Conway's game of life is a kind of cellular automata. Cellular automata are mathematical constructions which set up rules governing a grid of "cells" - each cell being considered alive or dead, changing state over time depending on the number of neighbors it has. The rules can vary but in the "traditional" version of Conway's Life, living cells with 2 or 3 neighbors survive, all others die, and dead cells with exactly 3 neighbors come to life. These simple rules can make for some surprisingly interesting patterns.

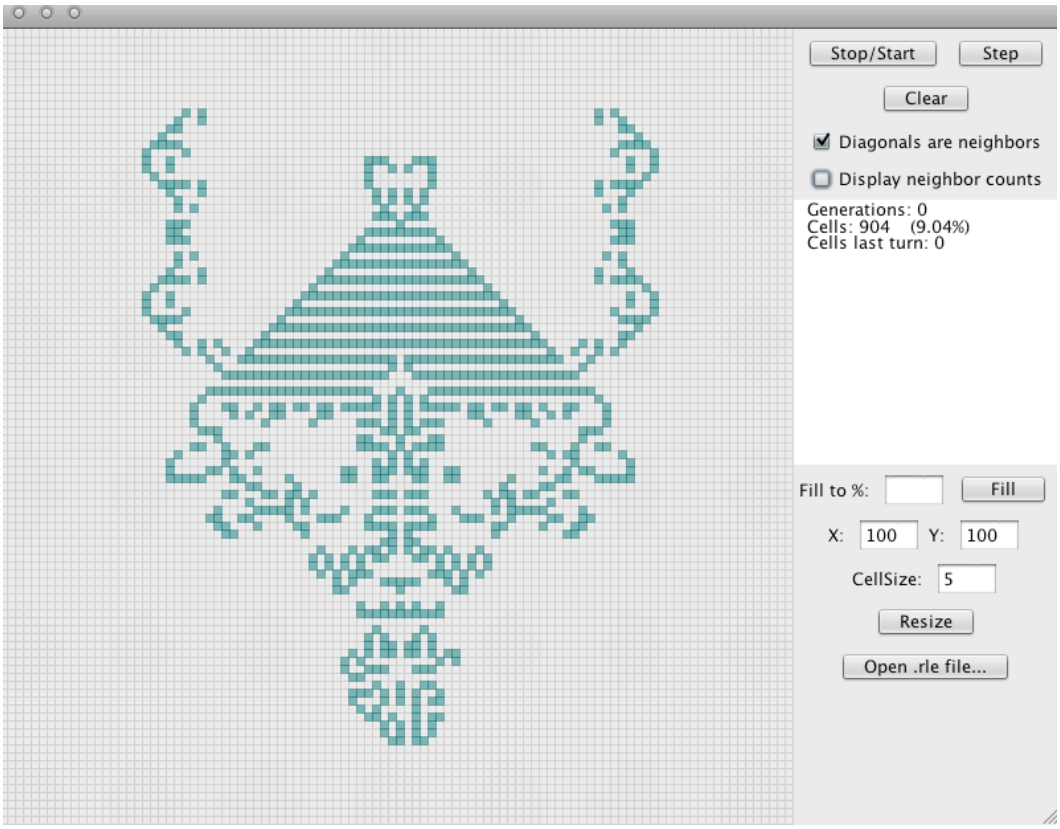

Fig. 1 Program Interface showing "Halfmax", from http://www.conwaylife.com/wiki/index.php?title=Halfmax

Program Operation: Run the program using "java gameOfLife".
Using the included implementation of the Game of Life is fairly simple. The left section of the window is the cell grid, which can be clicked on at any time to create or destroy cells. On the right are some controls, for stopping, starting and single-stepping the simulation, changing the grid and cell sizes, loading a .rle-formatted file (for loading premade automata), and randomly filling the grid. You can also toggle whether diagonal cells are counted as neighbors - this has a profound effect on the operation of the game.

Observations: Simply filling the grid with randomly placed cells can yield interesting results. As you would expect, the fewer cells the grid is filled with, the sparser the fill, so the less likely it is that any one cell will have enough neighbors to

either survive or come to life. Overcrowding (cells with 4 or more neighbors die the next turn) becomes a problem when around 75% of the cells are filled, though larger grids raise this number as there is more room to work with. In the middle ranges, most boards seem to reach an equilibrium state after somewhere between 200-500 generations. Equilibrium states usually include some combination of static "blocks" and oscillating shapes, very few initial configurations end in empty fields and none ever ended with full fields (the rules do not allow for this).

