Sudoku Puzzle Code Documentation :

Used algorithms and optimization :

This documentation describes sudoku puzzle code which written create using the Depth First Algorithm (DFS) with Back tracking also I used two techniques (Optimizers) : **Constraint propagation and Remove Minimum Values (MRV)** which enhanced the efficiency of the code and minimized the searched space .

- **sudoku_solver(sudoku):**

This function accepts every (sudoku) which is 9*9 sudoku grid as input, It will check firstly the validity of entered sudoku by calling (is_valid_sudoku () function, then it will check if we got the solution of this sudoku by calling (solve ()) function, which returns the solution if it's there otherwise it will return 9*9 ndarray filled by -1.

- **is_valid_sudoku(sudoku):**

This function checks whether the input Sudoku grid is valid by ensuring that each row, column, and 3x3 grid contains unique numbers without any duplicates. If any duplication is found, the function returns a grid filled with -1 to indicate an invalid Sudoku.

- **solve(sudoku):**

function implements a **recursive backtracking algorithm** to solve the Sudoku puzzle.
It systematically explores all possible combinations of numbers in empty cells, and if a conflict or invalid solution is encountered, it backtracks to the previous cell and tries a different number.

- **get_possible_values():**

This function returns the possible values that can be placed in the given cell.
- it contains the domains- this technique is efficient cause it will calculate the possible values exclude any presented value in the same col, row and grid then it returns an array of possible values that can be placed in the current cell.

- **find_next_empty(sudoku):**

function efficiently searches for the next empty cell in the Sudoku grid with the fewest remaining possible values. By identifying cells that have the minimum number of possibilities, the function helps reduce the overall time required to solve the puzzle. This optimization aims to make correct choices earlier in the solving process, minimizing the need for backtracking steps and improving the efficiency of the Sudoku-solving algorithm

Time complexity:

The time complexity for this code is O(9^(n^2)) in the worst case. This depends on difficulty of the entered sudoku puzzle where if there is many possibilities to search for the cell with zero values and it will be faster if the search domain becomes less.

Space complexity:

The space complexity of the code is O(1), because it operates on the input Sudoku grid in place and uses a constant amount of additional memory, even with the inclusion of **sudoku.copy().**

Suggestions for further work:

- Searching for advanced and complicated algorithms and tech. to solve this puzzle to enhance efficiency and running time.
- Consider developing a user interface which make solving sudoku puzzle more exciting.
- Error handling, by adding more error messages for the wrong entered sudoku's.
- Testing the algorithm on wider dimension (larger than 9*9) and checking execution time specially for hard sudokus.