MAS336 Computational Problem Solving

Problem 5: Conway's "Game of Life"

© Francis J. Wright, 2007

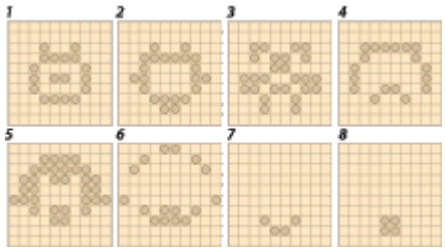
Topics: cellular automata, sparse arrays, loops, plotting, animation

Introduction

This introduction, including the illustrations, is taken almost verbatim from two articles in *Scientific American* [1,2].

John Horton Conway was born in Liverpool in 1937. He began his mathematical career at the University of Cambridge and in the mid-1980s moved to Princeton. At Cambridge, Conway was exploring the idea of the universal constructor, which is a hypothetical machine that could build copies of itself and was first studied by American mathematician John von Neumann in the 1940s. Von Neumann created a mathematical model for such a machine based on a Cartesian grid. Conway simplified the model, and it became the now famous "Game of Life", which is a game in the sense of solitaire, i.e. a single-player game. A great deal of information has been published about the Game of Life, much of it on the world-wide web.

In the game, you start with a pattern of "markers" on the grid – these represent the "live" cells. You then remove each marker that has one or no neighbouring markers or four or more neighbours (these cells "die" from loneliness or overcrowding). Markers with two or three neighbours remain on the grid. In addition, new cells are "born" – a marker is added to each empty cell that is adjacent to exactly three markers. By applying these rules repeatedly, one can create an amazing variety of forms, including "gliders" and "spaceships" that steadily move across the grid. The following is an amusing stationary pattern:



Cheshire Cat, a Game of Life pattern that transforms into a grin (7) and finally a paw print (8)

Conway showed the game of Life to his friend Martin Gardner, the longtime author of *Scientific American's* Mathematical Games column. Gardner described the game in detail in his October 1970 column [2], and it was an immediate hit. Computer buffs wrote programs allowing them to create ever more complex Life forms and there are several programs in various languages (including both Java and JavaScript) freely available on the web (which is why this problem may be solved using only Maple). In particular, there is a nice Windows version [3] that I have found useful. "The game made Conway instantly famous," Gardner comments. "But it also opened up a whole new field of mathematical research, the field of cellular automata." This field was then pioneered by Stephen Wolfram, who also created Mathematica, which is a direct competitor to Maple. One way to implement the Game of Life is with counters such as drafts pieces on a large chess board, although we are interested in implementing it as a Maple program.

The basic idea is to start with a simple configuration of counters (organisms), one to a cell, then observe how it changes as you apply Conway's "genetic laws" for births, deaths, and survivals. Conway chose his rules carefully, after a long period of experimentation, to meet three desiderata:

- There should be no initial pattern for which there is a simple proof that the population can grow without limit.
- There should be initial patterns that *apparently* do grow without limit.
- There should be simple initial patterns that grow and change for a considerable period of time before coming to end in three possible ways: fading away completely (from overcrowding or becoming too sparse), settling into a stable configuration that remains unchanged thereafter, or entering an oscillating phase in which they repeat an endless cycle of two or more periods.

In brief, the rules should be such as to make the behaviour of the population unpredictable.

Conway's genetic laws are delightfully simple. First note that each cell of the grid (assumed to be an infinite plane) has eight neighbouring cells, four adjacent orthogonally, four adjacent diagonally. The rules are:

