An Implementation of *Conway's Game of Life* in JavaScript and HTML5

By: Daniel Prilik

Note, that Conway's GOL is not to be confused with the popular Milton-Bradley board game The *Game of Life* with the same name.

Introduction

Developed by mathematician John Conway in 1970, the *Game of Life* is a cellular automation based on some very simple principles. Conway's GOL is played on either a finite or infinite two-dimensional grid. Each cell can have one of two states, on or off, or "dead" or "alive." The state of each cell in the grid is reliant on the state of the 8 cells surrounding it, and how many of the surrounding cells are "dead" or "alive." The rules are as follows:

* Any live cell with fewer than two live neighbours dies, as if caused by under-population.
* 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 overcrowding.
* Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

The GOL is a zero-player game, one where the only time a "player" is interacting with the grid is when setting up it's "seed," or initial state. Depending on the initial configuration of the cells, there can be a multitude of resulting patterns, ranging from stationary cell formations, to blinking cell formations, yet getting complex enough to glide across the screen. Furthermore, mathematicians and computer scientists have proven that the GOL is a Turing-complete simulation, and some intrepid GOL players have actually created basic ALU and CPU's within the simulation.

Technical Details

I chose to implement the GOL in JavaScript and HTML5, thereby allowing the application to be run on a multitude of browsers across many platforms. The grid itself is a simple two-dimensional array, one where each entry in an array is in-itself an array containing each grid-cell. Looping through the array is done through a for-loop looping through the y-columns, and a nested for loop, looping through the x-columns. As each cell is looped through, a count is started detailing what is the state of the surrounding cells. Once the state of the cells around the current cell is determined, the cell state is updated on a new grid, as to update the cell right on the main grid would result in skewed results when determining the state of the cell next-to-itself. After processing all the cells, the new array overwrites the regular grid, and then the rendering engine kicks in.

Rendering a grid over a canvas is fairly simple. Using a similar loop-system as before, the rendering engine gets the state of each cell, it's position in the grid, and then renders a square of variable size in the appropriate place on the canvas.

Lastly, input is handled by a custom library I've written over the course of a couple projects. It handles cursor input across mouse and touch, and then returns cursor position in a simple Cursor object that can be referenced at will. Also, a helper function was implemented to get grid-coordinates from the cursor's position using the ceil-function. Buttons are handled with basic jQuery, stylized by jQuery mobile's UI.

Saving and Loading consists of exporting the grid as a JSON object using JavaScript's built in JSON.stringify() function, and done is similarly by overwriting the grid with a JSON object parsed with JSON.parse() function.

Use Cases

It's entertaining to see how a seed can evolve beyond itself and make cool patterns. Other than that, the GOL really has no practical application.

Developer Notes

Throughout the year I worked on many projects that involved JavaScript and HTML5 canvas. I've grown quite fond of how simple it is to create great looking web-apps with the simplicity and resilience that JavaScript offers. Nevertheless, it's always annoyed me that JavaScript is really slow. Like REALLY slow. It's not at all multithreaded, and it's a very messy language to begin with, with no coherent theme throughout its syntax. Because of these drawbacks, I was forced to abandon some promising projects. In my Physics Thingy projects that I worked on in conjunction with Isaac Milach, we realized that implementing multi-body collision would require multithreaded processing, or else there would be massive frame drops. JavaScript uses a single thread for both rendering and computing, therefore if computation or rendering begins to lag, the entire app is slowed down doubly.

Anyhoo, the GOL is a nice, simple, not too computationally intensive application, and it was quite interesting to solve a couple array-referencing issues. All in all, I think it is a fitting end to a good year, and a nice call-back to my first major project of the year, where I did a simpler cellular-automation.