Assignment 1 Design Document

There were a lot of things going through my mind when it came to designing the program required for this assignment. Of course, the first thing that I did was to break up the entire game into smaller pieces and began there. I knew that if I can get an oscillating 3x3 example, similar to the one shown in the week one overview, I would be able to expand the remaining aspects of the program in order to fit the needs presented in the game. From there the oscillator could be repositioned to generate the other two designs. As a result that was my initial thought on creating the assignments.

With lab 1 fresh in my mind I proceed with the basic portions of the assignment; the initialization and printing of the “world.” There’s not much to talk about beside the fact that lab 1 was a good place to refresh my memory with multi-dimensional arrays. From there I contemplated on how to implement the different starting designs. In retrospect his was my first mistake. There was no reason to start with the designs when the rules of the world were not set up correctly. This would mean that there was no means to alter the design as the generations progressed. After being frustrated with the designs not properly fitting the limited 3x3 world, I quickly laid out several lines of code, listing each element as a single status, so that I could manual mark each element as dead or alive. This proved useful since I was then able to control my tests later.

I then worked on creating a reference image so that I could quickly look at it when writing the “if” statements to check on the cell status. It was then and there that I hit the first snag of the assignment. I was constantly breaking my code by asking it to read outside of the small 3x3 world. For example cell (0, 0) in the top left corner would only have a maximum of three neighbors since the other five positions didn’t exist. The solution was to add an additional row and column to both the beginning and the end of the world. This solution was hinted at as a solution to the fake infinite world problem. Although this would prove insufficient as I progressed further into developing the code, I was finally able to gather the correct number of cell neighbors when I manipulated each element manually. As a result I was able to place the oscillator on any cell without any issues.

Next I proceeded to work on implementing the 4 rules found in the world. Feeling confident I manually made an oscillator shape and ran my code. And like all success stories, it started off with a huge failure. Each element was reporting the correct amount of neighbors (because I was printing the number of neighbors for each cell out to test). However with the introduction of the 4 rules, my neighbor counts would continue to grow as dead cells started to come to life. The issues which I later solved, was the fact that each generation of cells were not being updated instantaneously. My initial solution to loop through each element and check the rules against them actually hindered the surrounding cells and therefore gave me the incorrect neighbor counts. While, I would like to say that I was able to figure out the solution to this issue by myself without any assistance whatsoever, I did turn to several web searches to figure out that I should be using a second array as storage and swapping them out via copying. With this implemented the goal of changing the world generation instantaneously was completed

After finally getting my program to successful create an oscillating cell design, I then moved onto working on the glider and glider gun. Creating a basic menu system with a simple number validation system, I was then able to create the designs needed the long manual list I had constructed earlier to manually control the status of each cell. I then began expanding my world to the 40x20 size requirement as well as providing the single cell of rows and columns to offset the unseen area. I soon found that my single extra row and column was not enough to contain the glider and glider gun. Elements that reached the end of my finite world would simply not disappear. Instead my free roaming gliders and gliders made from my gun would form 2x2 self-sustaining groups. In order to make this 2x2 group to not appear to the user and preventing it from affecting live visible cells, I did 2 things. The first was to expand the length and width of the world. The glider gun in particular would not fit properly if I placed it on the top or bottom rows and columns. By ensuring that I had extra space on each end I didn’t have to worry about having any unwanted visible cells. The second thing that I did was mark all of the other most rows and columns as dead. This would ensure that gliders that fell off the edge would die out on their own. This of course was done without the user’s knowledge.

The last thing to do when all is said and done is to run the program at the four corners of the visible world. These four extreme tests would give me more information that simply running several tests in the middle of the visible world. With everything working out correctly, the assignment is done.