Conway’s Game of Life

Evan Nichols

NPRG063 Winter 2020/21

Main Idea

Conway’s Game of Life is a zero-player automaton that consists of a grid and its cells, which are either ”alive” or ”dead” (filled or not filled). John Horton Conway, the British mathematician, developed the Game of Life in 1970. This program will present the user with a simple but robust interface that presents a board for the Game of Life and various controls for the user to manipulate the state of the automaton. The user will be able to place some standard patterns, adjust the speed and size of the game, and choose from several known starting patterns that have already been discovered.

The game runs indefinitely, advancing from state to state. Each cell’s status (filled/alive or empty/dead) in the next state is determined by its current status and these four rules:

- Any live cell with fewer than two live neighbours dies, as if by underpopulation.
- Any live cell with two or three live neighbours lives on to the next generation.
- Any live cell with more than three live neighbours dies, as if by overpopulation.
- Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

... which can be consolidated into these three essential rules:

- **Any live cell with two or three live neighbours survives.**
- **Any dead cell with three live neighbours becomes a live cell.**
- **All other live cells die in the next state & all other dead cells stay dead.**

These rules were developed by Conway to ensure the following conditions for the game:

- There should be no explosive growth.
- There should exist small initial patterns with chaotic, unpredictable outcomes.
- There should be potential for von Neumann universal constructors.
- The rules should be as simple as possible, whilst adhering to the above constraints.

Implementation

This program is a Windows Forms application developed in C# using Visual Studio 2019.

Forms

Two forms are presented to the user:

- **Form1** is the main user interface. It contains the board and all control options and interactions. The main feature is the game board, on which the cells are drawn from one state to the next. The user is able to change the pause and resume the game, change its speed, change the size of the board, toggle the drawing of grid lines, and change the filled cells' colors. The current generation is tracked and displayed.

  The user is also able to begin the game with several preset patterns, randomize or clear the board, and add some fundamental patterns to the board via mouse clicks directly on the board. The current state can be saved at any time and loaded later. Lastly, there is an *Info* which summons the only other form, which contains some text about how the game works.
- **Form_Info** is a small window that contains only a short summary of the game. It is created and shown when the user presses the *Info* button on Form1.

Classes

Form1 contains all of the controls for the user interface.

First, *timer1* advances the game and counts the generation. The game is driven forward by itself when the timer is enabled.

Then, the GUI interactions are handled via click and button press functions. The shape functions are called by the radio buttons, and they simply set the 'SHAPE' string which is used by *DrawShape()*. The functions to update the board when the size is changed, the cell color is changed, grid lines are toggled, or the board is cleared or randomized.

Lastly, this class holds the functions to draw the board and also to draw the basic patterns when the user places them. There are two *Reset()* instances, one of which takes a starting pattern and applies it. To draw the board, the program scans over the board and draws a filled square at the locations of live cells.

Model.cs contains the classes for the board and cells.

Cell is a class to represent the individual cells in the board, and its properties are two bools (if it is alive, and if it will live) and a List of other adjacent cells (its neighbors).

Board consists of a 2D array of cells, a size integer, and dimension integers. It has 3 functions: to find and set the neighbors of a cell, to set the next state of the board by setting cells' alive status, and to randomize the board by setting cells' alive status randomly.

Patterns contains a dictionary of patterns and their names. Each pattern is represented as a string, with newlines delimiting each line. The function *Fetch()* takes a string and finds the pattern with the matching name. The dictionary also contains a "not found" pattern.