Cellular Automaton

# Our Approach

Making this cellular automaton was one of the hardest but exciting programs we have made so far. We decided to use C instead of C++ because we have prior experience with using it in the multi-paradigm programming module. We also decided that it will benefit us for the future if we learn and develop our C programming skills as many other popular programming languages are C-based. For making this program we met up during the lab. We decided to split up the work and allocate tasks to each other. To tackle the cellular automaton and have a better understanding of the program and how it works we all made a basic program with standard functionality. To keep up to date with the work we decided to use the online repository called GitHub where we uploaded the code. This was so all of us could work on it simultaneously and keep improving the cellular automaton over time.

# Difficulties Encountered

The first difficulty we experienced while writing the code for the cellular automaton was generating the rule sets and looping through the array and making it ask the user what width and height should be used for the output. Then the next problem we had was with the ‘case’ statements and writing the table to a basic structure and get it to calculate where should it go. Since, we wrote the table wrong it was not working as intended and wasn’t generating it the way we wanted it to.

We also had trouble with custom starting row where if the current width is greater than the starting width then it will give strange values. The solution to this problem was to add a maximum width and initialise the array based on that. However, when fixing this another issue occurred that caused an infinite loop. This was because the number of elements in the array was defined with the current width variable instead of the maximum width, causing the loop which initialises the array to try and access elements which did not exist.

One other problem that we encountered was with input validation, the program would get stuck in an infinite loop if an invalid input was entered. We got rid of this problem by writing another function that passes through the invalid input first and then asks the user to re-enter.

When implementing file output, we had an issue where the compiler would claim that the variable that points to the file was never declared. This turned out to be an issue with scope as the variable was declared within an if statement and so we fixed it by declaring it at the start of the function and then initialising it within the if statement.

We also had pointer related problems with game of life, we forgot to free memory thus resulting in memory leaks. We detected this problem using a tool called Valgrind and managed to fix it afterwards.

# Conclusion and Evaluation

Considering our team’s working effort: everyone contributed, finished their allocated tasks on time, and helped each other out if anyone got stuck on anything. We used Facebook and email to communicate and to stay connected and to discuss any project related queries. In the end, we finished the with time to spare, allowing us to go over our code and have 100% satisfaction before submitting the project.