game of life manual

Steven Brawer
2/16/17

getting started

choose a pattern and run

- **Select a pattern** - say, *blinker* - from the ***patterns*** *drop-down list*. This adds some dots to the grid.
- **Click run.**

The pattern changes. The changes do not occur smoothly, but after some time interval. We say **the pattern changes in steps, or ticks.** At the slowest speed (which is the default when you start up), the change occurs twice a second, or every 1/2 second.

to change speed

- **Select a speed** from the ***speed*** *drop-down menu* - for example, *faster*.
- **Click run**.

The pattern now steps at a rate determined by the speed you selected.

other controls

to run manually

If you want to follow closely what is going on, **click tick**. (**If you are already running, click pause first, then click tick.**) This makes **one step** - a single application of the rules. Click *tick* again and again to step through the evolution of the pattern. You may click *tick* as rapidly as you want, and the pattern will change each time.

to pause, change things, do something, then step or run

- **Click pause.**
- You may **change speed** using the ***speed*** *drop down menu*.
- You may **add or delete dots** - see below.
- You may click **tick** for step-by-step (manually) running.
- You may **save** the pattern (see below)
- To start running again, click r**un.**

to choose a different start pattern

- Choose the pattern from the ***pattern*** *drop-down menu*. (This will set the speed to slowest, and will erase all existing dots and add the new dots of your selected pattern.)
- Select a different speed if desired.
- **Add or delete dots** as desired.
- **Click run.**

to reset everything

click reset. This puts the system in the same state it had when initially loaded. The speed is the slowest, there is no selected pattern (so no dots on the graph) and nothing is running.

to manually add or delete dots - create your own pattern

- To start with a fresh graph, click **reset**. You can now create your own pattern from scratch.
- Or you may add or delete dots to/from the existing graph.
- To **add dots**, click on an empty box.
- To **delete a dot,** click on the dot (which is black and will then become transparent).
- You can now **change speed, tick, save or run.**

the random patterns

The **pattern drop-down menu** has four patterns named **random1,…, random4.** These are not stored patterns but are generated randomly each time such a pattern is chosen. The number and placement of points are generated randomly, where the number of points is chosen randomly and increases in order from *random1* to *random4*. Once run, the pattern will be in the *run list,* as described later, where you can retrieve it if it turns out to be interesting.

model, rules, patterns and discussion

the rules of the game

This is a game that models the growth or decline of populations. Each **black dot** represents a group of animals living in the same area. Each **light dot** represents no animals in that area. The game models the change in population - ie, the increase or decrease in the number of animals, in a given area (on a given dot) due to reproduction. The basic idea is given below.

I stress that Game of Life is a very simple model of animal populations, and is not used for any realistic research. The model has become of interest to mathematicians and programmers

because of the complexity of behavior generated by very simple rules, not because of biological significance. Google it to see the large literature on this subject.

Here is the animal population model:

(1) An animal population (not an individual animal, but a population) will maintain itself if there are not-too-few and not-too-many neighbors - animals, living nearby (combines reproduction and death).

(2) If there are a proper number of animals nearby, an empty area will get a new population (reproduction).

(3) An animal population will decrease if there are too many animals (crowding) or too few animals (sparse population) living nearby. This is **death**. Crowding gives rise to disease, wars (among humans), and animals just moving away. In this model, scarcity leads to less reproduction and even more population decline.

Here are the rules of the graph, which model the above concepts. In all of this, the **number of neighbors of a dot** are the number of black dots next to the given dot - horizontally + vertically + on the diagonal (examples below). Black dots are called **live** areas (representing animals). Light dots are called **dead** areas (no animals there). The numbers below correspond to the numbers above.

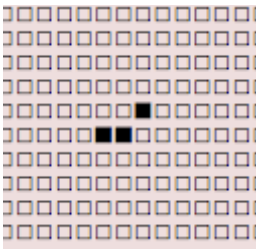
(1). If a black dot has 2 or 3 neighbors, it remains a black dot (lives) - that is, the population maintains itself.

(2). If a light dot has exactly 3 neighbors, it becomes a black dot (becomes alive). A population increase.

(3) If a black dot has 2 or less or 4 or more neighbors, it becomes a light dot (dies).

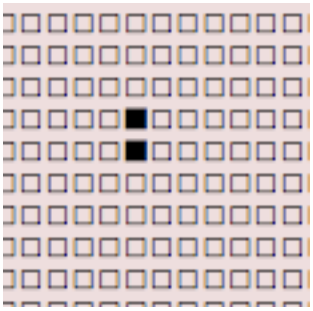
Otherwise, there is no change in the life or death status of a given dot.

