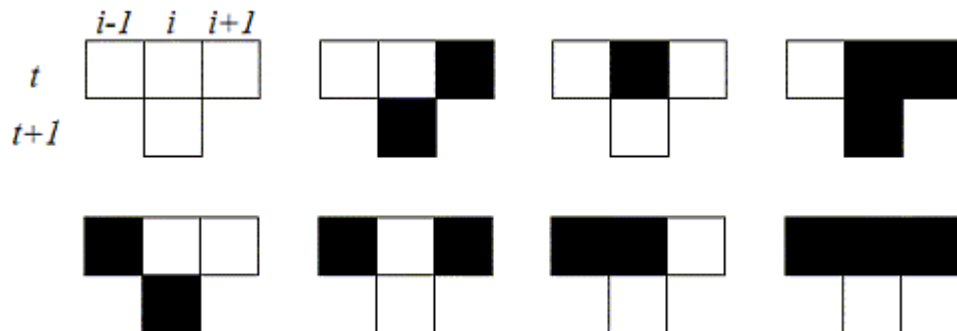
Tutorial by Nic Geard

1. [Background](#)
 - o Cellular Automata
 - o The Game of Life
 - o Further Information
2. [NetLogo](#)
 - o Introduction
 - o 1-Dimensional Cellular Automata
 - o 2-Dimensional CAs: The Game of Life
 - o Predator-Prey Model
 - o Starting from scratch
 - o Further Suggestions

Background

Cellular Automata

- Originally developed by John von Neumann in the 1940s.
- One-dimensional cellular automata are discrete dynamical systems that exist on a row of cells.
- The state (**on** or **off** in the simplest case) of cell i is updated at each time step by considering the previous state of cell i and its neighbours, cell $i-1$ and cell $i+1$.
- The rules for updating can be described by a lookup table, where the top three cells represent the current time step, and the bottom cell represents the future state :



- The property of interest in cellular automata is generally their long term behaviour. That is, the patterns that the generated after initial transient behaviour is discounted.
- As there are eight rules per set, and each rule has one of two possible outputs, there exist 256 possible rule sets.
- Stephen Wolfram investigated all 256 possible rule sets and described three classes of long term behaviour :
 - o Class 1 - system quickly settles to a boring configuration (all cells on/off).

- Class 2 - system settles to a stable state (either a point or cyclic attractor).
- Class 3 - system highly disordered with no discernible patterns ('chaos').
- Class 4 - system displays 'interesting' complex behaviour with some level of order.
- Some class 4 cellular automata are capable of propagating information and therefore may be used to perform computation. Much has been made of the fact that rule 110 can be proven to be a universal computer.

The Game of Life

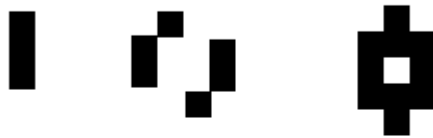
- Invented by mathematician John Conway in 1970. [Link](#) to the original 1970 *Scientific American* article.
- Life is two-dimensional cellular automata - it is played on a square grid, where cells are either **alive** or **dead**. At each time step the state of each cell is updated depending on how many neighbours it has :
 - If a dead cell has three or more neighbours, it becomes alive (birth).
 - If a live cell has two or three neighbours, it remains alive (survival).
 - If a live cell has less than two, or more than three neighbours, it dies (loneliness or overcrowding).
- Over time, a variety of different classes of behaviour can result, depending upon the initial state of the system :
 - Everything dies.
 - The system settles to a stable state.
 - The system settles to an oscillating, or cyclic, state.
 - The size of the system continues to increase.
- Life is one of the simplest examples of a self-organising system, in which a few simple rules result in a wide range of emergent behaviours.

- Several different initial conditions resulting in various classes of behaviour are illustrated below :

Still Life



Oscillators



Gliders



Other interesting shapes



Further Information

- [Stephen Wolfram's Homepage: A New Kind of Science](#)
- [Wolfram's earlier publications](#)
- [Evolving CAs to perform computation](#) (Santa Fe - more advanced)
- [Introduction to the Game of Life](#)
- [Another Life page with lots of patterns, software and links](#)

NetLogo

Introduction

- "NetLogo is a programmable modeling environment for simulating natural and social phenomena. It is particularly well suited for modeling complex systems developing over time. Modelers can give instructions to hundreds or thousands of independent 'agents' all operating in parallel. This makes it possible to explore the connection between the micro-level behavior of individuals and the macro-level patterns that emerge from the interaction of many individuals." (from the manual)
- NetLogo homepage : <http://ccl.northwestern.edu/netlogo/>
- NetLogo documentation (including tutorials and reference) : <http://ccl.northwestern.edu/netlogo/docs/>

