

## Chapter 9

# Game of Life

*Conway's Game of Life makes use of sparse matrices.*

The “Game of Life” was invented by John Horton Conway, a British-born mathematician who is now a professor at Princeton. The game made its public debut in the October 1970 issue of *Scientific American*, in the “*Mathematical Games*” column written by Martin Gardner. At the time, Gardner wrote

This month we consider Conway's latest brainchild, a fantastic solitaire pastime he calls “life”. Because of its analogies with the rise, fall and alternations of a society of living organisms, it belongs to a growing class of what are called “simulation games” – games that resemble real-life processes. To play life you must have a fairly large checkerboard and a plentiful supply of flat counters of two colors.

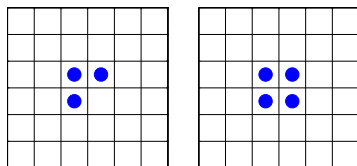
Of course, today we can run the simulations on our computers.

The *universe* is an infinite, two-dimensional rectangular grid. The *population* is a collection of grid cells that are marked as *alive*. The population evolves at discrete time steps known as *generations*. At each step, the fate of each cell is determined by the vitality of its eight nearest neighbors and this rule:

- A live cell with two live neighbors, or any cell with three live neighbors, is alive at the next step.

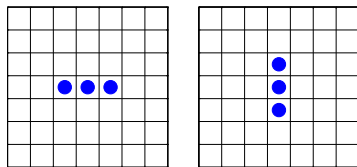
The fascination of Conway's Game of Life is that this deceptively simple rule leads to an incredible variety of patterns, puzzles, and unsolved mathematical problems – just like real life.

If the initial population consists of only one or two live cells, it expires in one step. If the initial population consists of three live cells then, because of rotational



**Figure 9.1.** *A pre-block and a block.*

and reflexive symmetries, there are only two different possibilities – the population is either L-shaped or I-shaped. The left half of figure 9.1 shows three live cells in an L-shape. All three cells have two live neighbors, so they survive. The dead cell that they all touch has three live neighbors, so it springs to life. None of the other dead cells have enough live neighbors to come to life. So the result, after one step, is the population shown in the right half of figure 9.1. This four-cell population, known as the *block*, is stationary. Each of the live cells has three live neighbors and so lives on. None of the other cells can come to life.



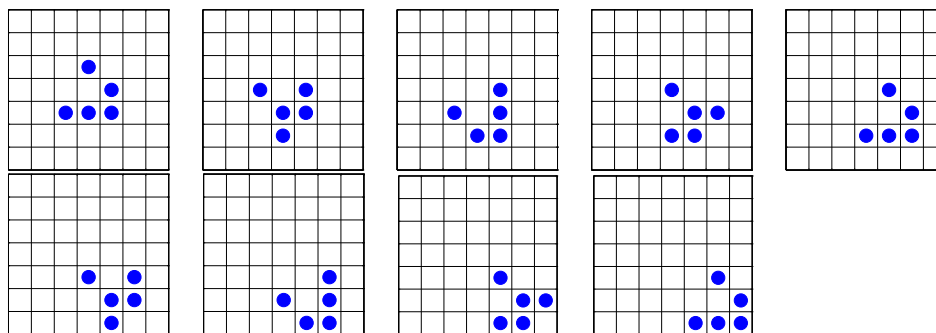
**Figure 9.2.** *A blinker blinking.*

The other three-cell initial population is I-shaped. The two possible orientations are shown in each half of figure 9.2. At each step, two end cells die, the middle cell stays alive, and two new cells are born to give the orientation shown in the other half of the figure. If nothing disturbs it, this *blinker* keeps blinking forever. It repeats itself in two steps; this is known as its *period*.

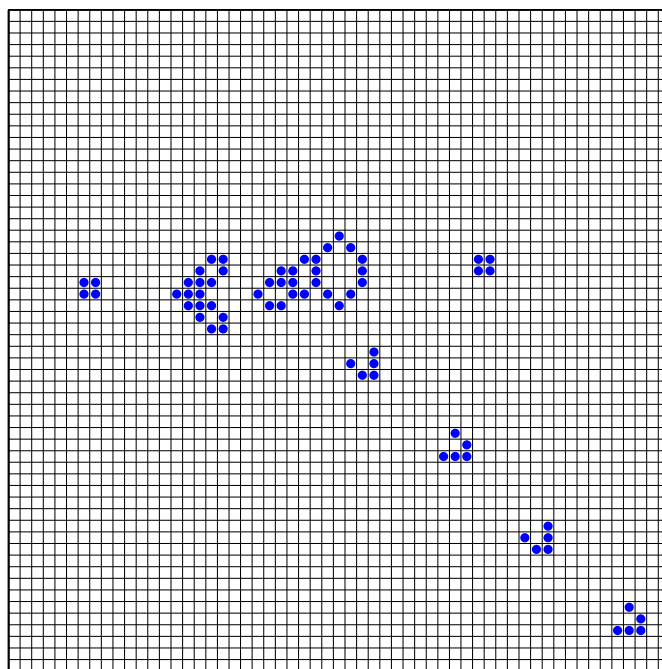
One possible four-cell initial population is the block. Discovering the fate of the other four-cell initial populations is left to an exercise.

