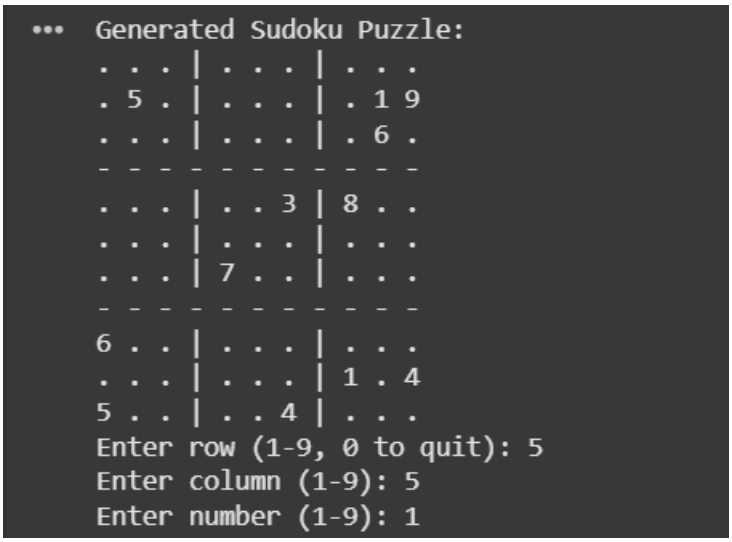
Sudoku Game Implementation in Python



Submitted by: Prajesh Singh Meena

Program : Btech Cse AIML

Roll No: 202401100400136

Library Id : 2428CSEAIML923

Date: 10 / 03 / 2025

Institution: KIET Group of Institutions

Introduction

Sudoku is an ever-popular logic-based number puzzle which involves putting numbers into a 9×9 grid so that each row, column, and 3×3 subgrid contains the numbers 1 to 9 without repeat. Sudoku calls for critical thinking and observation skills, hence an ideal exercise in problem-solving.

A Sudoku Solver Game is an application that not only enables people to play Sudoku but also offers an automated method of solving the puzzle through algorithms. It makes sure that the puzzle can be solved and assists players by giving them correct moves or solving the whole grid.

In this project, the Sudoku solver is programmed in Python based on a backtracking algorithm, a recursive technique that effectively identifies a solution by trying out possible values and backtracking when there is an invalid placement. The game also features an interactive interface where users can enter numbers, validate, and optionally have the puzzle solved automatically.

This project illustrates basic principles in programming, including recursion, constraint satisfaction problems, and user interaction, and is thus a great project for students interested in artificial intelligence, algorithms, and game development.

Methodology

Board Representation: The Sudoku board is represented as a 9×9 2D list with 0 representing the blank cells.

Puzzle Generation: A random Sudoku puzzle is generated by populating at least 17 pre-filled numbers such that there exists a unique solution.

Move Validation: Before establishing a number, the application checks for:

Row Constraint: The value cannot be duplicated within the row.
Column Constraint: The number cannot repeat in the column.
Subgrid Constraint: It can't be reused within the 3×3 subgrid.
Sudoku Solver (Backtracking Algorithm):

Find an empty cell and place numbers 1-9.
If the number is not valid, leave the number as empty and backtrack.
If none of the numbers work, retrace and use another number. Continue repeating until the entire board is finished.

Displaying the Board: The board is displayed in formatted fashion with | and - separators for readability.

User Interaction: User inputs row, column, and number to fill the Sudoku grid. Illegal moves show an error message. Game Execution: The program runs by generating a new puzzle, taking user input, and providing a facility to solve the puzzle automatically.