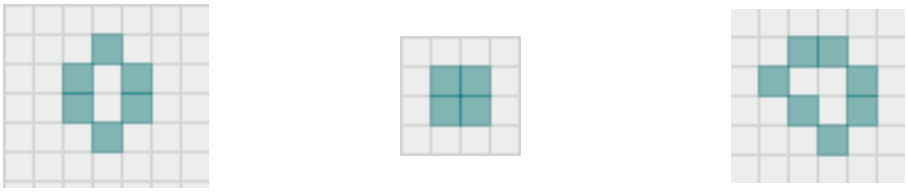

Fig. 2 Some Common Blocks

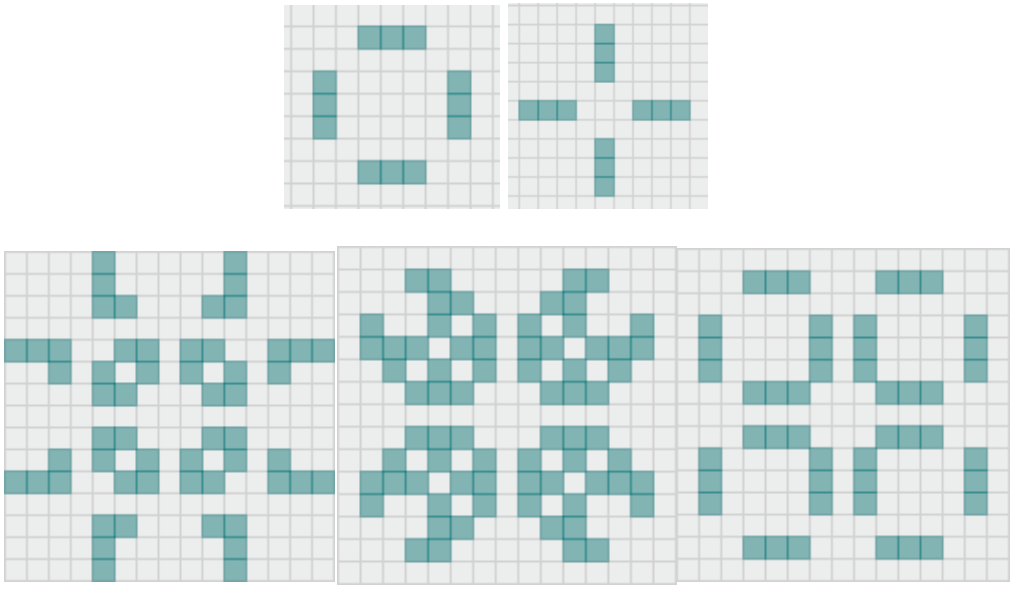

Fig. 3 Some Oscillators

The bottom row of figure 3 is the most interesting repeating figure I found while examining the randomly generated board configurations. The individual bars of the "twinkling star" (top row of fig. 3) are most often found alone, but sometimes come in groups like shown.

Fig. 4 The "Glider"

Another commonly found (even in randomly generated boards) is what Game of Life scholars have termed the "Glider". It's a set of 5 cells that moves, so to speak,

diagonally across the board in whatever direction its leading corner is facing. Of course the cells aren't actually moving, they're just arranged in a pattern that generates new cells in front and dies off in back. When a gliders hits an edge it turns into a stable block of 4 cells. Interestingly enough, if another glider is following it along the same path and collides with the block, they will eliminate each other.

Rule Variations: Built into this implementation of the Game of Life is the ability to change the rules. One can decide whether diagonal neighbors are counted, as well as the number of neighbors needed for the birth and death of cells. Changing the rules up reveals some interesting phenomena, but the numbers must be well-chosen as many combinations are unstable and lead mostly to noisy, chaotic boards.

Variation a: No diagonals, 3 to generate, 2 or 3 to live.
This is the standard Life ruleset, just not counting diagonals as neighbors. Because of this all cells have fewer neighbors so all cells/patterns die off faster. The cells seem to like to congregate in double lines or in blocks with holes in the center. It does have a 2-step oscilatting pattern that appears occasionally in randomly generated boards (checkerboard).

