Problem Understanding

Study the rules of Sudoku: every number from 1 to 9 must appear exactly once in each row, column, and 3×3 grid.

Understand how solving Sudoku can be treated as a constraint satisfaction problem, best solved using a backtracking algorithm.

2. Algorithm Design

Use backtracking to build the solver:

Find an empty cell.

Try placing numbers from 1 to 9 in that cell.

Check if the placement is valid according to Sudoku rules.

If valid, proceed recursively; if not, backtrack and try the next number.

Continue until the board is solved or no solution is possible.

3. GUI Development with Tkinter

Design a 9x9 grid of input fields using Entry widgets in Tkinter.

Create labels, buttons (e.g., "Solve", "Clear"), and greeting messages.

Use styling to create a lavender background and custom fonts.

4. User Input Handling

Allow users to enter Sudoku puzzles manually.

Validate inputs to accept only numbers 1–9 or empty cells.

5. Solving the Puzzle

On clicking the "Solve" button:

Extract the current board from the GUI.

Pass it to the backtracking solver.

Display the result back on the grid.

6. Error Handling and Feedback

Check for incorrect entries (like duplicate values in rows or columns).

Highlight mistakes using red backgrounds and show a warning message.

Remove the warning message once the user clicks OK.

7. Visual Effects (Enhancement)

Add animations like:

"Paper to dust away" effect after solving.

"Glitter pop" or confetti effect when the solution is successfully shown.

8. Testing

Test the solver with:

Easy, medium, and hard Sudoku puzzles.

Puzzles with multiple or no solutions.

Empty grids and invalid inputs.

9. Final Touches

Add a "Welcome + name" greeting.

Improve layout spacing and visual appearance.

Ensure the app is responsive and stable.