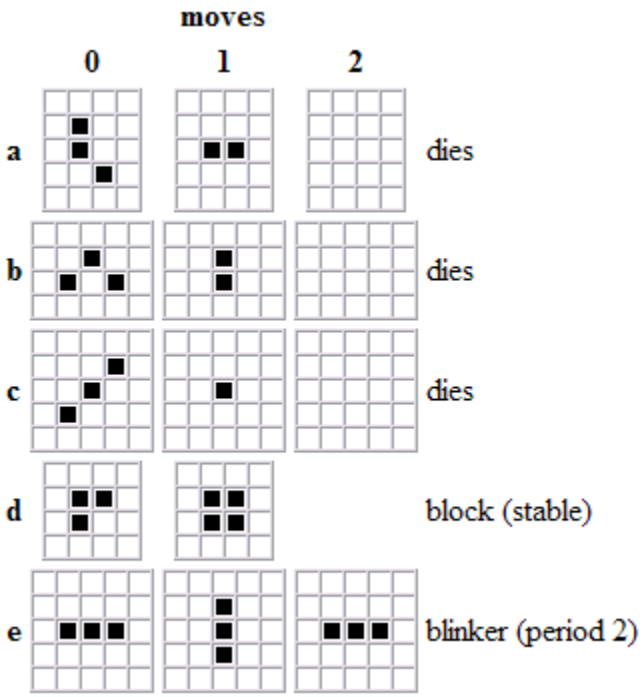
- **Survivals**. Every counter with two or three neighboring counters survives for the next generation.
- **Deaths**. Each counter with four or more neighbors dies (is removed) from overpopulation. Every counter with one neighbor or none dies from isolation.
- **Births**. Each empty cell adjacent to exactly three neighbors – no more, no fewer – is a birth cell. A counter is placed on it at the next move.

It is important to understand that all births and deaths occur *simultaneously*. Together they constitute a single generation or "move" in the complete "life history" of the initial configuration.

You will find the population constantly undergoing unusual, sometimes beautiful and always unexpected change. In a few cases the society eventually dies out (all counters vanishing), although this may not happen until after a great many generations. Most starting patterns either reach stable figures – Conway calls them "still lifes" – that cannot change or patterns that oscillate forever. Patterns with no initial symmetry tend to become symmetrical. Once this happens the symmetry cannot be lost, although it may increase in richness.

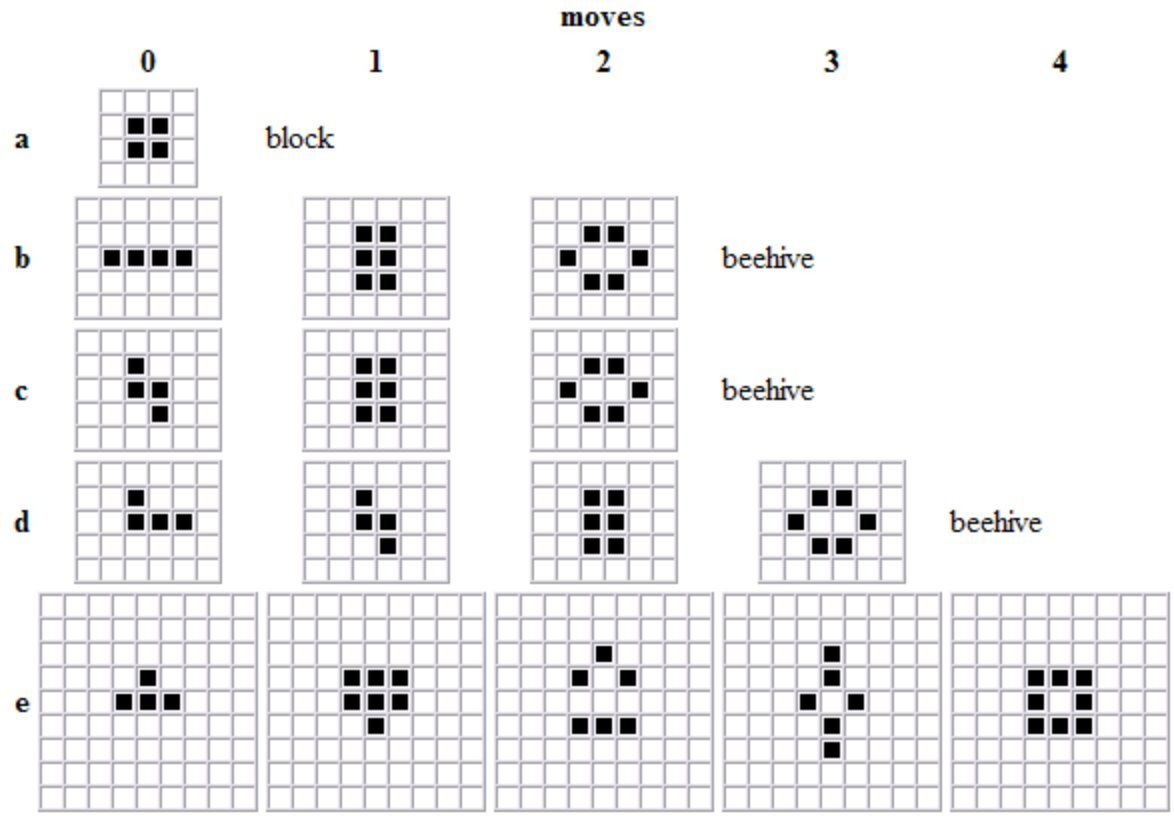
Conway conjectures that no pattern can grow without limit. Put another way, any configuration with a finite number of counters cannot grow beyond a finite upper limit to the number of counters on the field. This is probably the deepest and most difficult question posed by the game. One way to disprove it would be to discover patterns that keep adding counters to the field: a "gun" (a configuration that repeatedly shoots out moving objects such as the "glider"; see [2]) or a "puffer train" (a configuration that moves but leaves behind a trail of "smoke").

