K.I.E.T. Group of Institutions

Ghaziabad

Name: SHIVAM AGARWAL

Branch: CSE(AI) - D
Roll No: 11
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

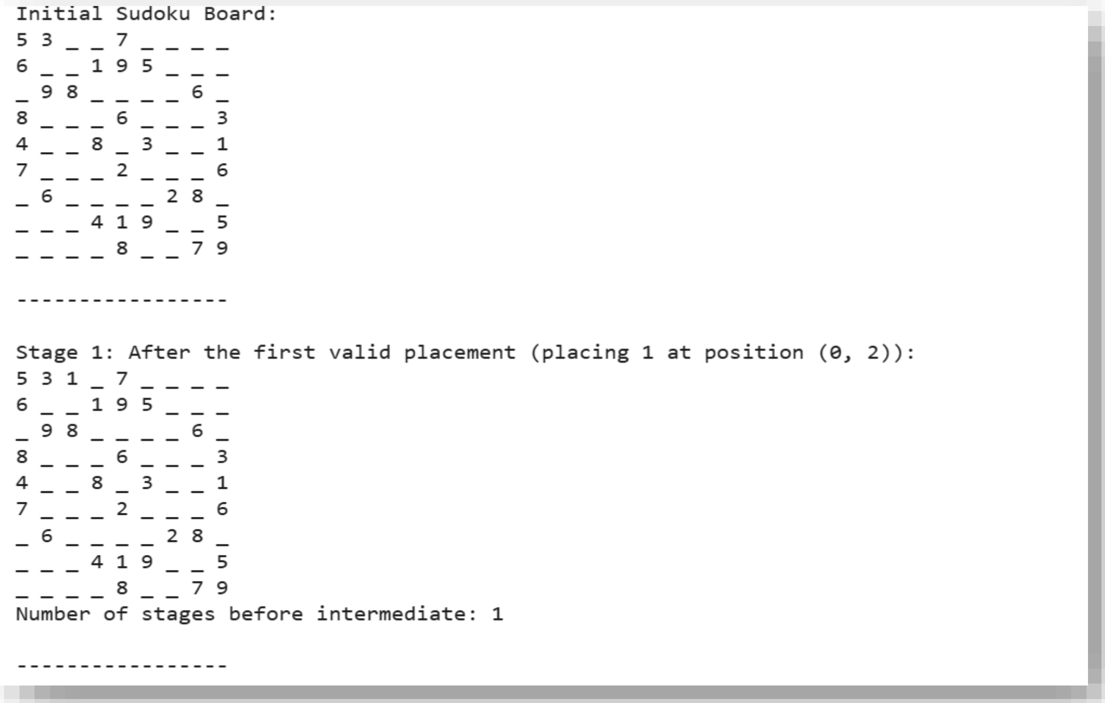
The methodology for developing a Sudoku solver involves the following structured steps:

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.

