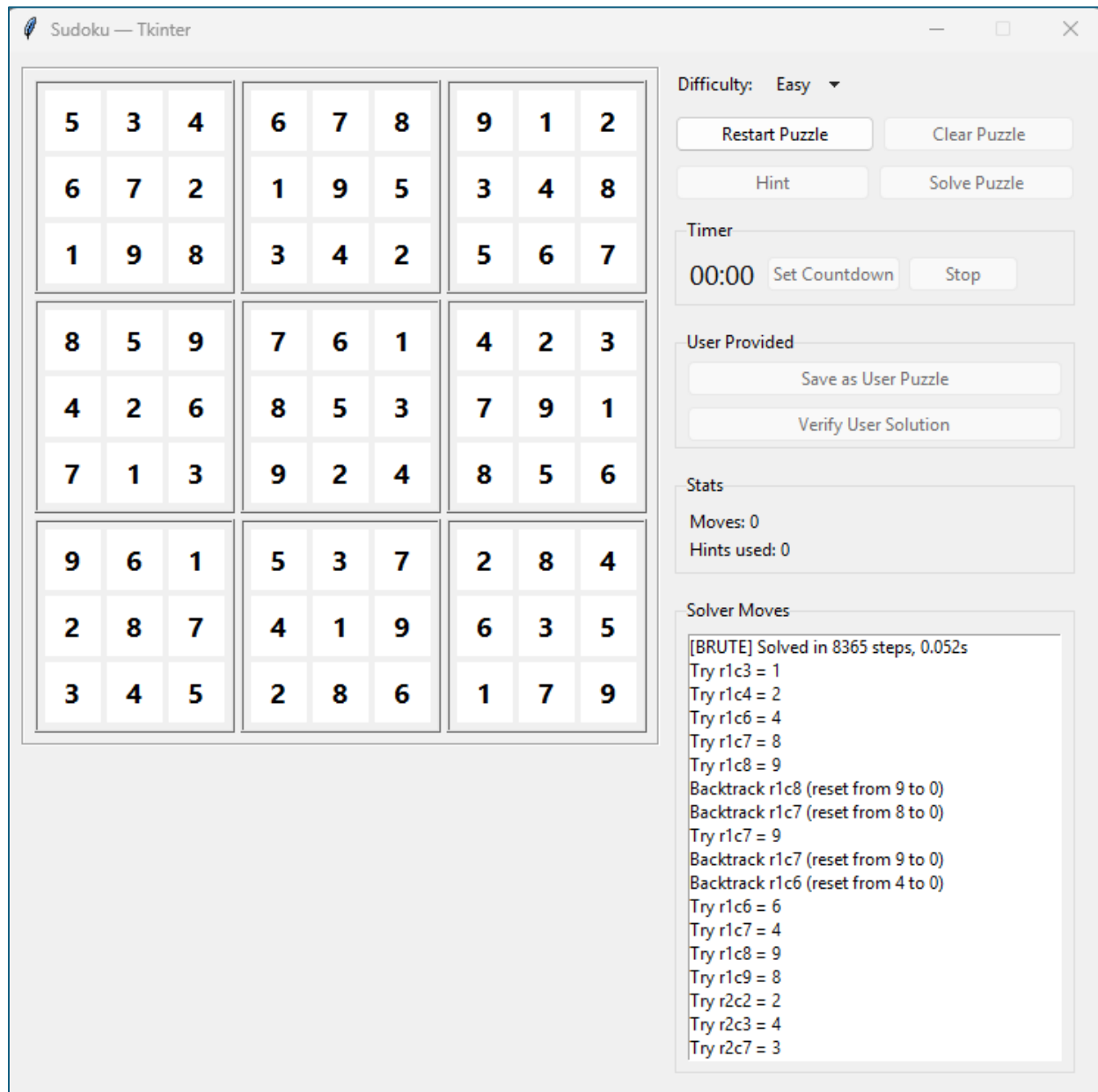


Sudoku Application

Capillo, Doria, Sunga

GUI Walkthrough



A. Left Side → Sudoku Grid (9×9)



- This is built in `_build_grid()`.
- Each cell is a Tkinter Entry widget linked to a `StringVar`.
- Black bold numbers = given puzzle clues (read-only, can't edit).

