CS 7A - Fall 2015 - John Conway's Game of Life. Due 9/28/15

Background Theory

John Horton Conway is a distinguished mathematician at the University of Cambridge, who, in addition to very serious work, also enjoys recreational mathematics. Although he is highly productive in this field, he seldom publishes his discoveries. One exception is his most famous brain-child, a fantastic zero-player pastime he calls "The Game of Life," perhaps because of its similarities with the dynamics of a society of living organisms, it belongs to a growing class of what are called "cellular automata"-processes with fairly simple rules whose dynamics simulate real-life processes. To play Life without a computer you need a fairly large checker-board, or, say, an Oriental "go" board with flat counters small enough to fit within its cells. Or you can work with pencil and graph paper but it is much easier, particularly for beginners, to use counters and a board.

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:

1. There should be no initial pattern for which there is a simple proof that the population can grow without limit.
2. There should be initial patterns that apparently do grow without limit.
3. There should be simple initial patterns that grow and change for a considerable period of time before coming to an end in three possible ways: (i) Dying off completely (from overcrowding or from sparsity), (ii) approaching a stable configuration that doesn't change, or (iii) oscillating in repetition of an endless cycle of two or more states.

In brief, the rules should interesting and unpredictable population dynamics. Conway's rules are very simple. First note that each interior cell of the checkerboard (either an infinite plane, or a bounded rectangle) has eight neighboring cells, four adjacent, four diagonal. The rules are:

1. **Survivals**. Every counter with two or three neighboring counters survives for the next generation.
2. **Deaths**. Each counter with four or more neighbors dies (is removed) from overcrowding. Every counter with one neighbor or none dies from isolation.
3. **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.

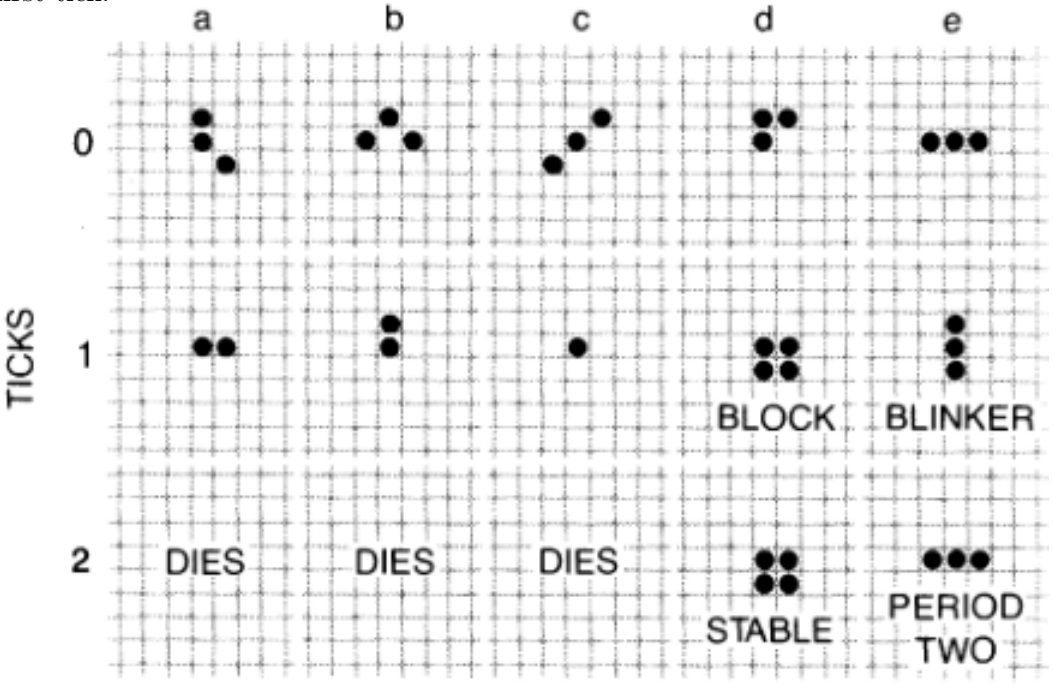
All births and deaths occur simultaneously, and together constitute a single generation or, as we shall may call it, a "tick" in the complete "life history" of the initial configuration. Conway recommends the following procedure for making the moves:

