The Sudoko Solver(back tracking method)

Course : B.Tech [Computer Science & Engineering]
Submitted By: *Challa Gokul Sai*
Submitted To: *Dr. Tanu singh*
Date: *30 nov 2025*

Abstract :-

This project implements a classic backtracking technique to solve a 9×9 Sudoku puzzle. The program accepts a partially filled grid from the user, where empty cells are represented using 0, and attempts to fill all missing positions while obeying the rules of Sudoku. The goal of this project is to demonstrate recursion, constraint checking, and matrix manipulation in the C programming language. The solver checks rows, columns, and 3×3 subgrids to determine whether a number can be safely placed in a given cell.

Problem Definition :-

Sudoku is a logic-based puzzle played on a 9×9 grid. The task is to fill the grid so that:

• Each row contains the numbers 1 through 9
• Each column contains the numbers 1 through 9
• Each 3×3 subgrid also contains the numbers 1 through 9

Manually solving Sudoku becomes more difficult with fewer clues. The objective of this project is to automate the solving process using the backtracking algorithm. When a number placement leads to a dead end, the algorithm reverses (backtracks) and tries another number. This process continues until the grid is completely solved or determined to be unsolvable.

System Design :-

Algorithm:

```
Step 1: Start
Step 2: Read the 9×9 Sudoku grid (0 represents empty cell)
Step 3: Search for the first empty cell
Step 4: For numbers 1 to 9:
  • Check if placing the number is valid
  • If valid, place it
  • Call the solver recursively
Step 5: If recursion succeeds — puzzle solved
Step 6: If none of the numbers fit backward — undo placement (backtrack)
Step 7: Continue until full grid is solved
Step 8: Print the solved grid
Step 9: End
```

Validity Checking Conditions :

A number is considered safe if:

• It is not already present in the same row
• It is not already present in the same column
• It is not present in the corresponding 3×3 block

These three checks ensure each move follows Sudoku rules.

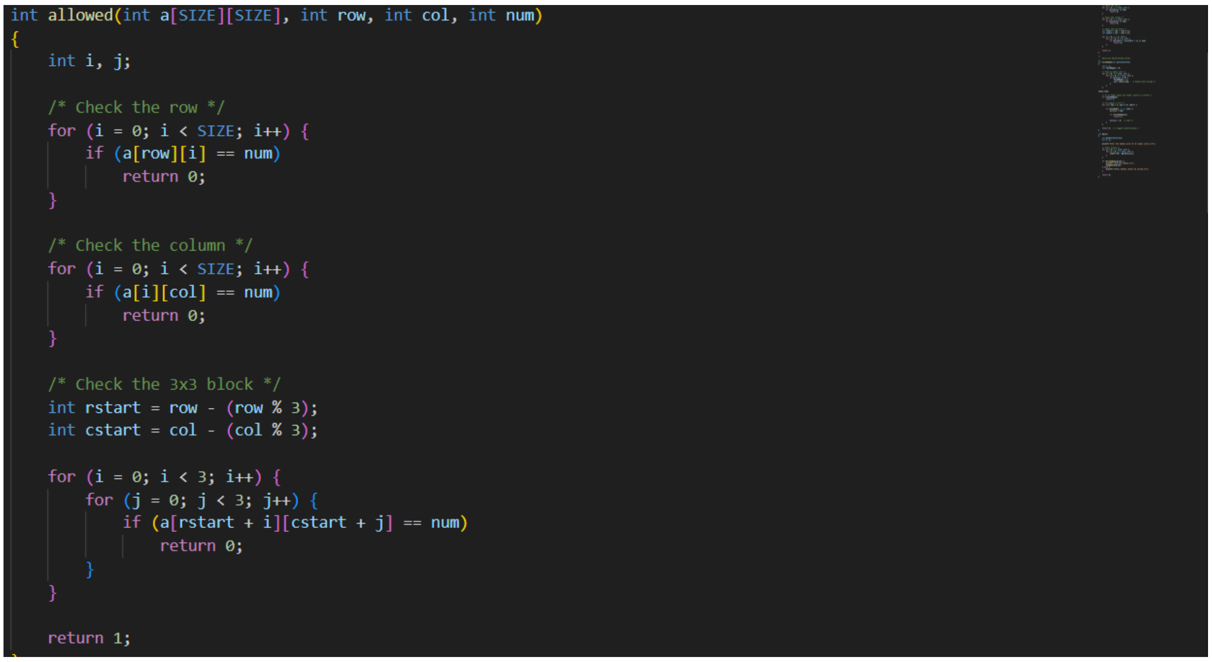
Implementation Details :-

Key Features Used

• 2D arrays for representing the Sudoku grid
• Recursion for exploring possible solutions
• Backtracking for undoing incorrect moves
• Nested loops for scanning rows and columns
• C conditional statements and function modularity