- d. Blue italic numbers (not seen here because you solved automatically) = user inputs.
- e. The grid is split into 3×3 sub-boxes with ridged borders, matching Sudoku layout.

B. Top Right → Difficulty & Controls

- a. **Difficulty dropdown** (Easy / Medium / Hard / User Provided):
 - i. Controlled by `self.difficulty_var`.
 - ii. When changed, `_on_difficulty_change()` shows a reminder to restart.
- b. **Restart Puzzle button** → calls `_restart_puzzle()`, loads a new puzzle from the chosen difficulty.
- c. **Clear Puzzle button** → clears only non-given cells, letting you reset without starting over.

C. Middle Right → Gameplay Tools

- a. **Hint button** → highlights your entries:
 - i.  Yellow if correct (matches solution).
 - ii.  Red if wrong.
- b. **Solve Puzzle button** → triggers `_solve_puzzle_interactive()`.
 - i. Asks you to pick algorithm:
 1. Brute Force
 2. Backtracking + Propagation
 - ii. Then `_run_solver_and_render()` runs it and logs the solver's thought process.

D. Timer Box

- a. Shows countdown timer (default `00:00`).
- b. Buttons:
 - i. **Set Countdown** → opens dialog to set minutes/seconds.
 - ii. **Stop** → stops timer.
- c. If timer hits 0, `_on_time_up()` asks whether to continue or auto-solve.

E. User Provided Box

- a. Lets you **create your own puzzles**:
 - i. "Save as User Puzzle" → enter your puzzle and provide the solution.
 - ii. "Verify User Solution" → check if a user-provided solution matches solver results.

F. Stats Box

- a. Tracks gameplay stats:
 - i. **Moves**: increments every time you enter/change a number.
 - ii. **Hints used**: increments when you request hints.
- b. Here it shows Moves: 0, Hints used: 0 since the solver auto-filled the puzzle.

G. Solver Moves Log

- a. A **Listbox** that shows step-by-step reasoning from the solver

- b. Breakdown:
 - i. Try `r1c3 = 1` → Solver tries to place 1 at row 1, column 3.
 - ii. Backtrack `r1c8` (reset from 9 to 0) → It realized this path led to conflict, so it erased the guess and backtracked.
 - iii. The solver keeps trying and backtracking until the whole board is filled.
 - c. The first line `[BRUTE] Solved in 8365 steps, 0.052s` shows:
 - i. Algorithm used (`BRUTE`)
 - ii. Total trial steps (8365)
 - iii. Solve time (`0.052 seconds`).
-

Code Walkthrough

A. Imports and Type Definitions

```
from __future__ import annotations

import tkinter as tk
from tkinter import ttk, messagebox, simpledialog
from tkinter import font as tkfont
from typing import List, Tuple, Optional, Set
import random
import time

Board = List[List[int]]
```

- a. `tkinter` → Python's built-in GUI library.
- b. `ttk` → themed widgets (modern look).
- c. `messagebox` & `simpledialog` → popup dialogs.
- d. `tkfont` → font customization.
- e. `typing` → for type hints like `List`, `Optional`.
- f. `random` → pick puzzles randomly.
- g. `time` → measure solving time.
- h. `Board` is defined as a type alias for a 9x9 list of integers (our Sudoku grid).

B. Sudoku Logic (Pure Backend)

```
class SudokuLogic:
```

- a. This class contains **static methods** for solving Sudoku, separate from GUI. That way the solver can be reused anywhere.

b. Copying the Board

```
def deep_copy(board: Board) -> Board:
    return [row[:] for row in board]
```

- i. Creates a duplicate board to avoid changing the original.

c. Valid Move Check

```
def is_valid_move(board: Board, r: int, c: int, v: int) -> bool:
```

- i. Ensures that placing value `v` at `(r,c)` doesn't break Sudoku rules:
- ii. Not in the same row
- iii. Not in the same column
- iv. Not in the same 3×3 box

d. Finding Empty Cell

```
def find_empty(board: Board) -> Optional[Tuple[int,int]]:
```

- i. Searches row by row for the first `0` (empty cell). Returns `(row, col)` or `None` if full.

e. Candidate Values

```
def candidates(board: Board, r: int, c: int) -> Set[int]:
```