1-Dimensional Cellular Automata

- Open the 1D-CA model by selecting 'Models Library' from the File Menu and choosing 'Computer Science :: Cellular Automata :: Cellular Automata 1D' from the folder list.
- Press 'Setup Single' to turn on a single cell in the centre of the system (displayed on the top row of the output window) and then 'Go' to propagate the activity of the system through time (towards the bottom of the window).
- Press 'Setup Random' to reset the system to a random initial condition, and then 'Go' to observe what ensues.
- The default rule is **1 0 0 1 0 1 1 0**. Press 'Show Rules' to illustrate this rule graphically at the top of the output window.
- Change the rule to **1 0 0 1 0 1 1 1** by selecting the bottom right rule switch. Press 'Setup Random' and 'Go' again to observe how the behaviour changes.
- Rule sets can also be chosen using the slider bar. Try rules 22, 31, 77, 104, 149 and 169 (among others) to see a variety of different behaviours
- More suggestions of features to notice and things to try can be found by selecting the **Information** tab.

2-Dimensional Cellular Automata : The Game of Life

- Open the Game of Life model (Computer Science :: Cellular Automata :: Life).
- Press 'Setup Blank' to clear the output screen.
- Using 'Add Cell' and 'Remove Cell' draw one of the formations illustrated above.
- Press 'Go Once' a couple of times to iterate the system.
- If the system is continuing to grow, press 'Go Forever' to see where it ends up. (Press 'Go Forever' again to stop the system running).
- Experiment with some of the other shapes above.

- **Changing the rules of Life!**

- Select the 'Procedures' tab to open up the source code for the model.
- First, note how short and clear the code is (less than 50 lines excluding comments, high-level syntax).
- Scroll down to the procedure beginning "to go". The section of code reproduced below defines the rules of Life :

```
ifelse live-neighbours = 3
  [cell-birth]
  [if live-neighbours != 2
    [cell-death]]
```

- These lines can be read as "If a cell has 3 neighbours, it becomes alive, else if it doesn't have two neighbours (i.e., it has 0, 1 or more than 3), it dies, otherwise it stays the same.
- Try changing the conditions for birth (e.g. so that 4 live neighbours are required), or the conditions for survival (e.g. so that at least 3 neighbours are

required) and notice just how finely balanced Conway's rules are. (Note: changes to the source code take effect as soon as you switch back to the **Interface** window)

Predator-Prey Model

- Open the model (Biology :: Wolf Sheep Predation).
- Setup and run the model a few times with the default parameters. Observe the possible outcomes.
- Try changing the parameters so that the population levels settle into a stable oscillation.
- Using the toggle switch, add grass to the model and observe what effect this increased complexity / noise has on the stability of the system.
- More suggestions of things to try can be found in the **Information** tab.

Starting from scratch

- Select 'New' in the file menu to start a new model
- Click on 'Button' in the tool bar and then place the button in the white area of the **Interface** tab.
- When the properties dialog appears, type `setup` in the 'code' box.
- Press 'OK'. This button will now run the procedure 'setup' when pressed. Notice how the **Error** tab has turned red. Select it to view the error.
- Select the **Procedures** tab and enter the following :

```
to setup
  clear-all
  create-turtles 100
  ask turtles
    [forward random screen-edge-x]
end
```

- In English...
 - `to setup` defines a procedure named 'setup'.
 - `clear-all` clears the screen, initialises variables to 0 and removes any turtles.
 - `create-turtles 100` creates 100 new turtles in the centre of the screen. Each turtle will have its own colour and heading.
 - `ask turtles` specifies that the following code (between the square brackets) is to be executed independently by each turtle.
 - `forward random screen-edge-x` is two nested commands. `random screen-edge-x` returns a random value between 0 and 'screen-edge-x', the maximum value on the screen along the x axis. `forward` tells the turtle to move forward this many steps.
 - `end` indicates the end of the procedure.
- Switch back to the **Interface** tab and press the 'setup' button. So long as there are no errors, the turtles should spread out over a roughly circular region of the screen.
- To make the turtles do more interesting things, more directions are contained in the documentation in Tutorial #3 (accessible through the help menu).

Further Suggestions

- Explore some of the other biological models, such as Ants, Fireflies, Flocking and Termites.
- Instructions, information and suggestions for each model can be obtained from the **Information** tab.
- Work through the tutorials in the documentation for a better understanding of how to implement your own models using the NetLogo scripting language.