Sudoku Pair Solver:

Input: Parameter k (through terminal), .csv file (without header) containing 2 sudokus (S1,S2).

Output: Filled sudoku pair, both in terminal as well as in another .csv file.

Assumptions: Input of .csv file is in the specified format: each line of the file represents one row of the sudoku, and S2 is written below S1.

Implementation:

The input from the .csv file is read and converted into a 2-dimensional list containing both sudokus.

Propositions fed to the SAT solver are indexed as follows:

The proposition representing the n^{th} value in the i^{th} row and j^{th} column is numbered:

$n + j \times k^2 + i \times k^4$

This allows for a unique and concise numbering for each proposition, in both sudokus, as each row and each column contains exactly k^2 cells.

Clearly, the corresponding proposition for S2 will have index k^6 more than the index for this proposition in S1.

Various conditions are imposed on these propositions to satisfy the necessary conditions of a sudoku:

- Cell: One number exactly (each cell contains one number)
- Row: Each number exactly once
- Column: Each number exactly once
- Box: Each number exactly once
- Pairing: The corresponding cells of the sudokus don't contain the same number

Except for the box conditions, we use the add_atmost module in rows, columns and cells to ensure their conditions. Since for the boxes we include that each number appears at least once, this also ensures that each number appears at least once in every row and column, and that each cell contains at least one number.

Thus, for the rows (columns), we pass all the propositions representing a particular number and column (row) and put an upper bound of 1 on the number of true propositions in the list.

For each cell, we pass all the propositions representing a particular cell and put an upper bound of 1 on the number of true propositions in the list.

For the box conditions we create an array that stores the index of the proposition that represents the number 1 at the top left of each box, then pass all propositions representing one number in the box as a clause. This ensures that each number is present at least once in a box, and thus, each number is present at most once (as if a number appears more than once that means that not all numbers appear in that box).

The inputs are entered as assumptions into the SAT solver.

If the SAT solver returns None, there is no solution and thus None is printed to the terminal. Else, the model is converted to a 2-dimensional list that is converted to a .csv file and is also displayed in a formatted way in the terminal.

Limitations:

- Since the functionality of the SAT solver varies based on input, we cannot specify an exact time for execution.

- For large values of k (>5), the program starts to take a significant amount of time to display a result. For k = 6, the program takes approximately 20-25 seconds.
- In the case of multiple solutions, the program returns any one of them and not any specified optimal solution.

Sudoku Pair Generator:

Input: Parameter k (through terminal)

Output: Maximal, randomised sudoku pair with unique solution in the form of a .csv file

Assumptions: Expected output of .csv file is in the format specified for the input of the solver.

Implementation:
First, a filled, randomised sudoku pair is created using the functionality of the solver while ignoring input.
Then, the model of this sudoku pair is converted to a 2-dimensional list as well as an input list.
We then add the negation of this model as a clause to the solver. Thus, the original solution we obtained is no longer valid.
We now modify the input list by randomly removing any one element out of the current filled elements in the sudoku (removing a proposition from the input assumption list)
After this, we check if the SAT solver provides a solution. If it does not, then we keep the element removed. If it does, then we restore the element and continue to try to remove other elements.
This is done till we have attempted to remove every single entry of the sudoku while making sure there are no new solutions. Thus, we obtain a maximal number of holes (empty entries) while maintaining a unique solution for the partially filled sudoku pair.
This partially filled sudoku pair is then sent to a .csv file in the aforementioned format.
Additionally, we print this partially filled sudoku into the terminal.

Limitations:
- Since the functionality of the SAT solver varies based on the randomised sudoku generated, we cannot specify an exact time for execution.
- Since this program involves multiple usages of SAT solver, the number of which depends on k, the program takes a significant amount of time for larger values of k.