- i. Finds all numbers `1–9` that could legally go into `(r,c)` by excluding row, column, and box values.

f. Brute Force Solver

```
def solve_bruteforce(board: Board, moves: List[str]) -> Optional[Board]:
```

- i. Backtracking algorithm:
 - 1. Find an empty cell.
 - 2. Try numbers `1–9`.
 - 3. If valid, recurse.
 - 4. If stuck, backtrack.
- ii. `moves` list logs every attempt for display.

g. Constraint Propagation (Singles)

```
def propagate_singles(board: Board, moves: List[str]) -> bool:
```

- i. Automatically fills cells that have only one candidate (“singletons”).
- ii. Repeats until no more can be filled.

h. Backtracking + Propagation (Smarter Solver)

```
def solve_backtracking_propagation(board: Board, moves: List[str]) -> Optional[Board]:
```

- i. Uses MRV heuristic (Minimum Remaining Values):
 - 1. pick the cell with the fewest candidates to reduce branching.
- ii. Combines backtracking with `propagate_singles` for efficiency.

i. Final Check (Solved & Valid)

```
def is_complete_and_valid(board: Board) -> bool:
```

- i. Confirms each row, column, and 3×3 box contains 1–9 exactly once.

C. Puzzle Datasets

```
EASY_PUZZLES, MEDIUM_PUZZLES, HARD_PUZZLES = [...]
```

- a. Pre-loaded Sudoku puzzles of increasing difficulty (lists of 9×9 boards).

D. GUI Class — SudokuApp

```
class SudokuApp(tk.Tk):
```

Subclass of Tkinter's Tk → main window.

a. Initialization

```
def __init__(self) -> None:
    super().__init__()
    self.title("Sudoku - Tkinter")
    self.resizable(False, False)
```

- i. Sets up window title & size.
- ii. Initializes state variables:
 - 1. board_vars → Tkinter StringVars linked to cell entries
 - 2. entries → Entry widgets for each cell
 - 3. given_mask → track which cells are fixed
 - 4. current_puzzle & current_solution → puzzle and solution
 - 5. moves_count, hints_used → stats
 - 6. timer variables for countdown
- iii. Then it builds the layout and loads an initial puzzle:

```
self._build_layout()
self._restart_puzzle()
```

E. Building the Layout

```
def _build_layout(self) -> None:
```

Creates the UI layout:

- **Main frame** → container for grid + side panel.
- **Grid (9×9)** built by `_build_grid()`.
- **Side panel** →
 - Difficulty menu (Easy/Medium/Hard/User)
 - Buttons (Restart, Clear, Hint, Solve)

- Timer (countdown set & stop)
- User puzzle options (Save/Verify)
- Stats (moves, hints used)
- Solver moves log (Listbox showing steps).

F. Sudoku Grid (9×9)

```
def _build_grid(self, parent: tk.Widget) -> None:
```

- Divides grid into 3×3 sub-boxes with borders.
- Inside each box, places 3×3 Entry widgets (total 81 cells).
- Each Entry is linked to a `StringVar`.
- Validation ensures only digits 1–9 allowed.
- Hover effects highlight row, column, and box.

G. Core Board Operations

- `_get_board` → reads Entry values into a 9×9 integer board.
- `_set_board` → fills Entries with a given board (read-only if given cell).
- `_clear_puzzle` → removes non-given values.
- `_hint` → compares current state with solution and colors:
 - Yellow = correct so far
 - Red = incorrect entry.

H. Solving Interaction

- `_solve_puzzle_interactive` → asks user if they want to reveal solution, then asks algorithm (Brute vs Propagation).
- `_run_solver_and_render` → runs chosen solver, times it, logs steps in listbox, updates board.

I. Timer Functions

- `_set_timer_dialog` → ask user minutes/seconds.
- `_start_timer` → starts countdown.
- `_tick` → updates every second, stops at 0.
- `_on_time_up` → when timer ends, either continue or auto-solve.

J. User Puzzle Features

- `_save_user_puzzle` → lets player enter their own puzzle + solution.
- `_verify_user_solution` → check if provided solution matches solver's solution.

K. Running the App

```
if __name__ == "__main__":
    app = SudokuApp()
    app.mainloop()
```