KIET
GROUP OF INSTITUTIONS
Connecting Life with Learning

SUDOKU SOLVER

SUBJECT: INTRODUCTION TO AI

MCA – 1st year (II sem)

Session: 2024-2025

Submitted To:

Ms. Komal Salgotra

(Assistant Professor)

Submitted By:

Yashvi Chaudhary – 202410116100253

Vaishnavi Yadav – 202410116100234

Vidushi Agrawal – 202410116100243

Vaibhav Singh Kalura – 202410116100233

INTRODUCTION

What is Sudoku?

Sudoku is a popular logic-based number puzzle that is played on a 9x9 grid, which is further divided into nine 3x3 sub grids. The puzzle starts with some numbers already pre-filled, and the player must fill in the remaining empty cells while following specific rules.

The objective of Sudoku is to fill the entire 9x9 grid so that:

1. Each row contains numbers from 1 to 9, with no repetition.
2. Each column contains numbers from 1 to 9, with no repetition.
3. Each 3x3 sub-grid (also called a "box" or "region") contains numbers from 1 to 9, with no repetition.

Sudoku puzzles come in varying difficulty levels, from easy to extremely challenging, depending on the number of pre-filled numbers and their placement. The game is widely played worldwide, both in newspapers and digital formats, as a means of entertainment and mental exercise.

Objective of the Project

The primary goal of this project is to develop a program that can automatically solve any valid Sudoku puzzle using a backtracking algorithm. Instead of manually solving a Sudoku puzzle, which can sometimes be time-consuming and complex, this program will take an unsolved Sudoku board as input and produce the completed solution.

The program will:

- Take a partially completed Sudoku puzzle as input.
- Apply an efficient solving algorithm (backtracking).
- Fill in all the missing numbers while ensuring Sudoku rules are followed.
- Output the solved Sudoku grid.

This project is a practical implementation of computational problem-solving and serves as a great example of algorithmic thinking.

Significance of the Project

This Sudoku Solver project is important for several reasons:

1. **Automates the Sudoku-solving process**
    - Instead of solving puzzles manually, which can be slow and difficult, this program provides a quick and reliable solution using a systematic approach.
2. **Demonstrates an efficient use of recursive algorithms**
    - The backtracking algorithm used in this project is a powerful problem-solving technique that systematically explores all possibilities while eliminating incorrect solutions efficiently.
3. **Enhances understanding of Constraint Satisfaction Problems (CSPs)**
    - Sudoku is an example of a Constraint Satisfaction Problem (CSP), which is a class of problems where a set of constraints must be satisfied.
    - Solving Sudoku helps in understanding constraints, recursion, and logical problem-solving, which are valuable concepts in computer science and artificial intelligence.
4. **Can be extended for AI-based Sudoku solvers**
    - This project lays the foundation for more advanced Sudoku solvers that can incorporate machine learning or artificial intelligence techniques to predict and solve Sudoku puzzles more efficiently.

METHODOLOGY

Algorithm Used: Backtracking

Backtracking is a recursive algorithm used to solve constraint-based problems by trying all possible combinations.

How Backtracking Works in Sudoku:

1. Find an empty cell in the Sudoku grid.
2. Try placing a number (1 to 9) in that cell.
3. Check if it's valid according to Sudoku rules.
4. If valid, recur for the next empty cell.
5. If a conflict arises, backtrack (remove the number and try the next possibility).
6. Repeat until the grid is completely filled.

Steps to Solve Sudoku Using Backtracking

1. Check for an empty cell (if none, the puzzle is solved).
2. Try numbers from 1 to 9, checking if each is valid.
3. If a valid number is found, place it and move to the next empty cell.
4. If stuck, remove the last placed number (backtrack) and try the next possible value.
5. Repeat until all cells are filled correctly.

