Algorithms & Data Structures Report (Part 2)

For the second submission of this coursework we were tasked with implementing the command line sudoku game which was proposed in the initial design (the report from part 1). In the initial design features, algorithms and data structures were proposed which would lead to a fully operating command line sudoku game. In this report, the difference in features, algorithms and data structures concerning those proposed in the initial design and those implemented in the final version of the sudoku game will be discussed along with reflections upon what worked well, what didn't work well and given more time, what features could have improved the final implementation of the game?

In the initial design of the sudoku game the following features were proposed; an option to choose an easy, medium or hard mode which will vary the amount of the squares/numbers which are removed/not filled in from the sudoku grid, an option to select a varying size of grid from the options of a four by four grid or a nine by nine grid, undo/redo capabilities (regarding player moves\ inputs), game history, which will allow the user to continue with a game after they re-open the app and an optional variant of the classic sudoku game involving coloured shapes instead of the numbers.

Beginning with the first requirement (incorporating an easy, medium and hard mode), this was achieved in the final implementation and does vary the amount of squares which are unassigned after a sudoku solution has been found. Secondly, the option to choose the size of a grid from a four by four to a nine by nine grid was not implemented in the final implementation. Thirdly, undo/redo capabilities were implemented into the final implementation. Fourthly, the 'game history' feature that would allow users to continue playing a game after they re-open the app is a bit vague, however a feature of that nature was implemented - that being, a user has the ability to save the game during the game which will write the four main data structures (two nine by nine two dimensional arrays which are used to store the current state of the grid and the initial state of the grid before any user input and two stacks to store the users input commands on undo or redo stacks) to a text file saved under the users input. Upon the starting of the program the user is given the option to start a new game or load in a previously saved game, which if chosen the user must enter the name of the game they saved and the four data structures will be read into strings and then stored back into their respective data structure to resume from the last state of play. Finally, the last feature to include an option to play a coloured/shape version of sudoku was not implemented into the final implementation.

Refined list of features incorporated into the final implementation:

- Easy, medium or hard mode (varies remaining squares at the start of the game)
- Undo capability
- Redo capability
- Save Game
- Load in saved game

The algorithms in the final implementation are very similar to the algorithms proposed in the initial design (if not the same with some added validation), with the exception of the function to generate the complete sudoku grid. Instead of two loops being used to traverse the two dimensional array a recursive algorithm is used to attempt to insert a (random) number into a grid position, if it is safe to

do so it will recursively call the same function, however this time moving onto the next position in the grid?If at any point a number cannot be placed in the grid it will return false and an alternative number will be inserted into that position and then recursively called with the next position until the grid is completed in accordance with the constraints of a sudoku grid (see Figure 1 for initial menu and beginning of game). Other functions not spoke about in the previous report which were incorporated into the design is a linear search. This is used when finding the amount of remaining places in the grid and when searching through the undo stack to see if a position had been previously altered from its original value (0, used in this case to show an unassigned position) so that it may be undone to that state instead of back to 0.

The completed grid algorithm worked well as its performance is good with regards to the time taken for the amount data being used within this program. It is also a brute force method however and may require a more complex algorithm if more data/ much bigger grids were being used however this is not the case for my final implementation. The linear search algorithms also worked well as they their is no ascending / descending order to the undo stack or grid which might of warranted some sort of tree structure, however as previously stated there was no order to the items within the data structures so a linear search is most appropriate way to traverse them and find an item. In the case that the user inputs a value and there are no positions in the grid remaining the program will call a check grid function to check if the every number / position is within the constraints of a sudoku solution, if it is the game will end (after giving the user a chance to save it) else the game will continue after displaying an incorrect solution message (Figure 2 & 3). One algorithm which could have been made better is the algorithm which writes the data structures to a file, in the case that the user enters a name of a file already being used it will overwrite the data within it. It could have been made better by including some conformation message that a file by the same name will be overwritten if the user choose to continue.

The choice of a two dimensional array to store the grid and the current state of the grid are good ideas as their positions can be easily altered/found based on their index and can be traversed easily with one or two loops.The other choice to use stacks as the data structures for undo / redo capabilities worked well as they can easily be traversed and their LIFO (Last In First Out) system matches that of undo / redo very well with pop and push methods.

