CMPT 470 Assignment 2

Name : Adithya Mattappily
NSID: adm115
Student number: 11289810
Class: CMPT 470

As far as variable names, calling sequence and order of arguments goes, I did not find much difficulty in recollecting them as I made sure to make variable names and arguments well defined and appropriately named.

However, one of the aspects that I forgot as to how it worked was how the program calculated which cells will be dead or alive for the next generation and how it was implemented. After reading my code I realized that I had implemented a method where two 2d-arrays of the same size were used. The first array would contain the cell states of the current generation, and another array will be used to write the cell states for the next generation. The function will then output the updated array.

I had implemented a loop which runs 100 times to cycle the generations and it would run the automaton for 100 generations. This default value of a 100 was specialized and it obstructed software flexibility. It was also brittle since the function was returning "grid A" regardless of how many generations were run. If we were to run this automaton for an odd number of generations, we would thus get the wrong value/cell states. This makes evolution, as well as testing very difficult. How I've employed changes to tackle this is to make the function return the correctly updated grid depending on whether the number of generations is an odd or even number. This makes it more flexible as now we have the ability to make the program run for whatever number of generations and it will return the correct value.

Another thing that I realized was that the code readability was not as good as I thought it was. While writing the program, I imagined that with better functional decomposition and very less duplicate code, the readability would be good. But some parts of the set_grid_values() function was difficult for me to read as it was very messy and contained duplicated messy code. I tackled this by adding more functional decomposition and making the messy code decomposed into separate functions. This decreased code duplication and immediately improved readability. However, the biggest plus point about this change was how easy it made changing the GoL automaton to the Brian's Brain automaton. It only took adding another function for the refractory state and implementing slight changes to the set_grid_values() function. Whereas this would've taken more effort if I haven't made the functional decomposition. It also would've made the messy code even messier and thus adding more complexity and reducing readability.

To summarize, The 2 big assumptions that this broke were how I assumed my code was very readable and how it could handle any number of generations without error. But by implementing the changes mentioned above, I was able to fix that. The time I took to employ these changes were well worth it because as these changes have been implemented, any further evolution of the software would take much less time. Thus reducing the total cost for maintenance and evolution. The best case scenario was to employ these changes in assignment 1, which would've reduced the cost even more.