The beginning of the evolution of the most important five-cell initial population, known as the *glider*, is shown in figure 9.3. At each step two cells die and two new ones are born. After four steps the original population reappears, but it has moved diagonally down and across the grid. It continues to move in this direction forever, eventually disappearing out of our field of view, but continuing to exist in the infinite universe.

The fascination of the Game of Life cannot be captured in these static figures. Computer graphics lets you watch the dynamic development. We will show just more one static snapshot of the evolution of an important larger population. Figure 9.4 is the *glider gun* developed by Bill Gosper at MIT in 1970. The portion of the population between the two static blocks oscillates back and forth. Every 30 steps, a glider emerges. The result is an infinite stream of gliders that fly out of the field of view.



**Figure 9.3.** *A glider gliding.*



**Figure 9.4.** *Gosper's glider gun.*

MATLAB is a convenient environment for implementing the Game of Life. The universe is a matrix. The population is the set of nonzero elements in the matrix. The universe is infinite, but the population is finite and usually fairly small. So we can store the population in a finite matrix, most of whose elements are zero, and increase the size of the matrix if necessary when the population expands. This is the ideal setup for a *sparse* matrix. Conventional storage of an  $n$ -by- $n$  matrix requires  $n^2$  memory. But sparse storage of a matrix  $X$  requires just three vectors, one integer and one floating point vector of length `nnz(X)` – the number of nonzero

elements in  $X$  – and one integer vector of length  $n$ , not  $n^2$ , to represent the start of each column. For example, the snapshot of the Gosper glider gun in figure 9.4 is represented by an 85-by-85 matrix with 68 nonzero entries. Conventional full matrix storage would require  $85^2 = 7225$  elements. Sparse matrix storage requires only  $2 \cdot 65 + 85 = 221$  elements. This advantage of sparse over full storage increases as more gliders are created, the population expands, and  $n$  increases.

The `exm` toolbox includes a program called `lifex`. (MATLAB itself has a simpler demo program called `life`.) The initial population is represented by a matrix of 0's and 1's. For example,

```
G = [1 1 1; 1 0 0; 0 1 0]
```

produces a single glider

```
G =
     1     1     1
     1     0     0
     0     1     0
```

The universe is represented by a sparse  $n$ -by- $n$  matrix  $X$  that is initially all zero. We might start with  $n = 23$  so there will be a 10 cell wide border around a 3-by-3 center. The statements

```
n = 23;
X = sparse(n,n)
```

produce

```
X =
All zero sparse: 23-by-23
```

The initial population is injected in the center of the universe. with the statement

```
X(11:13,11:13) = G
```

This produces a list of the nonzero elements

```
X =
(11,11)     1
(12,11)     1
(11,12)     1
(13,12)     1
(11,13)     1
```

We are now ready to take the first step in the simulation. Whether cells stay alive, die, or generate new cells depends upon how many of their eight neighbors are alive. The statements

```
p = [n 1:n-1];
q = [2:n 1];
```

generate index vectors that increase or decrease the centered index by one, thereby accessing neighbors to the left, right, up, down, and so on. The statement

$$Y = X(:,p) + X(:,q) + X(p,:) + X(q,:) + \dots \\ X(p,p) + X(q,q) + X(p,q) + X(q,p)$$

produces a sparse matrix with integer elements between 0 and 8 that counts how many of the eight neighbors of each interior cell are alive. In our example with the first step of the glider, the cells with nonzero counts are

$$Y = \begin{array}{llll} (9,9) & 1 & (10,9) & 2 \\ (11,9) & 2 & (12,9) & 1 \\ (9,10) & 2 & (10,10) & 2 \\ (11,10) & 3 & (12,10) & 2 \\ (13,10) & 1 & (9,11) & 3 \\ (10,11) & 3 & (11,11) & 5 \\ (12,11) & 1 & (13,11) & 1 \\ (9,12) & 2 & (10,12) & 1 \\ (11,12) & 3 & (12,12) & 1 \\ (13,12) & 1 & (9,13) & 1 \\ (10,13) & 1 & (11,13) & 1 \end{array}$$

The basic rule of Life is

A live cell with two live neighbors, or any cell with three live neighbors, is alive at the next step.

This is implemented with the single MATLAB statement

$$X = (X \& (Y == 2)) \mid (Y == 3)$$

The two characters == mean “is equal to”. The '&' character means “and”. The '|' means “or”. These operations are done for all the cells in the interior of the universe. In this example, there are four cells where Y is equal to 3, so they survive or come alive. There is one cell where X is equal to 1 and Y is equal to 2, so it survives. The result is

$$X = \begin{array}{ll} (11,11) & 1 \\ (12,11) & 1 \\ (10,12) & 1 \\ (11,12) & 1 \\ (12,13) & 1 \end{array}$$

Our glider has taken its first step.