Let us see what happens to a variety of simple patterns. A single organism or any pair of counters, wherever placed, will obviously vanish on the first move. A beginning pattern of three counters also dies immediately unless at least one counter has two neighbours. The following illustration



shows the five triplets that do not fade on the first move. (Their orientation is of course irrelevant.) The first three [*a*, *b*, *c*] vanish on the second move. In connection with *c* it is worth noting that a single diagonal chain of counters, however long, loses its end counters on each move until the chain finally disappears. The speed a chess king moves in any direction is called by Conway the "speed of light." We say, therefore, that a diagonal chain decays at each end with the speed of light. Pattern *d* becomes a stable "block" (two-by-two square) on the second move. Pattern *e* is the simplest of what are called "flip-flops" (oscillating figures of period 2). It alternates between horizontal and vertical rows of three. Conway calls it a "blinker".

The following illustration



shows the life histories of the five tetrominoes (four rookwise-connected counters). The square [*a*] is, as we have seen, a still-life figure. Tetrominoes *b* and *c* reach a stable figure, called a "beehive," on the second move. Beehives are frequently produced patterns. Tetromino *d* becomes a beehive on the third move. Tetromino *e* is the most interesting of the lot. After nine moves it becomes four isolated blinkers, a flip-flop called "traffic lights." It too is a common configuration.

Task: Animate the Game of Life

Animate ten steps of the Cheshire Cat pattern in the Game of Life, as illustrated above, which should stabilize after eight steps. The pattern should be (approximately) centred within a suitably sized finite grid. You may like to use a few of the smaller patterns described above as initial test cases. Here is some guidance on possible Maple implementations, but you can use a completely different implementation if you wish (provided it is in Maple).

The Game of Life assumes an infinite Cartesian grid, which is impossible to implement in practice: the infinite grid must be represented as a finite rectangular grid. There are various ways to do this. One is to use periodic boundary conditions, which means that the finite grid is considered to be wrapped around so that opposite edges are joined to give first a cylinder and then a torus. This means that when a path falls off one edge of the finite grid it reappears at the opposite edge. Another is to regard the finite grid as a window onto the infinite grid, so that when a path falls off one edge of the finite grid it simply disappears. Not all of the implementations that I have looked at appear to behave in a very well defined way at the edges of the finite grid; hence, your solution to this problem is not required to show well defined boundary behaviour, although this is an issue that you should certainly think about. It is probably best to make the size of the visible grid a parameter represented by a global variable, the value of which can be either a single number if the grid can only be square or a pair of numbers if the grid can be non-square. It is sufficient for this problem to use a square grid; allowing a non-square rectangular grid is optional.