Code

```
import numpy as np
import random

# Function to print the Sudoku board in a readable format
def print_sudoku(board):
    for i in range(9):
        if i % 3 == 0 and i != 0:
            print("- - - - - - - - - - -")  # Print horizontal separators
        for j in range(9):
            if j % 3 == 0 and j != 0:
                print("|", end=" ")  # Print vertical separators
            print(board[i][j] if board[i][j] != 0 else '.', end=" ")  # Print numbers or dots for empty cells
        print()

# Function to check if placing a number in a given cell is valid
def is_valid(board, row, col, num):
    # Check if the number exists in the row or column
    for i in range(9):
        if board[row][i] == num or board[i][col] == num:
            return False

    # Check if the number exists in the 3x3 sub-grid
    box_x, box_y = (row // 3) * 3, (col // 3) * 3
    for i in range(3):
        for j in range(3):
            if board[box_x + i][box_y + j] == num:
                return False
    return True

# Function to solve the Sudoku puzzle using backtracking
def solve_sudoku(board):
    for i in range(9):
```

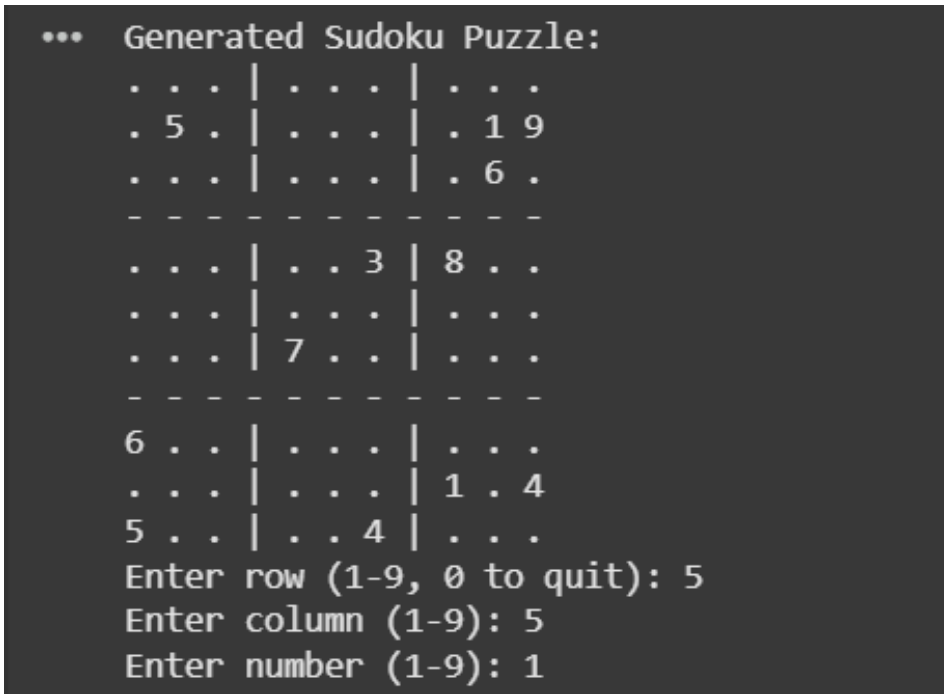
```
        for j in range(9):
            if board[i][j] == 0:  # Find an empty cell
                for num in range(1, 10):  # Try numbers 1 to 9
                    if is_valid(board, i, j, num):
                        board[i][j] = num
                        if solve_sudoku(board):  # Recursively attempt to solve
                            return True
                        board[i][j] = 0  # Undo move if it leads to failure
                return False
    return True  # Puzzle solved

# Function to generate a random Sudoku puzzle
def generate_sudoku():
    board = np.zeros((9, 9), dtype=int)  # Initialize empty board
    for _ in range(17):  # Ensure minimum numbers for a solvable puzzle
        row, col = random.randint(0, 8), random.randint(0, 8)
        num = random.randint(1, 9)
        if is_valid(board, row, col, num):
            board[row][col] = num  # Place a valid number
    return board

# Function to allow the user to play Sudoku
def play_sudoku():
    board = generate_sudoku()  # Generate a new puzzle
    print("Generated Sudoku Puzzle:")
    print_sudoku(board)

    while True:
        try:
            row = int(input("Enter row (1-9, 0 to quit): ")) - 1  # Get user input
            if row == -1:
                break  # Exit game if user enters 0
            col = int(input("Enter column (1-9): ")) - 1
            num = int(input("Enter number (1-9): "))
```

```
            if board[row][col] == 0 and is_valid(board, row, col, num):
                board[row][col] = num  # Place the number if valid
            else:
                print("Invalid move. Try again.")

            print_sudoku(board)  # Print updated board
        except ValueError:
            print("Please enter valid numbers.")  # Handle non-numeric input
        except IndexError:
            print("Row and column must be between 1 and 9.")  # Handle out-of-range input

    print("Game Over!")  # End message

if __name__ == "__main__":
    play_sudoku()
```