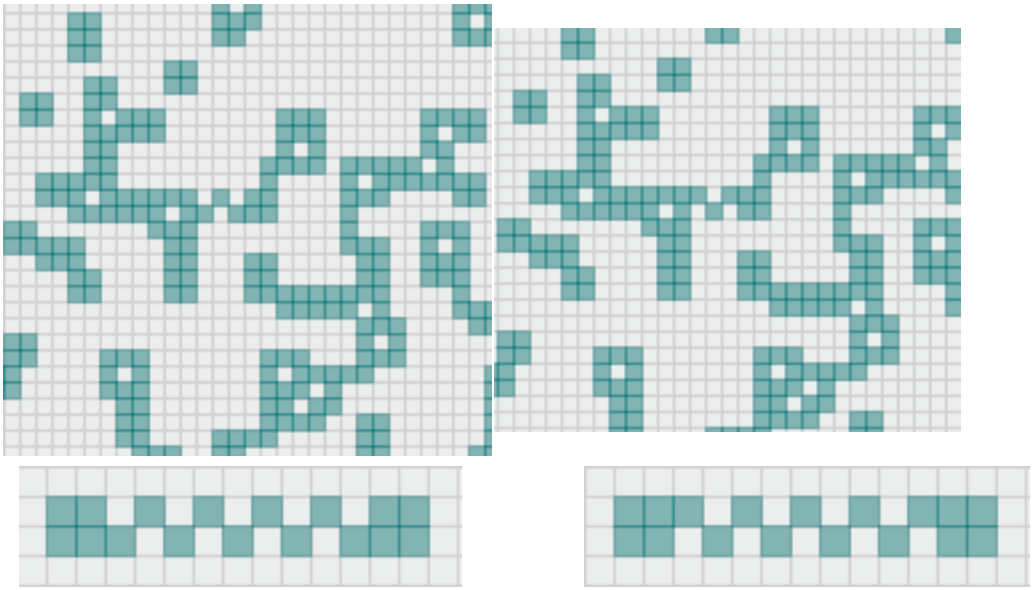

Fig. 4 Checkerboards

Checkerboards can be made by hand to any length desired (bottom half of fig. 4).

Variation b: No diagonals, 2 to generate, 2 to live.
This ruleset exhibits many different repeating patterns in randomly generated boards, mostly diagonal repeaters (diagonals are not counted as neighbors but a cell adjacent to two other cells will come to life while those cells will likely die).

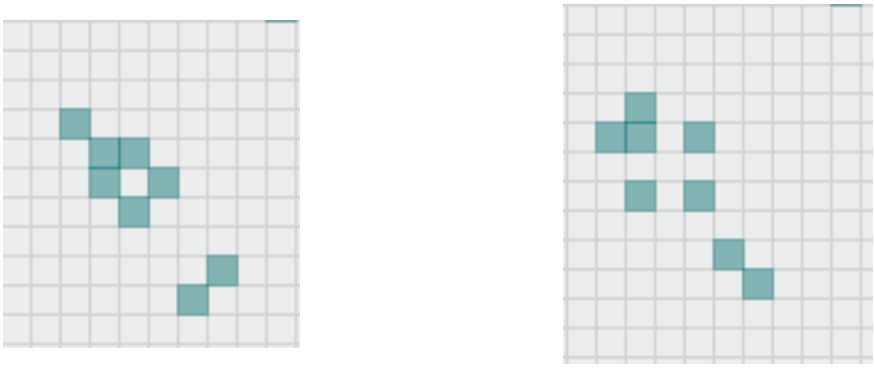

Fig. 5 Diagonal Repetition

Program Implementation Notes:

This program is written in Java and uses the Java "Swing" components to have a (fairly) platform-independent GUI, at the cost of some efficiency. Nevertheless it does well even with large cell grids due to only living cells and their immediate neighbors being processed when the grid is being advanced - there is no point in wasting time and memory storing and processing nonliving cells that are not going to change in the next turn. The program will slow down once it starts having to handle large (600x600) grids that are filling up (the included "Max" and "Halfmax" patterns are good for slowing your machine to a crawl).

Additional speed is gained by referencing all cell objects in a hashtable (hashed according to their coordinates) as opposed to an array. Every "turn", the hashtable is iterated over, each cell in it compared against the rules and pushed either onto a "deathRow" stack or the "lifeRow" stack. The hashtable contains not only living cell objects - every time a new cell is brought to life, dormant cells are initialized around it, with the correct neighbor counts so that next turn they are ready to be brought to full life if necessary.

This program is also able to open automata files in .rle format. These are just plain text files in "run length" encoding, and are commonly found on the internet as downloadable examples. Many interesting automata forms can be found on http://www.conwaylife.com/wiki/.