One way to use `lifex` is to provide your own initial population, as either a full or a sparse matrix. For example, you create your own fleet of gliders fleet with

$$G = [1 \ 1 \ 1; \ 1 \ 0 \ 0; \ 0 \ 1 \ 0]$$

```
S = sparse(15,15);
for i = 0:6:12
    for j = 0:6:12
        S(i+(1:3),j+(1:3)) = G;
    end
end
lifex(S)
```

The Web page

[http://www.argentum.freemove.co.uk/lex\\_home.htm](http://www.argentum.freemove.co.uk/lex_home.htm)

is the home of the “Life Lexicon”, maintained by Stephen Silver. Among thousands of the facts of Life, this 160-page document lists nearly 450 different initial populations, together with their history and important properties. We have included a text copy of the Lexicon with the `exm` toolbox in the file

`exm/lexicon.txt`

`Lifex` can read initial populations from the Lexicon. Calling `lifex` with no arguments,

```
lifex
```

picks a random initial population from the Lexicon. Either

```
lifex('xyz')
```

or

```
lifex xyz
```

will look for a population whose name begins with `xyz`. For example, the statements

```
lifex pre-block
lifex block
lifex blinker
lifex glider
```

start with the simple populations that we have used in this introduction. The statement

```
lifex Gosper
```

provides Gosper’s glider gun.

By default, the initial population is surrounded by a strip of 20 dead border cells to provide a viewing window. You can change this to use `b` border cells with

```
lifex('xyz',b)
```

If the population expands during the simulation and cells travel beyond this viewing window, they continue to live and participate even though they cannot be seen.

---

## Further Reading

The Wikipedia article is a good introduction.

[http://en.wikipedia.org/wiki/Conway's\\_Game\\_of\\_Life](http://en.wikipedia.org/wiki/Conway's_Game_of_Life)

Another good introduction is available from Math.com, although there are annoying popups and ads.

<http://www.math.com/students/wonders/life/life.html>

If you find yourself at all interested in the Game of Life, take a good look at the Lexicon, either by reading our text version or by visiting the Web page.

[http://www.argentum.freeseerve.co.uk/lex\\_home.htm](http://www.argentum.freeseerve.co.uk/lex_home.htm)

## Exercises

9.1 *Four-cell initial populations.* What are all of the possible four-cell initial populations, and what are their fates? You can generate one of the four-cell populations with

```
L = [1 1 1; 1 0 0];  
lifex(L,4)
```

9.2 *Lexicon.* Describe the behavior of each of these populations from the Lexicon. If any is periodic, what is its period?

```
ants  
B-52  
blinker puffer  
diehard  
Canada goose  
gliders by the dozen  
Kok's galaxy  
rabbits  
R2D2  
spacefiller  
wasp  
washerwoman
```

9.3 *Glider collisions.* What happens when:

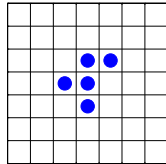
- A glider collides head-on with a block?
- A glider side-swipes a block?
- Two gliders collide head-on?

Two gliders clip each others wings?  
 Four gliders simultaneously leave the corners of a square and head towards its center?

See also: `lifex('4-8-12')`.

9.4 *Factory*. How many steps does it take the `factory` to make one glider?

9.5 *R-pentomino*. Of all the possible five-cell initial populations, the only one that requires a computer to determine its ultimate behavior is the one that Conway dubbed the *R-pentomino*. It is shown in figure 9.5 and can be generated by

$$R = [0 \ 1 \ 1; \ 1 \ 1 \ 0; \ 0 \ 1 \ 0]$$


**Figure 9.5.** *The R-pentomino.*

As the simulation proceeds, the population throws off a few gliders, but otherwise remains bounded. If you make `b` large enough, the statement

```
lifex(R,b)
```

shows all of the bounded behavior. How large does this `b` have to be? What is the maximum population during the evolution? How many gliders are produced? How many steps does it take for the population to stabilize? How many blinkers are present in the stabilized population? What is size of the stabilized population?

9.6 *Exection time*. Display actual computer execution time by adding `tic` and `toc` to `lifex.m`. Place the single statement

```
tic
```

before the start of the inner loop. Change the call of `caption` to

```
caption(t,nnz(X),toc)
```

Make the appropriate modifications in the `caption` subfunction at the end of `lifex.m`. Demonstrate your modified program on a few interesting examples.

9.7 *The Ark*. Run

```
lifex('ark',128)
```



for a few minutes. About how much time does it take on your computer to do one step? According to the Lexicon, the `ark` requires 736692 steps to stabilize. About how much time will it take on your computer for the `ark` to stabilize?

9.8 *Houses*. Check out

```
lifex(houses)
```

9.9 *Checkerboards*. This code generates an  $n$ -by- $n$  checkerboard of 0's and 1's.

```
[I,J] = meshgrid(1:n);  
C = (mod(I+J,2)==0);
```

What are the stabilization times and final populations for  $n$ -by- $n$  checkerboard initial populations with  $3 \leq n \leq 30$ ?