REPORT

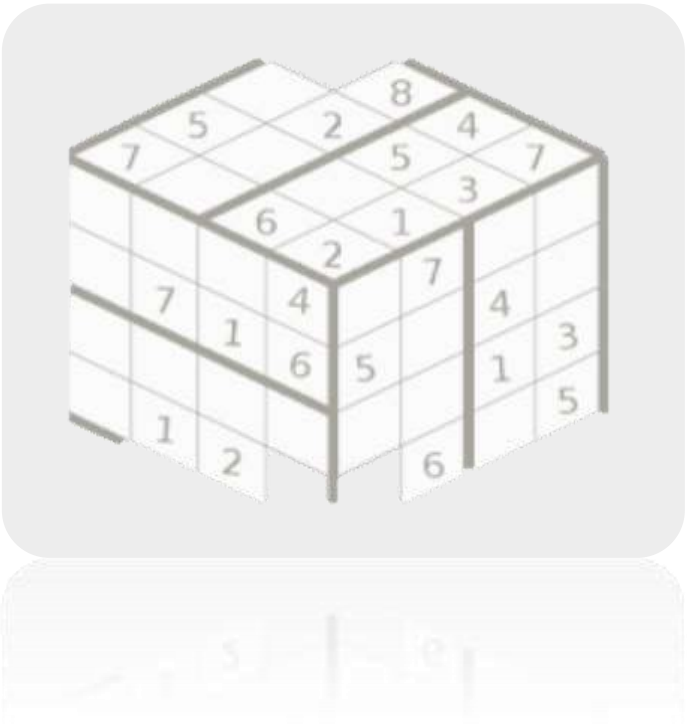
Name – Megha Thakur

Section – 9SK01

Roll No – R9SK01A16

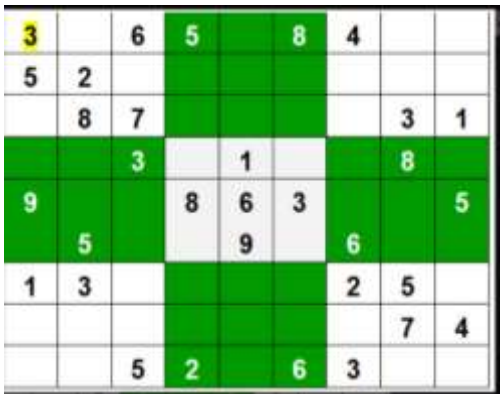
Registration Number - 12211812

Project - Sudoku solver visualizer



Introduction of Sudoku Solver

Sudoku is a popular logic-based, combinatorial number-placement puzzle. The objective is to fill a 9x9 grid with digits so that each column, each row, and each of the nine 3x3 subgrids (also called "boxes") contain all of the digits from 1 to 9. The puzzle starts with some cells already filled with numbers, and the challenge is to fill the empty cells according to the aforementioned rules.



Input: A partially filled 9x9 grid of numbers, where empty cells are represented by zero or left blank.

Output: A fully filled 9x9 grid that satisfies the conditions:

- Each row contains the digits 1 to 9 without repetition.
- Each column contains the digits 1 to 9 without repetition.
- Each 3x3 subgrid contains the digits 1 to 9 without repetition.

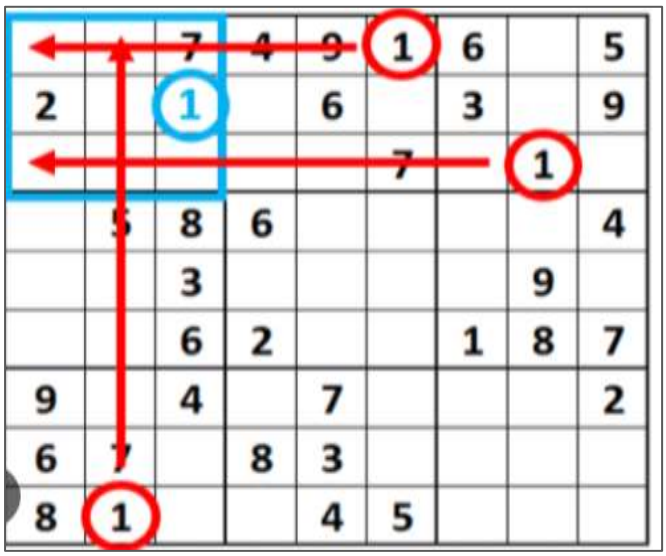
Naive Approach:

The naive approach is to generate all possible configurations of numbers from 1 to 9 to fill the empty cells. Try every configuration one by one until the correct configuration is found, i.e. for every unassigned position fill the position with a number from 1 to 9. After filling all the unassigned positions check if the matrix is safe or not. If safe print else recurs for other cases.

Sudoku using Backtracking:

Like all other Backtracking problems, Sudoku can be solved by assigning numbers one by one to empty cells. Before assigning a number, check whether it is safe to assign.

Check that the same number is not present in the current row, current column and current 3X3 subgrid. After checking for safety, assign the number, and recursively check whether this assignment leads to a solution or not. If the assignment doesn't lead to a solution, then try the next number for the current empty cell. And if none of the number (1 to 9) leads to a solution, return false and print no solution exists.



Key Challenges

1. **Validation:** Ensure that the initial configuration of the puzzle is valid according to Sudoku rules before attempting to solve it.
2. **Backtracking Algorithm:** Implement an efficient backtracking algorithm to explore possible solutions recursively. The algorithm must:
    - Place a number in an empty cell.
    - Recursively attempt to fill the remaining cells.
    - Backtrack if placing a number leads to an invalid configuration.
3. **Performance:** The algorithm should be optimized to solve puzzles in a reasonable amount of time. This involves:
    - Pruning the search space by rejecting invalid configurations early.
    - Using heuristics to decide the order of cell filling and number placement.
4. **User Interaction:** Provide a user-friendly interface where users can:
    - Input their own Sudoku puzzles.
    - Automatically solve the puzzles.
    - See the solving process in action through visual feedback.

Solving Approach

The Sudoku Solver application uses the following approach:

1. **User Input:** Users can input their Sudoku puzzles into a 9x9 grid of text fields.
2. **Validation:** The application checks the input for any immediate rule violations.
3. **Backtracking Algorithm:** The solver uses a backtracking algorithm to try filling the grid. This involves:
    - Trying each number (1-9) in each empty cell.
    - Checking if placing a number violates any Sudoku rules.
    - Recursively attempting to fill the next cell.
    - Backtracking if no valid number can be placed in the current cell.
4. **Visual Feedback:** During the solving process, the application updates the grid to show the progress, with different colors indicating numbers being inserted or removed.
5. **Random Filling:** For users who want to start with a partially filled puzzle, the application can randomly fill some cells with valid numbers, ensuring the puzzle remains solvable.

1. Class Definition and GUI Setup

- **Fields:** cells is a 9x9 grid of JTextField objects representing the Sudoku board. solveButton, clearButton, and fillRandomButton are buttons for user interaction, and messageLabel displays messages.
- **Constructor:** Sets up the main frame, grid panel, control panel, and their layouts. It also initializes the cells with appropriate fonts and colors for the 3x3 subgrids, and adds action listeners to the buttons.

2. Button Listeners

- **Action:** Disables buttons, checks input validity, and attempts to solve the Sudoku puzzle in a background thread using SwingWorker.
- **Background Task:** Parses the input, validates it, and calls the solving function. Updates the GUI with the result once done.

Clear Button

Action: Clears all cells and resets their background colors.