What is essential is to be able to consider all the neighbours of a cell that is on the boundary of the visible grid without causing an error. Only nearest neighbours need be considered in the Game of Life, therefore one easy way to avoid boundary errors is to use a "guard ring" of hidden cells around the visible grid, i.e. an additional single column either side and an additional single row above and below that are always empty. This is different from either of the boundary conditions discussed previously. Its advantage is that it is easy and efficient to implement; its disadvantage is that it violates the rules of the game at the boundaries of the visible grid. I suggest that you use this boundary condition first and then consider better-defined boundary conditions later if you have time. The boundary condition should not affect the Cheshire Cat pattern, or any other stable pattern that is small enough to be contained within the visible grid with at least one empty cell all around, but if does affect any pattern that moves across the grid.

A grid cell is considered to be "alive" if it contains a marker and "dead" otherwise. The obvious representation for the Cartesian grid is a rectangular array and the obvious representation for the marker in a cell is a Boolean value in the corresponding array element.

However, the state of a cell in the next generation depends on the *number* of live neighbouring cells, so it may be better to represent the Boolean values by 1 and 0, which can be added, rather than true and false. Then the state of a cell in the next generation is determined by the sum of the values in its eight nearest-neighbour cells.

Each generation of the Game of Life can be represented by an array with the same specification. The first generation can be established by entering the coordinates (or indices) of the cells that should be alive by using an initialization procedure that accepts an arbitrary number of coordinate (or index) pairs. Each subsequent generation can be established by calling a procedure that takes the array representing the previous generation as an argument and returns an array representing the next generation, determined by applying the rules of the game. In this way, a sequence (or list) of a specified number of generations can be constructed. The game rules are applied to each cell of the grid by using a double loop; the fate of a cell in the next generation is determined by the sum of the number of live cells surrounding the cell in the previous generation, which can be determined by one more loop or some equivalent summation structure.

The array elements must be initialized to 0 (or false), and for most of the time most elements will probably contain the value 0. Such an array is said to be *sparse*, and Maple provides special support for sparse arrays, which avoids the need explicitly to store all the array elements that are 0; any array element that has not been assigned a value automatically evaluates to 0. This avoids the storage overhead of storing "empty" elements and also avoids the need for any explicit initialization. See the online help for further details.

A single generation of the game can be displayed as a grid using different colours to represent live and dead cells. This is similar to the display used for the Eight Queens problem and could be handled identically, except that it is probably better for the Game of Life to fill each cell completely with colour. This is because the rules governing the Game of Life are entirely local (i.e. nearest neighbour) and each distinct collection of live cells can be usefully regarded as an individual organism, which looks more convincing if the live cells merge to some extent. The Maple procedure `plots[polygonplot]` can be used to plot polygons filled with a specified colour and a sequence of plots of single generations can be animated by using `plots[display]` as in Computational Mathematics II; see [4] and/or the online help for further details.

References

[1] Mark Alpert, NOT JUST FUN AND GAMES, *Scientific American* (17 April 1999). Available via the web at http://www.sciam.com/.

[2] Martin Gardner, Mathematical Games: The fantastic combinations of John Conway's new solitaire game "life", *Scientific American* (October 1970). Available in part via the web at http://ddi.cs.uni-potsdam.de/HyFISCH/Produzieren/lis_projekt/proj_gamelife/ConwayScientificAmerican.htm, a copy of which is also available locally via the course web site.

[3] Edwin Martin (2001). Available via the web at http://www.bitstorm.org/gameoflife/, a copy of which is also available locally via the course web site.

[4] Francis Wright, *Computing with Maple*, Chapman & Hall/CRC (2001).

Mark scheme

- Initialization and propagation of generations of the game: 4 marks
- Display of a single generation: 4 marks
- Animation of the Cheshire Cat pattern: 4 marks
- Presentation and discussion: 8 marks