# K.I.E.T. Group of Institutions

# Ghaziabad

**Name: Ayush Pandey**

**Branch:** CSE(AI) - B

**Roll No:** 12

**Date:** 11/03/2025

# Project Report On – Sudoku Solver

# Introduction

This project focuses on the development of a Sudoku solver, an algorithmic tool designed to efficiently solve any valid Sudoku puzzle. Using the backtracking technique, the program systematically fills in empty cells while ensuring adherence to Sudoku rules for rows, columns, and 3x3 sub-grids. The primary goal of this project is to demonstrate the practical application of algorithmic problem-solving and its capability to handle complex, logic-based challenges in an optimized manner.

# Methodology

The methodology for developing a Sudoku solver involves the following structured steps:

1. **Problem Analysis:**
    - Understand the rules of Sudoku, ensuring that every row, column, and 3x3 sub-grid contains unique numbers from 1 to 9.
    - Define the input format, where empty cells are represented by 0.
2. **Algorithm Selection:**
    - Choose the backtracking algorithm, a recursive approach to systematically explore potential solutions.
    - Ensure the algorithm checks for rule violations before placing a number.
3. **Implementation:**
    - Validation Function: Create a function to check whether a number can be placed in a specific cell without breaking Sudoku rules.
    - Backtracking Solver: Develop a recursive function to explore all possibilities for each empty cell, backtracking when conflicts arise.
    - User Interface: (Optional) Integrate functionality to take a Sudoku board input from users for a dynamic experience.
4. **Testing and Optimization:**
    - Test the solver with multiple Sudoku puzzles, including edge cases (e.g., very few clues or nearly complete boards).
    - Optimize the code for efficiency by reducing unnecessary computations during validation.
5. **Output Presentation:**
    - Format and display the solved Sudoku grid in a clear and user-friendly way, ensuring readability.

## Code:-

```
def get_sudoku_input():
    print("Enter your Sudoku puzzle, row by row (use 0 for empty cells):")
    board = []
    for i in range(9):
        row = list(map(int, input(f"Row {i + 1}: ").split()))
        if len(row) != 9:
            print("Each row must have exactly 9 numbers. Try again.")
            return get_sudoku_input()  # Restart input if invalid
        board.append(row)
    return board

def is_valid(board, row, col, num):
    for i in range(9):
        if board[row][i] == num or board[i][col] == num:
            return False
        if board[3 * (row // 3) + i // 3][3 * (col // 3) + i % 3] == num:
            return False
    return True

def solve_sudoku(board):
    for row in range(9):
        for col in range(9):
            if board[row][col] == 0:
                for num in range(1, 10):
                    if is_valid(board, row, col, num):
                        board[row][col] = num
                        if solve_sudoku(board):
                            return True
                        board[row][col] = 0
                return False
    return True

def print_board(board):
    for row in board:
        print(" ".join(str(cell) if cell != 0 else "." for cell in row))

# Take input from the user
sudoku_board = get_sudoku_input()

# Solve the puzzle and display the result
if solve_sudoku(sudoku_board):
    print("\nSolved Sudoku:")
    print_board(sudoku_board)
else:
    print("\nNo solution exists.")
```

## Output/Result:-

```
Enter your Sudoku puzzle, row by row (use 0 for empty cells):
Row 1: 5 3 0 0 7 0 0 0 0
Row 2: 6 0 0 1 9 5  0 0 0
Row 3: 0 9 8 0 0 0 0 6 0
Row 4: 8 0 0 0 6 0 0 0 3
Row 5: 4 0 0 8 0 3 0 0 1
Row 6: 7 0 0 0 2 0 0 0 6
Row 7: 0 6 0 0 0 0 2 8 0
Row 8: 0 0 0 4 1 9 0 0 5
Row 9: 0 0 0 0 8 0 0 7 9

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
```