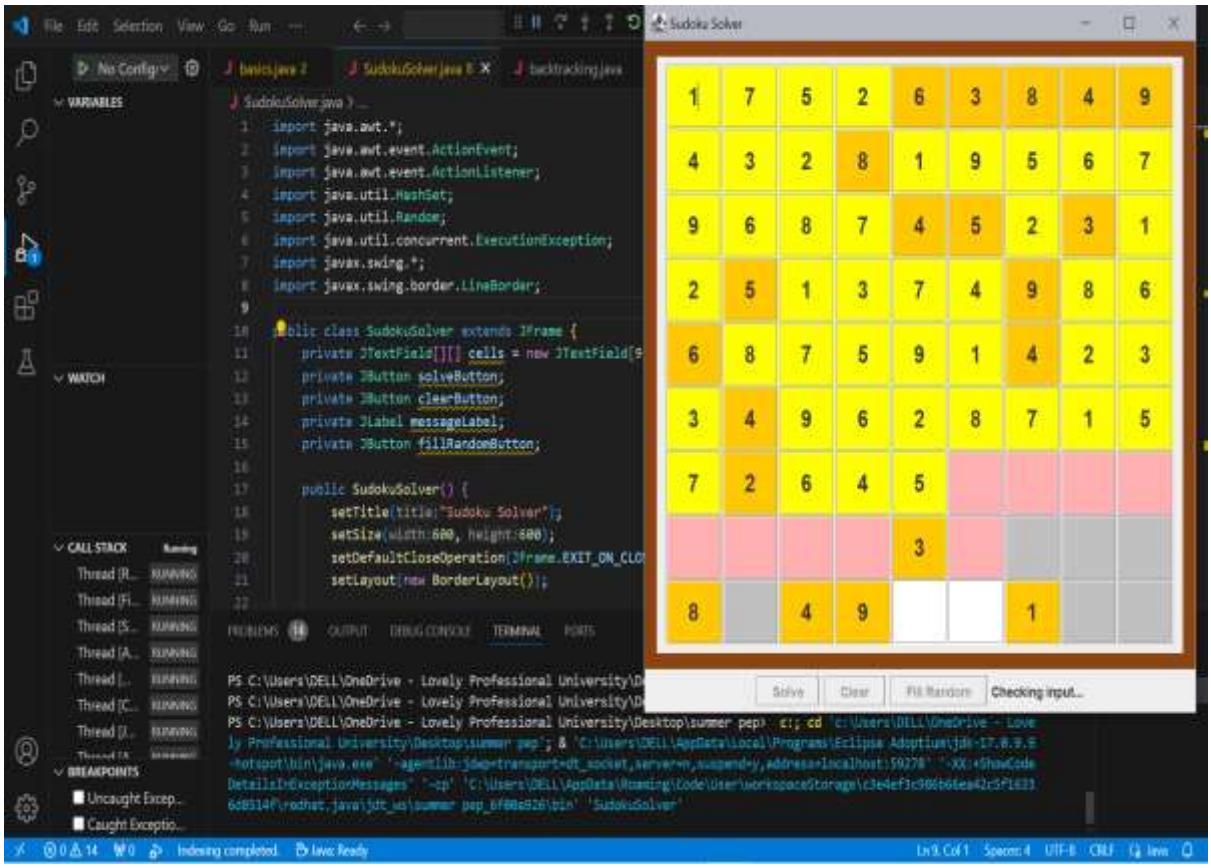
Fill Random Button

Action: Clears the board and fills it with random valid numbers, highlighting the filled cells.

3. Helper Methods

- **clearBoard:** Clears the board and resets cell colors.
- **fillRandomValues:** Fills 20 random positions with valid numbers, highlighting them.
- **solveSudoku:** Uses backtracking to solve the Sudoku puzzle. Highlights cells as numbers are inserted and removed.
- **updateBoard:** Updates the GUI with the current state of the board.
- **isSafe:** Checks if placing a number at a specific position is valid.

Sudoku solver visualizer



By clicking on the Fill Random button , the values from 1 to 9 fill at any random places(20 random values) .

When we will click on the Solve button, the values starts filling as that same value should not be in the same row and column.

Clear Button, clear all the number and gives the empty board.