Some examples:



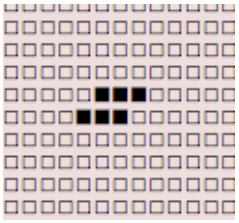
The middle black dot has two neighbors, one on the horizontal, one on the diagonal. On the next step it remains black (remains alive). The other two black dots have only one neighbor each, so on the next step both will become light dots (dead). The light dot above the middle

black dot has 3 live neighbors (one horizontal, one vertical, one on the diagonal), so on the next step it will become a black dot. The result of one step is:



On the next step after this, both will die (why?), so the graph becomes blank.

Here is the **toad** pattern, a built-in pattern of this application. I will just list some of the neighbors.



Let's label the dots:

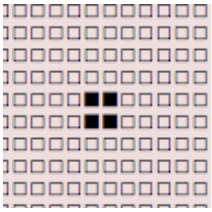
A B C D - an all-light row
E F G H - top black squares
I J K L - next black squares
M N O P - an all-light row below I J K L

F, G, H, I, J, K are black dots, the rest are light. Then A has 1 live neighbor, B has 2 live neighbors, C has 3 live neighbors, N has 3 live neighbors, J has 4 live neighbors, K has 4 live neighbors, L has 3 live neighbors, and so forth. See if you can figure out what happens next, based on the rules. You can see the result in the game itself. Use the *tick*.

types of patterns

Applying the rules to even a simple pattern can lead to very complex situations. Try running the **acorn**, **Rpentomino** and **flower** patterns. It's fun to do it at a fast speed. There are a few basic types:

static - there are some patterns which stay live but do not change. One example is



This is called the **block**. See if you can figure out why it does not change.

oscillators - some patterns move between 2 or 3 patterns and come back to the original one, so they oscillate. The **blinker** and **toad** patterns are examples.

gliders - Some patterns move across the graph. These oscillate among a few patterns, but they also move. The **glide** pattern is an example.

complicated - that says it all. These are the rest.

There are also **spaceship** patterns, **cloud** patterns, and so forth. You can google these on the web.

saving patterns, retrieving saved patterns

There are three separate storage areas. Two are temporary and go away when the browser is closed. The other is long-term browser storage (called localStorage).

the run list

Every time **run** is clicked, before any change, the pattern on the graph is saved to a **run list.** This is a FIFO list holding about 30 patterns, which maintains a marker. The patterns are not named. To retrieve a pattern, do **storage -> get from run**. Every time you do this, you will get the pattern which is pointed to by the marker, which then moves to point to the next list item. Eventually you come back to the starting point. If you click *run*, the marker goes to the start of the list.

Click **storage->clear run** to empty this list.

The run list is a kind of backup. You might have created a pattern, run it, and found out after a while that the pattern is interesting. So long as you have not clicked *run* more than 30 times since the start, this initial pattern will be available in the *run list.*

The save list

To explicitly save the current pattern on the graph, click **storage->save to list**. This saves the pattern to the **save list**, which is a circular list as described above. To retrieve patterns from this list, do **storage->get from list**. Do **storage->clear list** to clear this list.

You might be experimenting with patterns. You might be using *tick* to move forward, to see what happens. The initial pattern may become interesting after a few ticks. Use this to save your initial pattern, before ticking.

The pattern list

The **pattern list** contains **named patterns**, and the named patterns appear in the **pattern drop-down menu**. If you want to save a pattern as a *named pattern*, so it shows up in the pattern drop-down, do **storage->save as pattern**. A screen pops up where you can enter your name for the pattern. **storage->delete pattern** allows you to delete one of the patterns that you have saved. The pattern drop-down has a number of **core patterns** which appear on start-up, are always first in the list, and which cannot be deleted. **storage->clear patterns** deletes all the user-named patterns.

setting up safari to run from a file

—-These directions are meant for my grandchildren, not for programmers.

Move the game file to any directory. This is an ordinary text file, so do *not* put it in the applications folder. Make sure file restrictions are *Read by everyone*. **It uses the internet.**
Double click the file in Finder. Or do in Safari: *File -> open file,* and select the file for the game. This should work.
If this does not work, proceed as follows:
Safari->preferences->Advanced tab. On the bottom, check *Show develop menu in menu bar.*
In the *Develop* menu, turn on *Disable Local File Restrictions.* (Be sure to turn this off later.)
The try *File -> open file.*