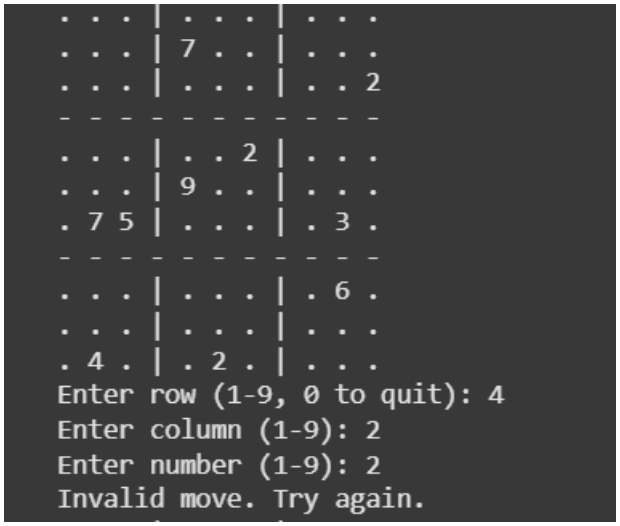
Output

The program is successful in creating an arbitrarily generated Sudoku puzzle with filled numbers.
The user is able to input values in order to solve the puzzle step-by-step.
The program verifies moves for legality based on Sudoku rules.
Sudoku backtracking algorithm solver is able to solve the puzzle automatically upon being called.
The final product shows a filled-out Sudoku grid in its entirety, either fully hand-written by the user or via a computer.

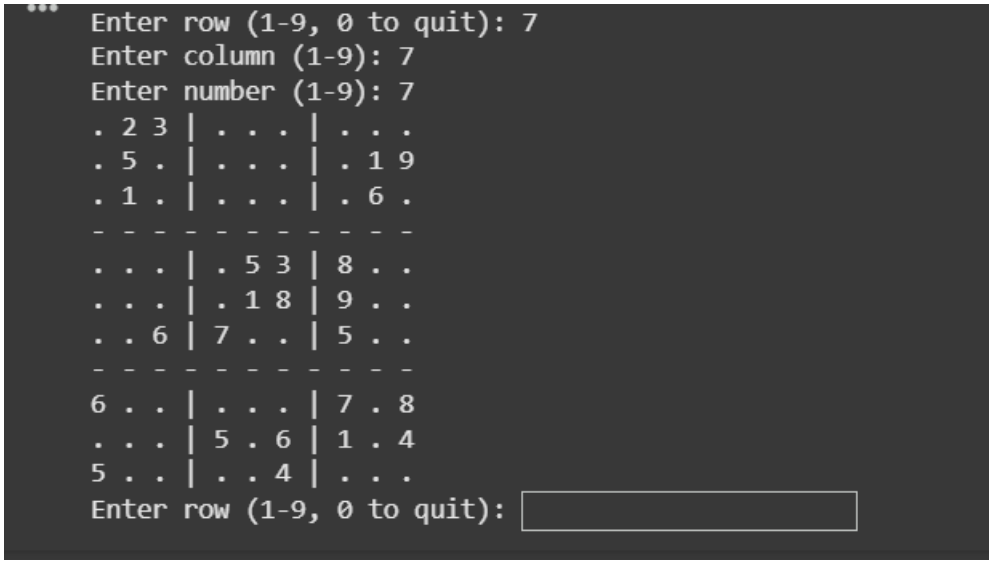
At first it takes row number, column number and value from the user:



Make sure to not repeat the number in same row, column or the sub grid of 3 x 3 :



And keep filling the 9 x 9 order complete grid so that you can complete the sudoku game :



References/Credits

Python documentation: https://docs.python.org/.

NumPy library: https://numpy.org/

Online programming resources' Sudoku-solving algorithms.

This project was created as a learning project in game design and Python programming using logic. Great appreciation to online resources and forums for assistance.