Report for the group T - Cellular automata assignment

Kamil Krauze 2414951

Harry Inglis

Vojtech Loskot 2424633

There were many important considerations and challenges when approaching the assignment. We had the freedom to choose a language, user interface (menu vs CLI) as well as the actual way of solving the problem. This report outlines our reasoning behind some of our decisions as well as some challenges we faced and how we managed them.

We settled on the C programming language as we had more experience with it. At the end of the day we didn’t want to waste too much time on this decision as C and C++ are interoperable and we could always “upgrade” to C++ if we needed it. C++ might have actually been a better choice as we spend significant time debugging memory issues. Menu was the obvious choice as it creates (in this team’s opinion) a better user experience.

In our first meeting we decided on an architecture for our project which allowed us to work on separate parts of the program individually. We ended up with headers: binConv.h for manipulating binary numbers to and from decimal, grid.h for all the grid displaying/manipulation, IO.h for file operations and system.h for some system operation wrappers. We further split grid.h into files grid.c for the common functions and then grid1d.c

And grid2d.c for operations on the 1d and 2d grids respectively.

In our first meeting we decided to store the Cell values as chars as that seemed to be the best compromise between memory efficiency and simplicity. We didn’t want to over optimize our code at the start unnecessarily and in the end even with the char implementation it runs smoothly. We also made significant abstractions, so that if we decided to switch to bit arrays later it would be simple. This is why we’re not accessing the array directly but through getValueGrid and updateGridValue functions and why we aren’t using the char type directly, but through an alias Cell.

A problem we reached was making the grid size dynamic and user-editable. We started with defined constants for the row and column count and used these in all loops and checks. Swapping to dynamic arrays included replacing these with global variables and our structs with flexible array members (<https://en.wikipedia.org/wiki/Flexible_array_member>). Allocating became more complicated due to the necessity to have a flexible array of flexible arrays, but we found a source with a helpful example (https://stackoverflow.com/questions/54795235/flexible-array-of-flexible-arrays-in-c) . This created an issue when copying the grid to create a copy, which we tried to solve using memcpy, but in the end solved by instantiating a second grid using our initialize function, looping over it and copying all the values one by one.

Good naming of variables became important. When looping, the traditional i and j combined with col\_count and row\_count became confusing and led to several hard-to-debug errors. Renaming these variables to rowIndex, columnIndex and rowCount and columnCount, wherever possible made the debugging trivial as any error was now immediately obvious. This is probably the biggest takeaway from this exercise as it makes 2D arrays much, much simpler to deal with.

Overall, we faced some issues, but we managed to overcome those and gain experience.