6.170 Project 1 Phase 1 Design Document
Michael Wee – September 15, 2013

GitHub link:

The first design challenge was to conceive of how to implement the Game of Life rules and functionality itself and how to bridge the rules of the game and the iterative graphical result shown to end users. I decided to abstract away the game logic away from the graphic logic to make the code more modular and abstracted. This decision gives me the freedom to change graphic libraries on a whim or change implementations or data structures while preserving the look of the graphics board.

In choosing how to implement the Game of Life logic, I deduced that I needed to keep track of the state of the board and have a way of updating it. I had to create an object that held the state of the board and had two main options of doing it: creating an object with the new and this keywords or holding state in a closure that is returned. I decided to return a closure because of the nice properties it has with keeping the namespace clean and avoiding the inconveniences that come with using new/this.

The decisions within the main driver of the program included using the idiomatic Javascript main function that executes a function to keep the namespace enclosed. I call the SetInterval method to update the board in steps within the driver loop and used an anonymous function to first draw the grid, update the graphics based on the Board object, and then update the Game of Life on step further. The only other decision I considered here was doing a quick refresh of the page per step of the game, but that would require state be held beyond the life of a single page, for example, in a cookie or a database (I considered using Firebase quickly), but this is actually an unnecessary that SetInterval resolves elegantly.

In terms of naming conventions, I named each variable and function to be as descriptive as it could be in the context of what state it held or what functionality it performed within the game. I followed the capitalization convention outlined in the provided graphics library code, keeping variable names (other than Objects) lower case, and separating whole words with underscores. I abstracted away function calls in modular components to make understanding better for the code, and for ease of unit testing and ensuring correctness of individual methods. By doing this it makes individual methods easier to test and it straightforward to reason about the correct logic within the interplay of these methods. For example, the draw_grid method was put in a separate function as the update_graphics method which updated the graphical board from the state. Though both methods work on the graphical window, they perform different functions and this separation helps the sanity of the programmer and understanding of the code.

The main Game of Life logic was implemented primarily with for loop logic. This was chosen as the rules were applied for each cell on the board, which two for loops iterate over well. For loops were chosen rather than iterative abstractions such as the functional from_to shown in class for programmer understanding and readability, as this programmer was long accustomed to programming with for loops, as will many other programmers who might read or work with this code. I made sure to minimize the scope of all relevant variables by preceding then with the 'var' keyword.

To finish off the logic of the Game of Life, I solved the problem of making sure each cell's logic was dependent on the current state by keeping another board that held the state of the next iteration, rather than changing the values on the board on the fly, which would affect the result. Rather than moving around pointers by reassigning variables, which would cause problems in the update, I kept a wholly separate next_gen board and copied values over to the board array one by one in order to ensure correctness. I finished off the logic with if-else-if logic predicated on the cell in question's status as well as the number of live neighbours it has.

For the initial configurations, I had to come up with a way to display several different initial configurations. One straightforward way to do this was to create multiple static boards with initial configurations, but I came across a rather more clever way (that I borrowed from Law Smith) of initializing the board to the current time.

In terms of testing, I set up other initial configurations that are standard Game of Life patterns to test the logic of the board. These also verify the proper positioning of cells on the board. I included one with still lifes such as the block, beehive, and loaf. I also included the oscillating beacon and glider. These tests can be activated by selecting the corresponding initialize board function in the Board class. These tests are the most important as they are a way of verifying behaviour in the game logic that we know and can clearly define. These tests also verify very important parts of the game: that cells are correctly initialized and that the mapping between the board array and the graphics board are in sync. For this relatively simple system, it makes sure the few moving parts work as desired and are in sync: the game logic, the graphics engine, and the interface between the game representation and the graphics engine. The other necessary inspections are all observed graphically such as the proper location of the grid lines, the pad, and the visual style.

One problem I had was aligning the results of the graphical user interface to look good on the given grid. A design decision I made in terms of the user-facing design of the board was to veer away from the green circles and red rectangles toward a cleaner, more minimalist black-and-white square grid. This and made obvious that a cell coloured black was alive, and a cell coloured white was dead. This user interface decision made the progression of the game of life cleaner, and made the patterns that emerged simpler to perceive and observe. I did away with the black box circling the grid and created my own grid with clean lines for aesthetic appeal. I gave the black boxes a white padding within the grid in for its nice visual effect.

Overall, I am pleased with the design of this project. The code is modular, with few moving pieces, attractive, and straightforward to read. I am proud of my visual overhaul of the graphics engine to a simple aesthetic and the drawing of the time as the initial configuration is a neat feature that polishes off this project phase.