REPORT FILE

Problem statement: Sudoku Solver

Name: Mahima Bhat

Roll No: 202401100400116

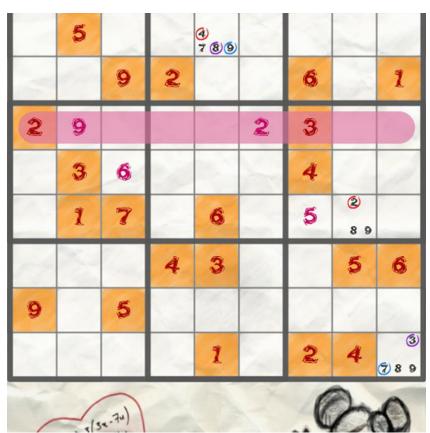
Course: Introduction to Al

Date: 10-03-2025

Introduction-

Sudoku is a popular logic-based number puzzle that requires filling a 9x9 grid so that each column, each row, and each of the nine 3x3 grids contain all digits from 1 to 9. The objective of this project is to develop a Python program that can efficiently solve any given Sudoku puzzle using the backtracking algorithm.

The provided Sudoku Solver takes an incomplete Sudoku board as input and fills in the missing numbers while ensuring all constraints are satisfied. The backtracking algorithm explores possible solutions by testing each potential number and backtracking when conflicts arise.



Methodology-

The Sudoku Solver is implemented using the following steps:

- 1. **Input the Puzzle:** The unsolved Sudoku puzzle is provided as a 9x9 matrix where empty cells are marked as zero.
- 2. **Validation Check:** The program verifies if placing a particular number in a cell is valid by checking:
 - 。 Row
 - 。 Column
 - 。3x3 Grid

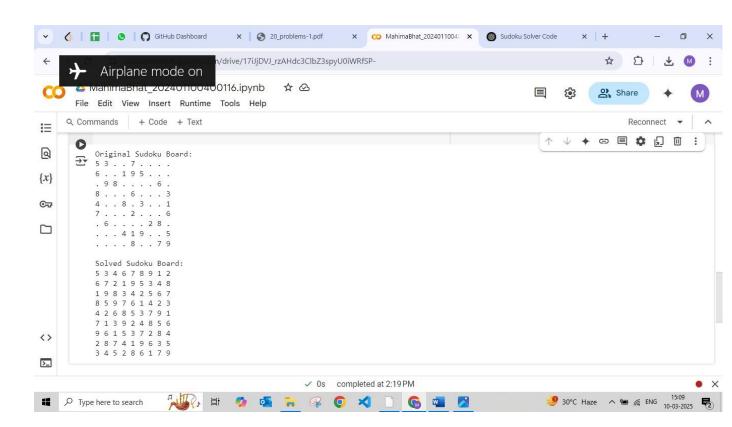
3. Backtracking Algorithm:

- The algorithm iterates through each cell in the puzzle.
- If an empty cell is encountered, the program attempts to place numbers from 1 to 9.
- If a valid number is found, it proceeds further; otherwise, it backtracks and tries alternative numbers.

4. **Output the Solution:** The solved Sudoku puzzle is displayed in a clear format.

```
5. def is valid (board, row, col, num):
      # Check if the number is not in the current row and column
7.
      for i in range(9):
          if board[row][i] == num or board[i][col] == num:
              return False
10.
11.
           # Check if the number is not in the 3x3 box
            box start row, box start_col = 3 * (row // 3), 3 * (col )
12.
// 3)
13.
            for i in range(3):
14.
                for j in range(3):
                    if board[box start row + i][box start col + j]
 == num:
16.
                        return False
17.
18.
           return True
19.
20.
        def solve sudoku(board):
21.
            for row in range (9):
               for col in range (9):
22.
23.
                    if board[row][col] == 0: # Empty cell found
                        for num in range(1, 10): # Try numbers 1 to
24.
9
25.
                            if is valid(board, row, col, num):
26.
                                board[row][col] = num
27.
                                if solve sudoku(board): #
 Recursively attempt to solve
29.
                                    return True
30.
31.
                                # Backtrack if the current number
 doesn't work
32.
                                board[row][col] = 0
33.
                       return False # Trigger backtracking
34.
           return True
35.
       def print sudoku(board):
37.
            for row in board:
38.
               print(" ".join(str(num) if num != 0 else '.' for num
  in row))
39.
        # Sample Sudoku Puzzle (0 represents empty spaces)
40.
41.
       sudoku board = [
42. [5, 3, 0, 0, 7, 0, 0, 0],
```

```
43.
             [6, 0, 0, 1, 9, 5, 0, 0, 0],
44.
             [0, 9, 8, 0, 0, 0, 0, 6, 0],
45.
             [8, 0, 0, 0, 6, 0, 0, 0, 3],
             [4, 0, 0, 8, 0, 3, 0, 0, 1],
46.
             [7, 0, 0, 0, 2, 0, 0, 0, 6],
47.
48.
             [0, 6, 0, 0, 0, 0, 2, 8, 0],
49.
             [0, 0, 0, 4, 1, 9, 0, 0, 5],
50.
             [0, 0, 0, 0, 8, 0, 0, 7, 9]
51.
         1
52.
53.
        print("Original Sudoku Board:")
54.
        print sudoku (sudoku board)
55.
56.
        if solve sudoku(sudoku board):
             print("\nSolved Sudoku Board:")
57.
58.
             print sudoku(sudoku board)
59.
         else:
             print("\nNo solution exists for the given Sudoku
  board.")
```



References/Credits for images-

https://d2.alternativeto.net/dist/s/sudoku-solver-and-

generator_214277_full.png?format=jpg&width=1200 &height=1200&mode=crop