In conclusion the final implementation of the sudoku game was able to incorporate undo and redo, saving games and replaying those saved games along with basic fundamental of a sudoku game. This lead to an overall working Command line sudoku game as specified in the requirements, in the future, to improve the quiz I would add a timer and a leader board however what I did successfully get working, works well based on the algorithm and data structures it uses.

References:

https://www.geeksforgeeks.org/sudoku-backtracking-7/ - Used for design of completed grid algorithm

https://www.w3schools.com/java/java_files_create.asp - Used for research into writing to files for java

https://www.w3schools.com/java/java_files_read.asp - Used for research into reading from files for java

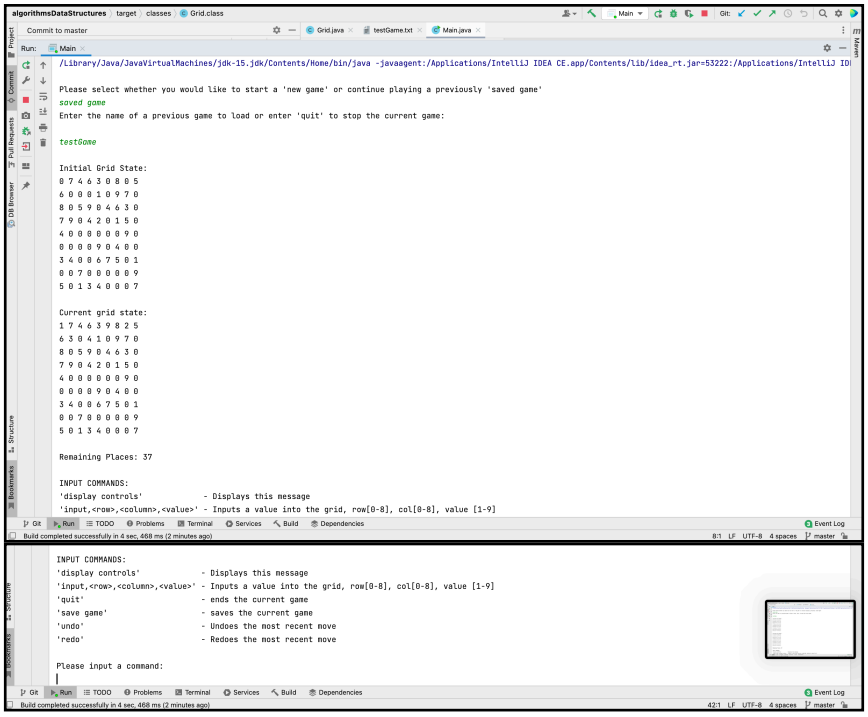

Figure 1: Initial game after selecting modes and choosing from presaged game (2 images)

```
Please input a command:
input,7,8,7

7 5 1 3 4 8 6 2 9
3 4 8 6 2 9 7 5 1
6 2 9 7 5 1 3 4 8
5 7 3 1 8 4 2 9 6
1 8 4 2 9 6 5 7 3
2 9 6 5 7 3 1 8 4
4 1 7 8 3 5 9 6 2
8 3 5 9 6 2 4 1 7
9 6 2 4 1 7 8 3 5


Input Value: Position(7,8) set to 7
Well done, You successfully completed the sudoku

Enter 'save'  to save the current game or 'quit' to end it:
```

Figure 2: Completed sudoku correctly

```
Please input a command:
input,7,8,1

7 5 1 3 4 8 6 2 9
3 4 8 6 2 9 7 5 1
6 2 9 7 5 1 3 4 8
5 7 3 1 8 4 2 9 6
1 8 4 2 9 6 5 7 3
2 9 6 5 7 3 1 8 4
4 1 7 8 3 5 9 6 2
8 3 5 9 6 2 4 1 1
9 6 2 4 1 7 8 3 5


Input Value: Position(7,8) set to 1

Unfortunately your solution is incorrect, keep trying;)

Please input a command:
```

Figure 3: Completed sudoku incorrectly