SUDOKU SOLVER

1. Overview

1.1 Description

This project aims to develop a Sudoku solver program that allows efficient entry of Sudoku puzzles via GUI or text file, implements various solving strategies, benchmarks their performance, evaluates success rates, and provides the solved puzzle for further use.

1.2 Goals

- Develop a Sudoku solver with efficient puzzle entry methods
- Implement basic, randomized, and advanced solving strategies.
- Incorporate complex solving strategies in different sequences based on efficiency.
- Benchmark and compare the success rates and solving times of each strategy.
- Output the solved Sudoku puzzle for use in other applications.

2. Technologies Used

- **Programming Language**: C
- **GUI Framework**: Text GUI
- **Tools**: Visual Studio Code, GCC, GitHub for version control

3. Plan the Timeline

BY	DESCRIPTION	COMPLETE?	DATE COMPLETED
07.11	Break Down Code into sizable chunks	YES	07.11
07.13	Step 1 Complete (Text GUI)	YES	07.12
07.14	Step 2 Complete (Solver - Basic)	YES	07.13
07.18	Project Completion and Testing	YES	07.20
07.20	Write a Conclusion	YES	07.20

4. Design the Architecture

Variables

- PUZZLE_SIZE: sudoku height and width - 9
- **puzzle[PUZZLE_SIZE][PUZZLE_SIZE]**: 2-dimensional array for Sudoku grid (numbers 1-9 or 0 for blank).

- **puzzle_notes[PUZZLE_SIZE][PUZZLE_SIZE][PUZZLE_SIZE] :** 3-dimensional array for puzzle notes

Methods

- **print_puzzle() Module**: print sudoku puzzle
- **is_puzzle_solved() Module**: check if the puzzle is solved.
- **strat1() Module**: basic take_notes() and if a square can only be 1 number fill it in
- **strat2() Module**: also basic but checks each row, column and block for numbers that are noted once and fills them in.
- **strat3() Module**: in progress, intended to use the "locked candidate" strategy to eliminate number canidates
- **take_notes() Module**: view a single square and eliminate candidates by looking at the square's row, column and block
- **print_notes() Module**: view notes taken.
- **Documentation**: README.md and documentation for strategies.

Main Logic

- Initialize Sudoku puzzle from intro text.
- Repeat solving process for each strategy:
    - Start timer.
    - Solve Sudoku using the chosen strategy.
    - Stop timer and record solving time.
- Display solutions and corresponding solving times.

5. Choose the Tech Stack

- **Language**: C (for performance and direct memory control).
- **IDE**: Visual Studio Code

6. Break Down Tasks

User Interactions

- **Introduction**: Display initial instructions and puzzle entry method.
- **Puzzle Entry**: Implement text-based GUI for puzzle entry and validation. (removed for ease of use)
- **Conclusion**: Display solved puzzles and corresponding solving times.

Timer

- **Start Timer**: Implement a function to start recording time.
- **Stop Timer**: Function to stop the timer and calculate elapsed time.
- **Save Time**: Store solving times for analysis and display.

Solvers

- **Basic Solver**: Implement a straightforward backtracking algorithm.
- **Complex Strategies**: Utilize strategies from "Sudoku Solving Guide.pdf" (e.g., Naked Singles, Hidden Singles, advanced techniques). (in progress will not be completed without more extensive time taken or different strategy taken)
    - Implement these strategies in different orders to determine optimal solving path.

7. Allocate Resources

- **GitHub Account**: Set up `andrewc272` GitHub repository.
- **Development Machine**: 16" MacBook Pro with Apple M2 Max for development and testing.

8. Conclusion

In this project I set out to solve a sudoku puzzle faster than I can. Timing myself, it takes me 9 minutes to solve an expert puzzle by myself. When I use the tool I can plug in the puzzle, solve it, and reenter the solution in about 2 minutes the main slow down being myself. Unfortunately, the solver can't solve puzzles that are more difficult than "Expert" as of right now. It is very simplistic, and C wasn't conducive to more complex strategies. This project demonstrates my expertise in using C for a task as mentioned. It shows the usefulness of C and its very low level and fast in a use case. Further development of this project would likely look like moving it into an object oriented programming language to help clean up code and make more complex strategies easier to implement and read.