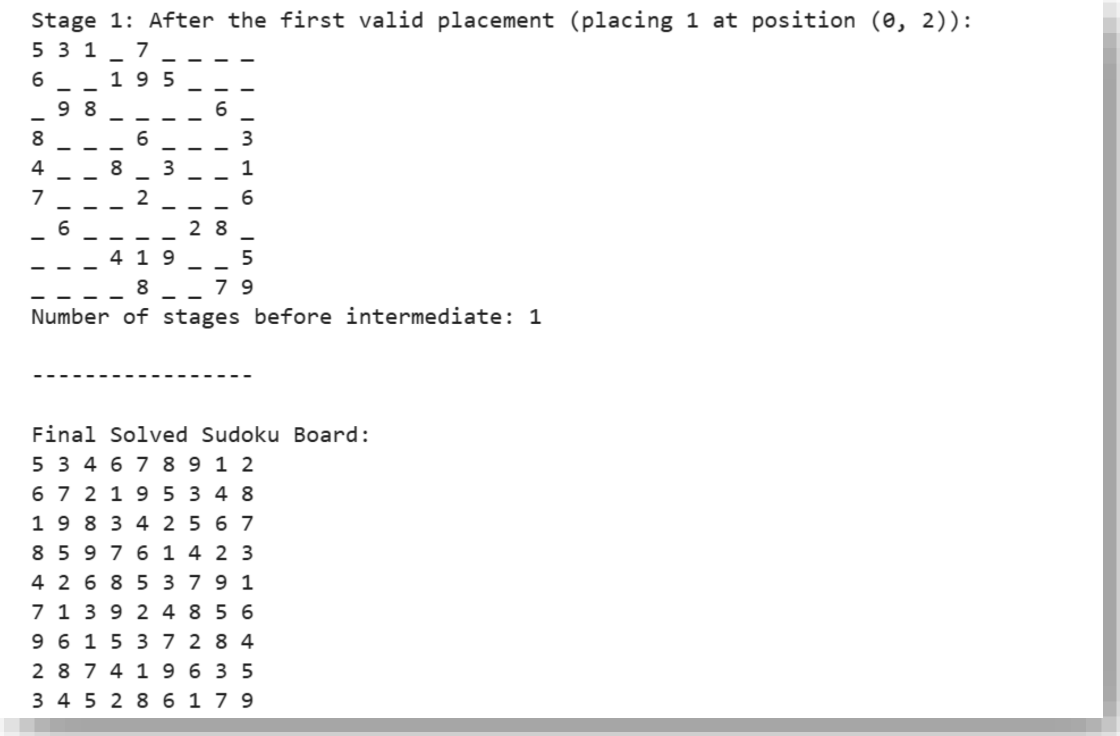
Code:-

```
def print_board(board):
    """Function to print the Sudoku board in a readable format, using '__' for empty
cells"""
    for row in board:
        print(" ".join(str(cell) if cell != 0 else "_" for cell in row))

def is_valid(board, row, col, num):
    """Check if placing a number at position (row, col) is valid"""
    # Check if the number already exists in the row
    for i in range(9):
        if board[row][i] == num:
            return False

    # Check if the number already exists in the column
    for i in range(9):
        if board[i][col] == num:
            return False

    # Check if the number already exists in the 3x3 grid
    start_row = row - row % 3
    start_col = col - col % 3
    for i in range(3):
        for j in range(3):
            if board[i + start_row][j + start_col] == num:
                return False

    return True

def solve_sudoku(board, first_placement_done=False, stages_counter=None):
    """Function to solve the Sudoku puzzle using backtracking"""
    if stages_counter is None:
        stages_counter = {"before": 0, "after": 0}  # Initialize stage counters

    for row in range(9):
        for col in range(9):
            # If the cell is empty (represented by __)
            if board[row][col] == 0:
                for num in range(1, 10):  # Try all numbers from 1 to 9
                    if is_valid(board, row, col, num):
                        board[row][col] = num
                        stages_counter["before"] += 1  # Increase the stage count before intermediate state

                        # Print one intermediate state (board after the first number placement)
                        if not first_placement_done:
                            print(f"Stage 1: After the first valid placement (placing {num} at position ({row}, {col})):")
                            print_board(board)
                            print(f"Number of stages before intermediate: {stages_counter['before']}")
                            print("\n-----------------\n")
                            first_placement_done = True

                        # Recursively solve the puzzle with this number
                        if first_placement_done:
                            stages_counter["after"] += 1  # Track stages after intermediate state

                        if solve_sudoku(board, first_placement_done, stages_counter):
                            return True

                        # Backtrack if the number didn't work out
                        board[row][col] = 0  # Reset the cell to empty

                return False  # No valid number found, need to backtrack
    return True  # Puzzle is solved when no empty cells are left

# Define the initial Sudoku puzzle (0 represents an empty cell)
sudoku_board = [
    [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]
]

# Initialize the stage counters before starting the solving process
stages_counter = {"before": 0, "after": 0}

# Print the initial state of the Sudoku puzzle
print("Initial Sudoku Board:")
print_board(sudoku_board)
print("\n-----------------\n")

# Solve the Sudoku puzzle
if solve_sudoku(sudoku_board, stages_counter=stages_counter):
    print("Final Solved Sudoku Board:")
    print_board(sudoku_board)
else:
    print("No solution exists")
```

Output/Result:-



```
Initial Sudoku Board:
5 3 _ _ 7 _ _ _ _
6 _ _ 1 9 5 _ _ _
_ 9 8 _ _ _ _ 6 _
8 _ _ _ 6 _ _ _ 3
4 _ _ 8 _ 3 _ _ 1
7 _ _ _ 2 _ _ _ 6
_ 6 _ _ _ _ 2 8 _
_ _ _ 4 1 9 _ _ 5
_ _ _ _ 8 _ _ 7 9

-----------------

Stage 1: After the first valid placement (placing 1 at position (0, 2)):
5 3 1 _ 7 _ _ _ _
6 _ _ 1 9 5 _ _ _
_ 9 8 _ _ _ _ 6 _
8 _ _ _ 6 _ _ _ 3
4 _ _ 8 _ 3 _ _ 1
7 _ _ _ 2 _ _ _ 6
_ 6 _ _ _ _ 2 8 _
_ _ _ 4 1 9 _ _ 5
_ _ _ _ 8 _ _ 7 9
Number of stages before intermediate: 1

-----------------
```



```
Stage 1: After the first valid placement (placing 1 at position (0, 2)):
5 3 1 _ 7 _ _ _ _
6 _ _ 1 9 5 _ _ _
_ 9 8 _ _ _ _ 6 _
8 _ _ _ 6 _ _ _ 3
4 _ _ 8 _ 3 _ _ 1
7 _ _ _ 2 _ _ _ 6
_ 6 _ _ _ _ 2 8 _
_ _ _ 4 1 9 _ _ 5
_ _ _ _ 8 _ _ 7 9
Number of stages before intermediate: 1

-----------------
```

```
Final Solved Sudoku Board:
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
```