1. Start with a pattern consisting of black counters.
2. Locate all counters that will die. Identify them by putting a black counter on top of each.
3. Locate all vacant cells where births will occur. Put a white counter on each birth cell.
4. After the pattern has been double-checked to make sure no mistakes have been made, remove all the dead counters (piles of two) and replace all newborn white organisms with black counters. You will now have the first generation in the life history of your initial pattern. The same procedure is repeated to produce subsequent generations. It should be clear why counters of two colors are needed. Because births and deaths occur simultaneously, newborn counters play no role in causing other deaths or births. It is essential, therefore, to be able to distinguish them from live counters of the previous generation while you check the pattern to be sure no errors have been made. Mistakes are very easy to make, particularly when first playing the

game. After playing it for a while you will gradually make fewer mistakes, but even experienced players must exercise great care in checking every new generation before removing the dead counters and replacing newborn white counters with black.

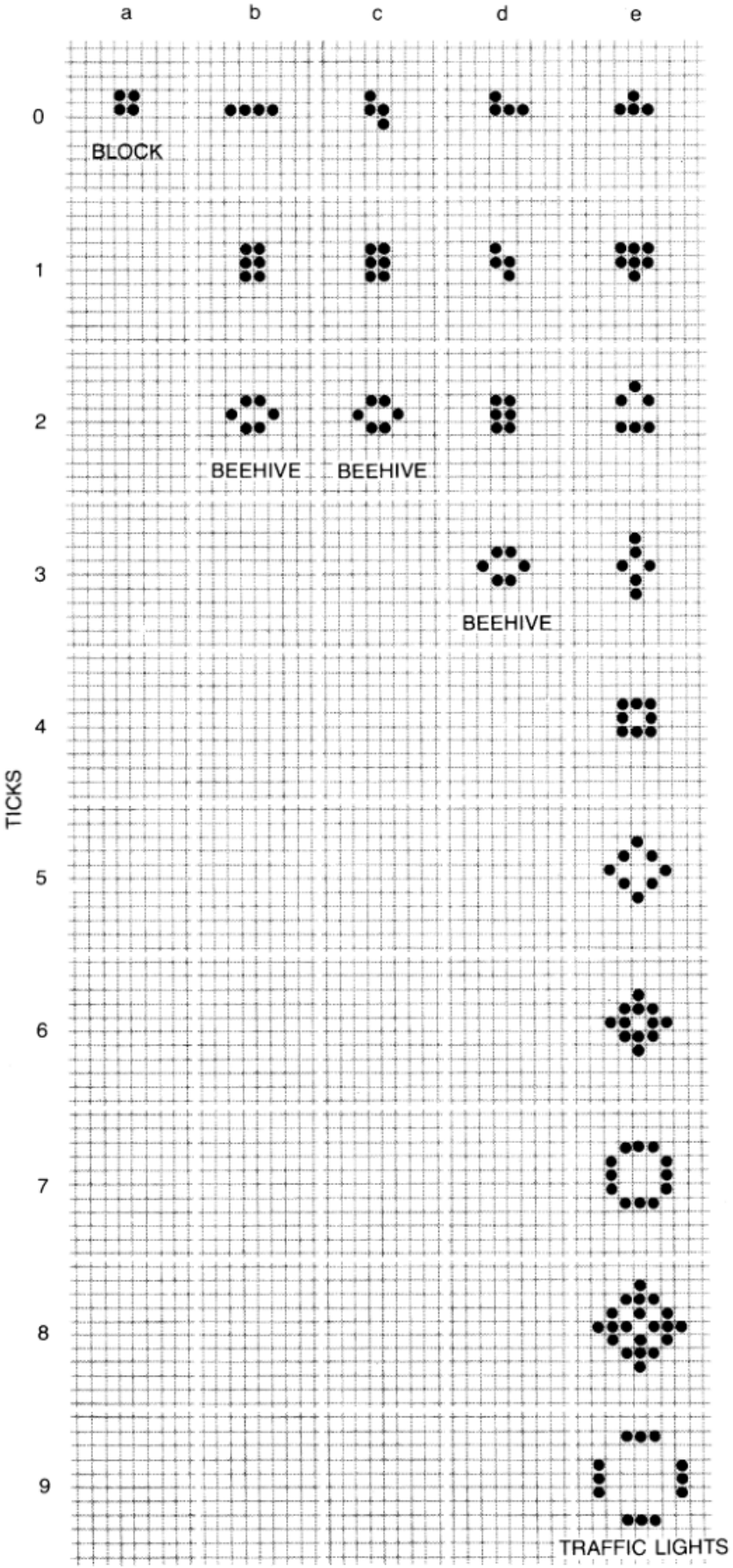
You will sometimes find the population undergoing strange, beautiful and unexpected change. The society may eventually die out (all counters vanishing), though sometimes not until after many generations. Most initial configurations either reach stable figures-Conway calls them "still 1ifes"-that cannot change or patterns that oscillate forever. Patterns with no initial symmetry tend to be come symmetrical. Once this happens the symmetry cannot be lost, although it may increase in richness. Conway originally conjectured 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. At the time this was one of the most difficult questions posed by the game. Conway offered a prize of $50 to the first person who could prove or disprove the conjecture before the end of 1970. 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," to be explained below) 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 tick. A beginning pattern of three counters also dies immediately unless at least one counter has two neighbors. The figure below shows the five connected triplets that do not fade on the first tick.



