**Algorithms and Data Structures – Sudoku Game Report 2**

**Introduction:**

The aim of this coursework was to create a working command prompt Sudoku Game and this report, submitted together with the game, is a deep analysis of it from two different points of view: the design and the code, meaning its user interface and meaning its main data structures and algorithm.

It is also an analysis of the developing process of the game and a comparison to the first ideas and features that were part of the original coding plan. The comparison between the finished application and the first ever version will show the impact of the literature review and of the analysis of different data structures and algorithms on the final product.

This report contains other 2 sections, the Critical Evaluation, divided in “1” and “2” and Appendices, containing some extra information.

**Body:**

**Critical Evaluation:**

The first thing that I am going to cover is the literature review of my previous report. I have completely changed my idea about Bryce’s way of printing the board (cplusplus, 2010) and decided to keep mine as it requires only a few lines of code and uses logic to print the board instead of just printing each single value one by one.

Regarding the Java code I found (Java Game Programming, 2021) I initially implemented the Enum as I said I would but things changed when I had to decide how to let the game know if the match was finished. At that point I understood that all those states stored in an Enum were not necessary, as a cell cannot be overwritten if there is interference with another cell, and as the game will only end when there are no empty cells, making the NO\_GUESS, GUESSED and WRONG states useless.

As I mentioned in my past literature review, I would have removed the cell’s attribute that contains the value that that specific cell should contain as the sudokus I am generating can actually have more than one solution.

Regarding the appearance I found some color inspiration in my past literature review but nothing extreme as it is some kind of game for children and colors are not necessary. Also, implementing a different color for each block would have been a bit challenging given the way I decided to print the board in. These are the motives that lead me to just keep my first board design but adding some tiny separators and minor colors to make everything neater. For colors I have used two libraries, Colorama and termcolor, as in certain situations and under specific circumstances using Colorama was way slower than using colored.

As you can very well see from my past analysis, I was thinking that one of the best ways to generate my sudoku grid would be to utilise backtracking. My sudoku grids (nine times nine and four times four) run a backtracking algorithm twice to be filled in, but how did I do it?  
After reading online that by randomly filling in the diagonal blocks, a sudoku grid becomes way easier to be solved with an algorithm, I have implemented a little algorithm that does it. After that a backtracking algorithm fills in the grid, returning a full grid. Now the only step left was to empty it, and leave only some clues. I decided to leave 25 clues for a difficult sudoku, 35 for a medium one and 45 for an easy one. The algorithm that I have implemented keeps removing a number and checking if the sudoku is still solvable, if it is it can go on removing numbers, otherwise a new grid is generated.

In the end I did not actually implement an algorithm to fully fill in a grid as I said I might do but I made one to partially fill it and a backtracking algorithm that does the rest of the job.

Regarding the choice of using Python I ended up being completely happy with it, Python just gave me a lot of opportunities during the implementation of this game and made my life way easier, after all that is what you can expect from one of the most utilized programming languages. I ended up using a bit more than two libraries, but, as defined from the beginning, none of them adds pre made algorithms to my game, but only personalization and minor features. I also did not encounter any problem with speed, which I was thinking I would.

By analysing the Sudoku game I developed and the list of features that I said I would add it is noticeable that they have all been implemented: the detailed instructions introducing the player into the game, an easy and clean interface that is easy to understand, a smart way of tiding up the terminal’s output every iteration, a coordinate system for the movements and last but not least a countdown that uses threading to run at the same time of the game, parallelly.

Regarding the architecture I have actually implemented more classes than I said I would: a Main class that manages the entire system, a Board class that manages the board and everything that happens within it, multiple Cell classes that store the numbers, a GameManager class that handles undo, redo and saving game to file to replay, a GridGenerator class that handles the backtracking for both 9x9 grids and 4x4 grids, a Game class that stores a game so it can be written to file and last a ReplayBoard class that handles the replay of a match.

I mentioned multiple times that I would have found it useful to use an enum data structure but after thinking about it more accurately I decided not to use that, but instead to use a simple Boolean attribute called overwritable

I also talked about choosing between 1d arrays and 2d arrays to store my data. Since I decided to develop my Sudoku in a particular way, using ids for cells, a 1d array was the best choice for myself. But regarding backtracking I thought it would be more useful and quicker to adopt a 2d array to store the cell’s content since I just needed to generate some numbers after all.

Something went exactly like planned, and that is the logic behind undoing and redoing movements, the program has a list of movements, we will call it list one, which, every time a movement is made, including undo and redo, that movement is added into list one. Every time a move is made it is also added to other two lists, we will call them list two and list three, but not undo and redo movements. List three is used when undoing, in fact moves are popped off it, list two is used when redoing, since it contains moves that were undone, unlike list three. And in the end, to avoid branching, when a user undoes and makes a new move, the branch coming after the last undone move is deleted. To save the games to replay, simply, list one is saved into a file through serialization (I have used the pickle library to do so) when the user decides to save the match and quit.

**Conclusion:**

To conclude my analysis, after all my considerations, I can say that the Sudoku ended up looking basically like planned but ended up utilising quite different algorithms and data structures from what was planned.

I was and I am still happy with my choice of using Python as the project programming language since the Sudoku ended up looking great and providing a very wide range of features, fit for a simple game.

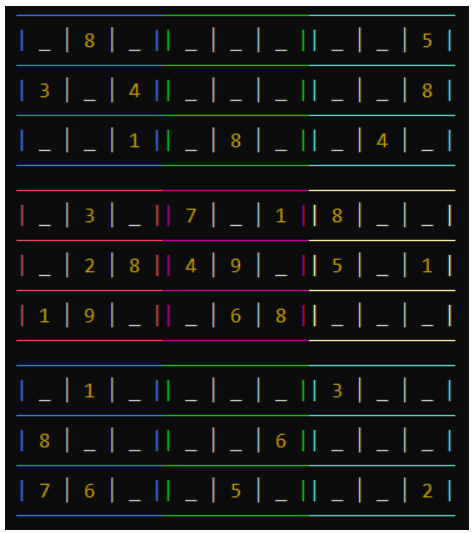
**References:**

- cplusplus, (2010, December 3). *Sudoku Game*. Retrieved March 3, 2022 from <http://www.cplusplus.com/forum/beginner/32467/>

- Java Game Programming, (May, 2021). *Java Graphics Programming Assignment – Sudoku*. Retrieved March 3, 2022 from <https://www3.ntu.edu.sg/home/ehchua/programming/java/JavaGame_Sudoku.html>

**Appendices:**

- **Appendix A**: Sudoku grid.



**Figure 1** Sudoku grid (cppsecrets, 2021)