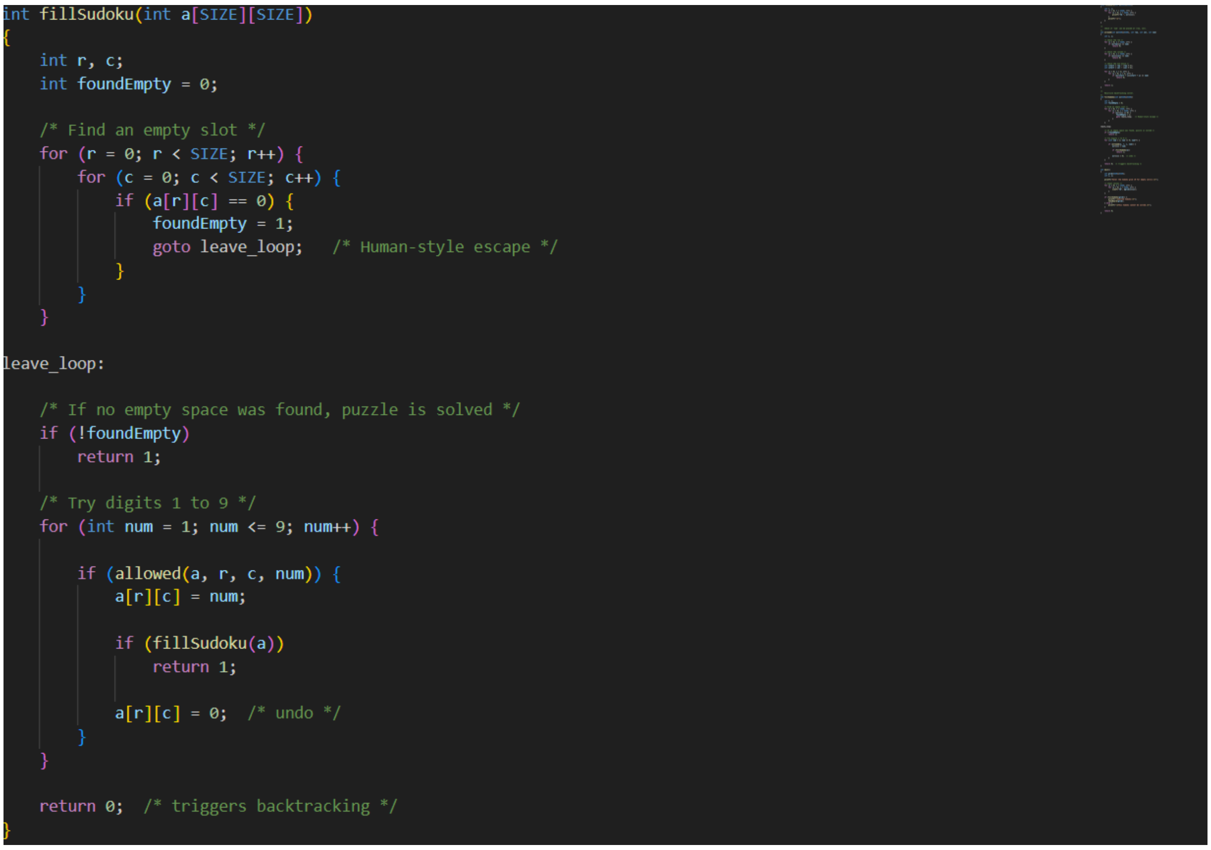
Important Code Snippets :-

1. **Checking if a Number Can Be Placed :**



```c
int allowed(int a[SIZE][SIZE], int row, int col, int num)
{
    int i, j;

    /* Check the row */
    for (i = 0; i < SIZE; i++) {
        if (a[row][i] == num)
            return 0;
    }

    /* Check the column */
    for (i = 0; i < SIZE; i++) {
        if (a[i][col] == num)
            return 0;
    }

    /* Check the 3x3 block */
    int rstart = row - (row % 3);
    int cstart = col - (col % 3);

    for (i = 0; i < 3; i++) {
        for (j = 0; j < 3; j++) {
            if (a[rstart + i][cstart + j] == num)
                return 0;
        }
    }

    return 1;
```

2. Backtracking Logic :



```c
int fillSudoku(int a[SIZE][SIZE])
{
    int r, c;
    int foundEmpty = 0;

    /* Find an empty slot */
    for (r = 0; r < SIZE; r++) {
        for (c = 0; c < SIZE; c++) {
            if (a[r][c] == 0) {
                foundEmpty = 1;
                goto leave_loop;   /* Human-style escape */
            }
        }
    }

leave_loop:

    /* If no empty space was found, puzzle is solved */
    if (!foundEmpty)
        return 1;

    /* Try digits 1 to 9 */
    for (int num = 1; num <= 9; num++) {

        if (allowed(a, r, c, num)) {
            a[r][c] = num;

            if (fillSudoku(a))
                return 1;

            a[r][c] = 0;  /* undo */
        }
    }

    return 0;  /* triggers backtracking */
}
```

Testing & Result :

Input Sudoku (0 = blank):

5 3 0 0 7 0 0 0 0

6 0 0 1 9 5 0 0 0

0 9 8 0 0 0 0 6 0

8 0 0 0 6 0 0 0 3

4 0 0 8 0 3 0 0 1

7 0 0 0 2 0 0 0 6

0 6 0 0 0 0 2 8 0

0 0 0 4 1 9 0 0 5

0 0 0 0 8 0 0 7 9

Output (Solved Sudoku):

5 3 4 6 7 8 9 1 2

6 7 2 1 9 5 3 4 8

1 9 8 3 4 2 5 6 7

8 5 9 7 6 1 4 2 3

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

7 1 3 9 2 4 8 5 6

9 6 1 5 3 7 2 8 4

2 8 7 4 1 9 6 3 5

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

The solver correctly filled all missing cells.

Conclusion :

The Sudoku Solver project successfully demonstrates the use of recursion and backtracking to solve logical puzzles efficiently. The program is able to handle partially filled grids and produce a correct solution whenever one exists. This project strengthened understanding of multidimensional arrays, function design, and constraint-based problem solving.

References :

• Based on Sudoku rules and backtracking principles
• Programming in C – Class Notes
• Online articles on recursion and puzzle-solving algorithms