Introduction to AI-Report

Problem Statement

Problem Statement: Sudoku Solver

Name:Utkarsh Singh

Roll.No-51

Branch-CSE(AI)-D

University roll.no-202401100300271

Introduction

The goal is to develop an algorithm to solve a Sudoku puzzle automatically within a mobile game. The puzzle consists of a 9x9 grid, where some cells are filled with numbers (1-9) and others are empty (represented by 0). The solver must identify the empty cells, validate potential numbers for each cell based on Sudoku rules (no repeated numbers in any row, column, or 3x3 subgrid), and fill in the numbers using a backtracking approach.

The backtracking algorithm attempts placing numbers in the empty cells. If a number is valid, the solver proceeds to the next empty cell. If placing a number leads to a conflict later, it backtracks and tries a different number. The algorithm continues until the board is completely filled or it determines that no solution exists.

Methodology

1. **Solve Sudoku using Backtracking**: The `solve_sudoku` function uses recursion to solve the puzzle. It tries numbers from 1 to 9 in each empty cell and checks their validity.

2. **Find Empty Cells**: The `find_empty` function scans the board to identify the next empty cell (represented by 0). This determines where the next number should be placed.

3. **Check Validity of Placement**: The `is_valid` function ensures placing a number is valid by checking the row, column, and 3x3 subgrid for conflicts.

4. **Recursive Backtracking**: If a number is valid, it's placed in the empty cell, and the function recursively attempts to solve the rest. If no valid number is found, it backtracks.

5. **Backtrack on Invalid Solution**: If placing a number doesn't lead to a solution, the number is removed (set to 0), and the function tries another number or backtracks further.

6. **Print Sudoku Board**: The `print_board` function displays the Sudoku board in a readable format with appropriate grid lines after every 3 rows and columns for clarity.

7. **Initial Setup and Execution**: An initial Sudoku board is provided, and the `solve_sudoku` function is called to attempt solving it. The solution, if found, is printed; otherwise, a failure message appears.

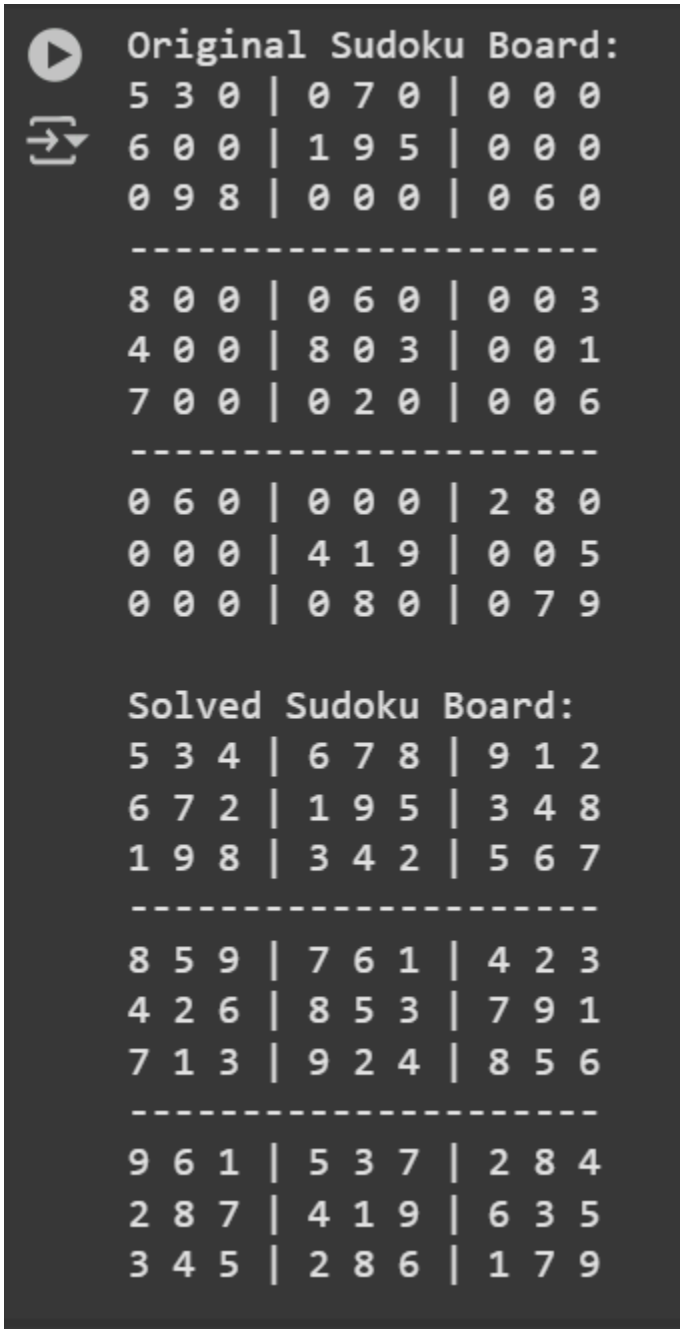
CODE

```
def solve_sudoku(board):
    """
    Solve the Sudoku puzzle using backtracking.
    """
    empty = find_empty(board)
    if not empty:
        return True  # Puzzle is solved if no empty cells are left
    else:
        row, col = empty

    for num in range(1, 10):  # Try numbers from 1 to 9
        if is_valid(board, row, col, num):
            board[row][col] = num  # Place the number

            if solve_sudoku(board):  # Recur to solve the rest
                return True

            board[row][col] = 0  # Backtrack if the solution is invalid

    return False  # Trigger backtracking if no number works

def find_empty(board):
```

```
                """
    Find the next empty cell (represented by 0) in the board.
                """
                for row in range(9):
                  for col in range(9):
                  if board[row][col] == 0:
        return (row, col)  # Return the position of the empty cell
          return None  # If no empty cells are found


          def is_valid(board, row, col, num):
                """
      Check if placing `num` in `board[row][col]` is valid.
                """
                  # Check the row
                 for i in range(9):
                if board[row][i] == num:
                        return False


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


               # Check the 3x3 subgrid
```

```
    start_row, start_col = 3 * (row // 3), 3 * (col // 3)
    for i in range(3):
        for j in range(3):
            if board[start_row + i][start_col + j] == num:
                return False
    return True

def print_board(board):
    """
    Print the Sudoku board in a readable format.
    """
    for row in range(9):
        if row % 3 == 0 and row != 0:
            print("-" * 21)  # Print a horizontal line after every 3 rows
        for col in range(9):
            if col % 3 == 0 and col != 0:
                print("|", end=" ")  # Print a vertical line after every 3 columns
            print(board[row][col], end=" ")
        print()
```

Result/Output:



```
Original 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

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
```