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

**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.

- All the material learnt from the lectures and the information gathered during my online research will help me with this task.

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 kept 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.

- After deep research about Sudokus to discover different details needed for the generation of the partly filled grid, I have read that, depending on the given numbers, there could be multiple ways of solving one Sudoku so it is a reason for me to implement the Enum data structure and remove the attribute which holds the number that a certain Cell should contain.

- To comment

- During my research I also found a nice grid idea that I could possibly use instead of my current one (see Appendix A).

- Implementing a different color for each block would have been a bit challenging given the way I decided to print the board in. I just kept my first board design but added some tiny separators and minor colors to make everything neater.

- Most of the Sudokus that I have found online utilize the backtracking algorithm to generate a Sudoku grid. I consider using backtracking the best solution

- To comment

- Despite that I might still try to implement my own algorithm to create a puzzle, generating each time a random number and checking if another same number is already in the same row/column/box, otherwise it can be written and the next cell can be examined.

- To comment

- The other issue is that not only the grid needs to be filled, but values also need to be removed. “define a parameter p (where 0 ≤ p ≤ 1). Each cell in the grid is now considered in turn, and is removed with a probability of exactly 1 – p.” (Rhys, 2007). Some even suggest to just start adding numbers into an empty grid until one and only one solution is possible.

- To comment

- Out of all the programming languages I could have chosen from I decided to utilize Python. The reason behind that thinking is pretty simple: Python is the language I am more familiar with. Did I find myself in lack of speed?

- To comment

Elements I thought I would add:

- The first game’s output is a set of instructions that the user can read for as much time as needed and then start playing, of course the user can always re-display them if anything was missed.

- The game user interface is designed to have the least possible elements, in fact at the top is positioned the title of the game, in the middle the grid and at the bottom the user input section, where the user can choose the next move.

- Every time the user inputs something the command line is cleared and the three game sections (explained above) are reprinted, so the user does not get confused with previous outputs.

- The moves input will work through a coordinates system, like in a game of Battleship. The coordinates will be displayed on the top and left side of the grid and the user will be taught how to use them.

- There will be a countdown that will make the game more challenging giving the user only a few minutes to complete the puzzle. This is an extra feature and, if it will be implemented, the user will be allowed to switch it off.

- I worked hard on the indentation of every single line of output, so that it is not either too close to other text or to borders and corners. And I kept texts as short and informative as possible.

- I decided to display a fancy Sudoku game name (Vavassudoku) at the very top of the command prompt tab, to give it a little bit of a personal touch.

Regarding the architecture there follow my personal architecture choices:

- two main classes: a Board class that contains the game settings and the cells and the Cell class, which stores all the data required for every single cell to work properly.

- To know all the time if a cell is overwritable, empty and not empty I will implement an Enum data structure to select the Cell’s state from.

I actually did not use an Enum, but a simple Boolean attribute called overwritable

- To store the Cell objects I am currently using a 1d array since the array does not have a fixed size, meaning it will be faster than a 2d array. But I will consider making this array of fixed size, since the grid’s cell number will not change, and then change it into a 2d array.

I kept the 1d, easier to build an algorithm that manages it and works faster

- To record the history of the moves I will save every single movement as a string into a list. Managing undo e redo movements will be quite challenging and that is how I was thinking of implementing it: to go back (undo) the program will read the list, search for the current move, pop it off from the end of the list and add it to another list of movements to redo, and undo it. To go forward (redo) the program will read the list of movements to redo, and get the following move, redo it and pop it off from the beginning of the list and add it to the list of movements. Of course every time the user goes back with moves, and makes a new move, there will not be the possibility of redoing anymore.

- To comment

- Even if not needed I thought that using the help of some useful tools could make my Sudoku game look even better, those useful tools are libraries. Python is known to be the best programming language for libraries, but, since we are not allowed to import libraries that provide special data structures, for now I will only import two libraries called Colorama and Playsound, to respectively introduce colours and sounds into the game.

**Conclusion:**

To conclude my analysis, after all my considerations, I can say that the Sudoku ended up looking basically like planned but ended up utilizing 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. 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/>

- cppsecrets, (2021, August 12). *Sudoku Console Game using C++*. Retrieved March 3, 2022 from <https://cppsecrets.com/users/141731151079711010097114105641121044610510511611446979946105110/Sudoku-Console-Game-using-C00.php>

- GeeksForGeeks, (December, 2021). *Backtracking | Introduction*. Retrieved March 2, 2022 from <https://www.geeksforgeeks.org/backtracking-introduction/>

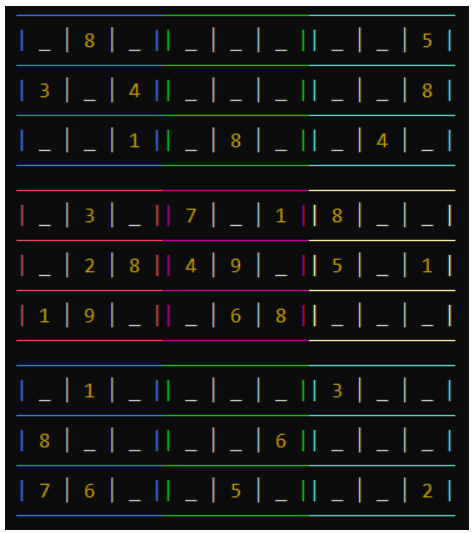
- Java Game Programming, (May, 2021). *Java Graphics Programming Assignment – Sudoku*. Retrieved March 3, 2022 from <https://www3.ntu.edu.sg/home/ehchua/programming/java/JavaGame_Sudoku.html>

- Rhys, L. (2007, May 1). *Metaheuristics can solve sudoku puzzles*. Retrieved March 2, 2022 from <https://link.springer.com/content/pdf/10.1007/s10732-007-9012-8.pdf>

- Wikipedia, (2021, October 31). *Java collections framework*. Retrieved March 2, 2022 from <https://en.wikipedia.org/wiki/Java_collections_framework>

**Appendices:**

- **Appendix A**: Sudoku grid.



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