Tools & Technologies Used

- **Programming Language:** Python
- **Algorithm:** Backtracking
- **IDE Used:** Google Colab
- **Libraries Used:** None (Standard Python)

CODE

```
def print_board(board):
    """Function to print the Sudoku board in a readable format."""
    for i in range(9):
        if i % 3 == 0 and i != 0:
            print("- - - - - - - - - - - - ")
        for j in range(9):
            if j % 3 == 0 and j != 0:
                print(" | ", end="")


            if j == 8:
                print(board[i][j])
            else:
                print(str(board[i][j]) + " ", end="")


def find_empty_cell(board):
    """Finds an empty cell (0) in the Sudoku grid."""
    for i in range(9):
        for j in range(9):
            if board[i][j] == 0:
                return (i, j)  # Return row, column
    return None


def is_valid(board, num, position):
```

```
    """Checks whether a number can be placed at a given position."""
    row, col = position

    # Check row
    for i in range(9):
        if board[row][i] == num and i != col:
            return False

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

    # Check 3x3 sub-grid
    box_x = col // 3
    box_y = row // 3

    for i in range(box_y * 3, box_y * 3 + 3):
        for j in range(box_x * 3, box_x * 3 + 3):
            if board[i][j] == num and (i, j) != position:
                return False

    return True

def solve_sudoku(board):
    """Solves the Sudoku puzzle using backtracking."""
```

```
    empty_cell = find_empty_cell(board)
    if not empty_cell:
        return True  # Puzzle solved!

    row, col = empty_cell

    for num in range(1, 10):
        if is_valid(board, num, (row, col)):
            board[row][col] = num

            if solve_sudoku(board):
                return True

            board[row][col] = 0  # Backtrack

    return False  # No solution found

def get_user_input():
    """Takes user input for a 9x9 Sudoku board."""
    print("Enter your Sudoku puzzle row by row (use 0 for empty cells):")
    board = []
    for i in range(9):
        while True:
            try:
                row = list(map(int, input(f"Enter row {i+1} (9 space-separated numbers): ").split()))
                if len(row) == 9 and all(0 <= num <= 9 for num in row):
```

```
                board.append(row)
                break
            else:
                print("Invalid input. Please enter exactly 9 numbers (0-9).")
        except ValueError:
            print("Invalid input. Please enter only numbers.")
    return board

# Get user input
sudoku_board = get_user_input()

print("\nOriginal Sudoku Board:")
print_board(sudoku_board)

if solve_sudoku(sudoku_board):
    print("\nSolved Sudoku Board:")
    print_board(sudoku_board)
else:
    print("\nNo solution exists.")
```

OUTPUT

1. Valid Sudoku Grid

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

```
Original Sudoku Board:
0 5 0  | 0 0 0  | 4 0 0
1 6 0  | 8 0 0  | 7 0 5
4 0 0  | 0 0 0  | 0 2 6
- - - - - - - - - - - - -
0 4 9  | 0 0 0  | 0 0 0
8 0 5  | 6 0 0  | 0 0 1
0 0 0  | 0 0 0  | 8 7 0
- - - - - - - - - - - - -
0 0 0  | 3 9 0  | 0 6 4
0 0 0  | 0 0 6  | 0 1 0
9 0 0  | 0 2 0  | 0 0 0
```

```
Solved Sudoku Board:
3 5 7  | 1 6 2  | 4 9 8
1 6 2  | 8 4 9  | 7 3 5
4 9 8  | 7 3 5  | 1 2 6
- - - - - - - - - - - - -
7 4 9  | 2 1 8  | 6 5 3
8 2 5  | 6 7 3  | 9 4 1
6 1 3  | 9 5 4  | 8 7 2
- - - - - - - - - - - - -
5 8 1  | 3 9 7  | 2 6 4
2 7 4  | 5 8 6  | 3 1 9
9 3 6  | 4 2 1  | 5 8 7
```

2. Invalid Sudoku Grid

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

```
Original Sudoku Board:
5 1 6  | 8 4 9  | 7 3 2
3 0 7  | 6 0 5  | 0 0 0
8 0 9  | 7 0 0  | 0 6 5
- - - - - - - - - - - - -
1 3 5  | 0 6 0  | 9 8 7
4 7 2  | 5 9 0  | 0 1 0
9 6 8  | 3 7 1  | 0 0 0
- - - - - - - - - - - - -
2 0 3  | 1 8 6  | 5 7 9
6 8 4  | 9 5 7  | 3 2 1
7 9 1  | 2 3 0  | 6 5 8
```

```
No solution exists.
```