# Assignment 1 Project Report for Team 24

Ross Mitchell (170008410), Christopher Carr (170010924) & Nicole Orr (1700105941)

# Our Approach

We decided to break the problem up into 3 main sections initially, so each group member could have part of the problem to solve. Then we amalgamated our code and repeated the process for the extensions until we had a satisfactory project.

# Difficulties encountered

There were a few issues that we encountered in the process of completing the assignment. First we found that our code needed to be grouped together while we were working on it and this was resolved by using GitHub as a version control system, so all group members had access to all the code that they needed. Secondly, we encountered an issue where our initial generation input from the user would not convert properly into binary for the first generation, this was resolved as we found that string, int and char conversion was not necessary for our input, as the input could be directly put into an array straight off the bat. We also encountered some issues coding in Visual Studio where the IDE would throw some “external errors” that could not be fixed promptly, hence why the final solution was mostly developed, compiled and tested using Sublime Text, g++ and the Linux terminal.

# Changes we would make

Given more time, we would have created a full GUI so the program was easier to use and more visually appealing. An initial step in this would be to enable the console output of the automaton to be in more visually distinct characters, such as filled and hollow squares so there is an equal length and height of generations and a more uniform feel for the output. We also could have made some of the further extensions work with the solution, such as the Game of Life extension. Given more time, we would also have liked to have researched potential real-life uses of a cellular automaton and the ways in which we could have implemented these into our code for a more realistic feel to the final solution.

# Description of the final solution

The final solution incorporates the following features the ability to:

* A basic menu interface to choose between a normal run of the automaton and a run which would be saved to a file
* Enter a custom rule/beginning generation for the cellular automaton to begin generation from
* This is then converted into a binary array for the generation to begin
* Generation loops to create new lines based off the ruleset and the line above
* A full output can be read from a terminal window using ascii characters
* The ability to save generations to a file for browsing later

The solution was coded in C++ in Sublime Text and Visual Studio and was tested to compile in g++ within an Ubuntu distro. The final solution was also run in ubuntu, so we suggest using this for compatibility’s sake. Use of Visual Studio and the built in C++ compiler in Windows is not recommended as is seems to throw an error based on the lack of a header file from our program (a resolution to this was attempted but was not successful).

Word count (excl. title & team members): 516