The fate of five triplets in "life"

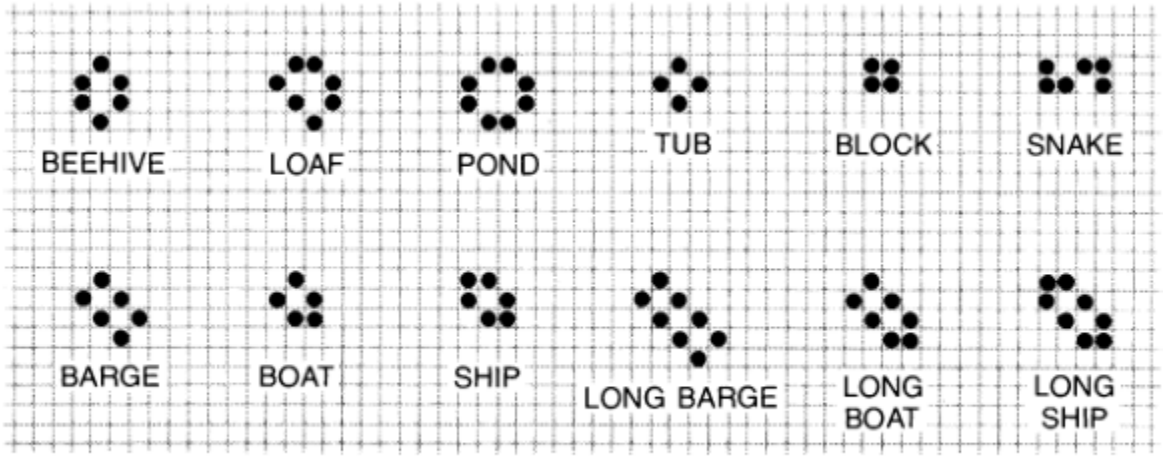
(Their orientation is of course irrelevant.) The first three [a, b, c] vanish on the second tick. In connection with c it is worth noting that a single diagonal chain of counters, however long, loses its end counters on each tick until the chain finally disappears. The speed a chess king moves in any direction is called by Conway (for reasons to be made clear later) 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 tick. Pattern e is the simplest of what are called "blinkers" (oscillating figures of period 2). It alternates between horizontal and vertical rows of three.



The life histories of the five tetrominoes

The figure at left 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 tick. Beehives are frequently produced patterns. Tetromino d becomes a beehive on the third tick. Tetromino e is the most interesting of the lot. After nine ticks it becomes four isolated blinkers, a blinker called "traffic lights." It too is a common configuration.

