SUDOKU SOLVER

(Using backtracking algorithm)

Name: Prateek Rai

Roll number: 202401100300179

Course: Introduction To AI

INTRODUCTION

Overview:

Sudoku is a widely recognised logic-based number placement puzzle. The objective of the game is to fill a 9x9 grid with digits so that each column, each row, and each of the nine 3x3 subGrids (also called boxes) contains all the digits from 1 to 9 without repetition. This puzzle-solving task requires logical thinking and often involves trial and error.

Problem Statement:

The challenge of solving a Sudoku puzzle manually can be time-consuming, especially when attempting to solve puzzles with higher difficulty levels. This report presents an automated solution for solving Sudoku puzzles using the Backtracking algorithm. The Backtracking approach is a form of depth-first search where we explore possible solutions and backtrack when we encounter an invalid state.

Objective:

The objective of this report is to describe the implementation of the Sudoku solver using the Backtracking algorithm. It will explain the steps taken to solve a given Sudoku puzzle and present the solution obtained using the algorithm.

Scope:

This report covers:

- Introduction to the Sudoku puzzle.
- A detailed explanation of the Backtracking algorithm.
- Implementation of the algorithm in Python.
- Presentation of the results and the time complexity of the solution.

METHODOLOGY

Problem Definition

The Sudoku puzzle consists of a 9x9 grid, divided into 9 smaller 3x3 subGrids. Some cells are pre-filled with numbers (from 1 to 9), and the objective is to fill the remaining cells while ensuring no repetition occurs within any row, column, or 3x3 subGrid.

Backtracking Algorithm

Backtracking is a general algorithm used to find solutions to problems by incrementally building candidates for solutions and discarding those that fail to satisfy the constraints. For the Sudoku solver:

- **Step 1**: We scan the grid to find an empty cell.
- **Step 2**: We try placing digits 1 to 9 in that cell, checking if placing each number respects the Sudoku rules (no repetition in row, column, and subGrid).
- **Step 3**: If a valid number is found, the algorithm recursively tries to fill the next empty cell.
- **Step 4**: If an empty cell cannot be filled with a valid number, the algorithm backtracks, undoing the last filled cell and trying a different number.
- **Step 5**: The process repeats until the entire grid is filled or no solution exists (in which case, the puzzle is unsolvable).

Steps of Implementation

1. **Input Representation**: The Sudoku puzzle is represented as a 2D list in Python, where zeros represent empty cells.
2. **Validation Function**: A function to check whether a number can be safely placed in a given row, column, and subGrid.
3. **Backtracking Function**: The recursive function that attempts to place numbers in empty cells and backtracks when necessary.
4. **Termination**: The algorithm terminates once all cells are filled with valid numbers.

Time Complexity

The time complexity of the Backtracking algorithm for solving Sudoku is difficult to compute exactly because it depends on the puzzle's difficulty and how many recursive calls are made. However, in the worst case, the algorithm may need to try all possibilities (9^81 potential solutions), though heuristics and constraints typically reduce this drastically.

CODE

```
# Sudoku Solver using Backtracking Algorithm

# Function to print the Sudoku board
def print_board(board):
    for row in board:
        print(" ".join(str(num) if num != 0 else '.' for
num in row))

# Check if placing a number in a specific row and column
is valid
def is_valid(board, row, col, num):
    # 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 box
    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

# Function to solve the Sudoku board using backtracking
def solve_sudoku(board):
    for row in range(9):
        for col in range(9):
            # Find an empty spot (denoted by 0)
            if board[row][col] == 0:
                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

                        # Recursively try to solve the
rest of the board
                        if solve_sudoku(board):
                            return True

                        # If it doesn't work, backtrack
                        board[row][col] = 0
```

```
                return False  # No valid number found,
backtrack
    return True  # Sudoku is solved
```

```
# Function to take input from the user for the Sudoku
board
def input_board():
    board = []
    print("Enter the Sudoku puzzle row by row (use 0 for
empty cells):")
    for i in range(9):
        while True:
            try:
                row = list(map(int, input(f"Row {i+1}:
").strip().split()))
                if len(row) == 9 and all(0 <= num <= 9
for num in row):
                    board.append(row)
                    break
                else:
                    print("Invalid row, please enter
exactly 9 numbers (0-9)!")
            except ValueError:
                print("Invalid input, please enter only
integers!")
    return board
```

```
# Main function to execute the Sudoku solver
def main():
    # Get input from user
    sudoku_board = input_board()
```

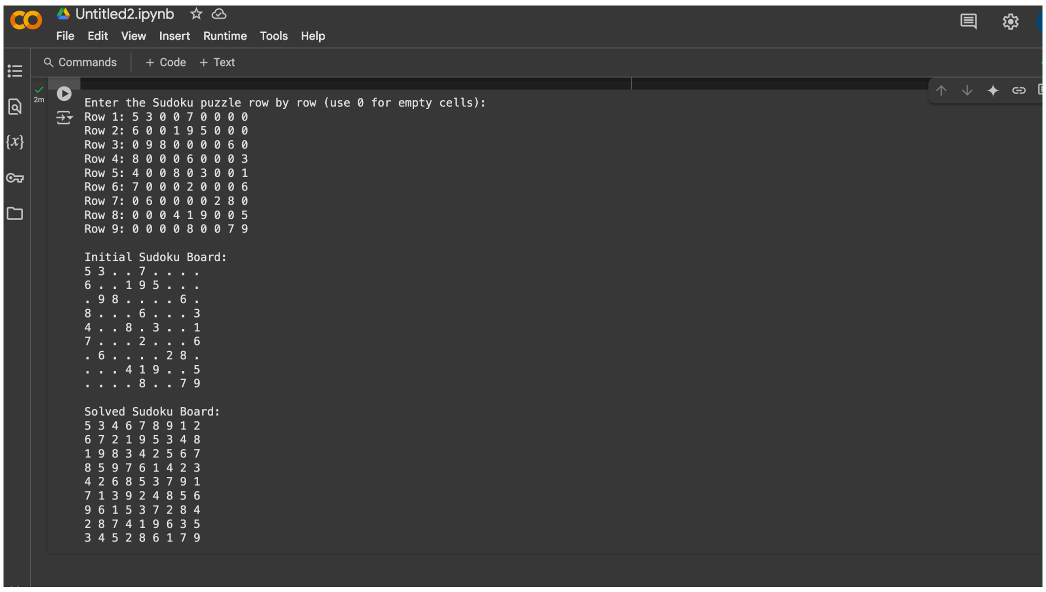
```python

    # Print the initial board
    print("\nInitial Sudoku Board:")
    print_board(sudoku_board)

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

# Run the main function
if __name__ == "__main__":
    main()
```

SCREENSHOT OF OUTPUT



```
Enter the 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

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

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
```