The next figure shows 12 common forms of still life. You may enjoy experimenting with the 12 pentominoes (all possible patterns of five rookwise-connected counters) to see what happens to each. You will find that five vanish before the fifth tick, two quickly reach a stable loaf, and four in



The commonest stable forms

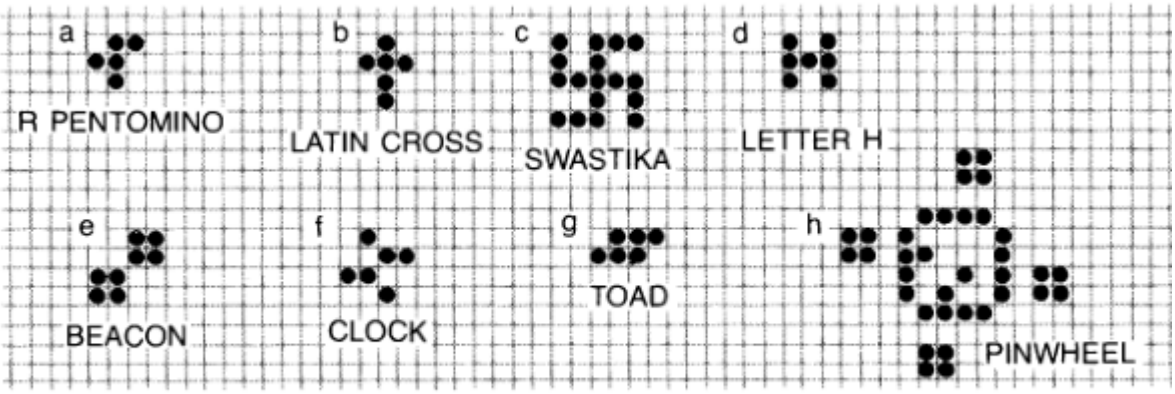
a short time become traffic lights. The only pentomino that does not end quickly (by vanishing, becoming stable or oscillating) is the R pentomino ["a" in of exercise 1. Conway has tracked it for 460 ticks. By then it has thrown off a number of gliders. Conway remarks: "It has left a lot of miscellaneous junk stagnating around, and has only a few small active regions, so it is not at all obvious that it will continue indefinitely."

Here's some starter code for the cellular automata "Life."

```
/// Geoff Hagopian - A shot at life.

#include "..\std_lib_facilities.h"
#include <windows.h>  //for colors
#include <ctime>
// a global constant the is the edge length of a square board
const int bsize = 10;

void show_Board(vector<char> brd) {
    // write code to display the vector<char> brd as a rectangular array.
}

void setup_Board(vector<char>& brd, int rndm) {
    // write code that allows for various initial configurations including
    // random, manual (user enters cells) or standard configs
}

int neighbors(vector<char> brd, int pos) {
    // This is the main engine that counts the number of living neighbors
    // Treat the boundary with care!
}

void evolve_Board(vector<char> brd0, vector<char>& brd1) {
    //for each element of the brd0 decide if it should survive, die or be born
    //and record the result in brd1
    // Note the need to preserve the board while it's updated, and thus
    // the need for a second board
}

int main() {
    srand(unsigned(time(0))); // seed the random number generator
```

```
    vector<char> board0(bsize*bsize,'0'), board1(bsize*bsize,'0');
33  setup_Board(board0,1);
    while(1) {
35      system("cls");  // brute force clear screen
        show_Board(board0);
37      evolve_Board(board0,board1); // swap boards each time
        cin.clear();
39      cin.get(); // pause
        system("cls");
41      show_Board(board1);
        evolve_Board(board1,board0);
43      cin.get();
    }
45 }
```

1. Write code to evolve "Life" forms on a `bsize` by `bsize` square grid. In particular, write a set-up program that will allow the user to choose each of the following initial configurations (a-h) and then watch what happens.



The R pentomino *(a)* and exercises for the reader