Vamshi Jandhyala

Posidoku

✠

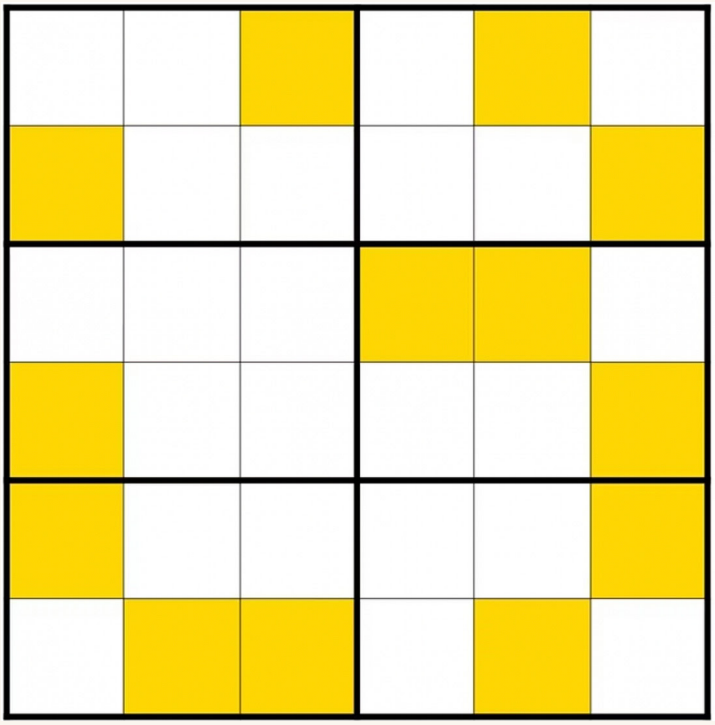
February 2025

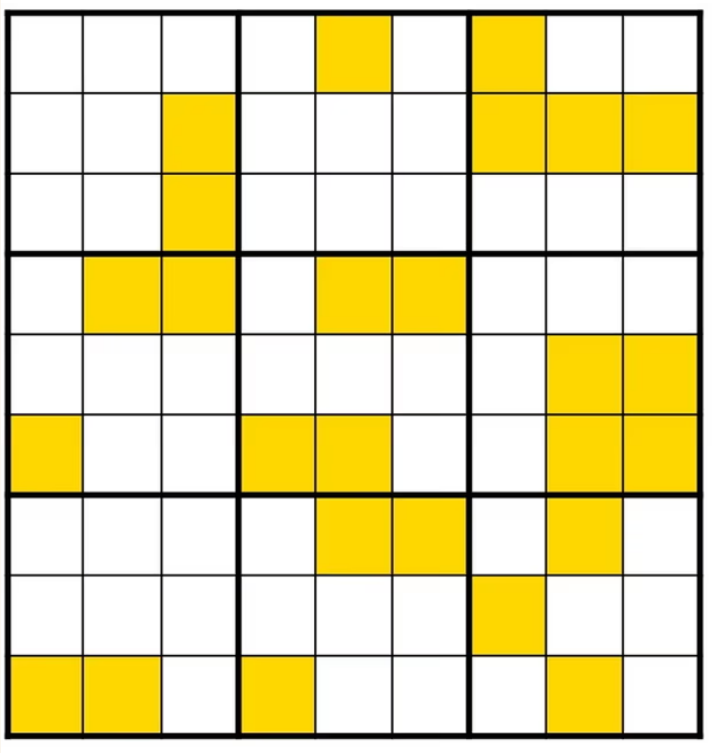
A Sudoku variant invented by Alf Smith.

Problem

Posidoku is a Sudoku variant in which the starting grids carry no number clues at all. Instead, certain cells are shaded gold; the digit placed in each gold cell must equal the cell's position in either its row, its column, or its box (with box positions read left-to-right, then top-to-bottom). White (non-gold) cells must *not* take their row, column, or box position as their value. As in standard Sudoku, the digits $1, \ldots, n$ must each appear exactly once in every row, column, and box.

Two example puzzles are given below.





Solution by constraint programming

The constraints are mixed: standard Latin-square row, column, and box uniqueness, plus an exclusive-or constraint at every gold cell (it must equal exactly one of three derived values), plus a not-equal constraint at every white cell. Google's `CP-SAT` handles the lot directly.

6 × 6 *solution*

Gold cells marked with ∗.

```
| 2  4  3* | 6  1* 5  |
| 1* 6  5  | 3  4  2* |
| 6  5  2  | 4* 3* 1  |
| 4* 3  1  | 5  2  6* |
| 5* 1  4  | 2  6  3* |
| 3  2* 6* | 1  5* 4  |
```

9 × 9 *solution*

```
| 4  7  8  | 5  2* 9  | 1* 3  6  |
| 9  1  6* | 3  4  8  | 7* 5* 2* |
| 2  5  3* | 6  1  7  | 9  4  8  |
| 8  2* 4* | 9  5* 6* | 3  1  7  |
| 6  9  7  | 1  3  4  | 2  8* 5* |
| 1* 3  5  | 7* 8* 2  | 4  6* 9* |
| 5  4  9  | 8  7* 3* | 6  2* 1  |
```

```
| 3  6  1  | 2  9  5  | 8* 7  4  |
| 7* 8* 2  | 4* 6  1  | 5  9* 3  |
```

Python code

```python
from ortools.sat.python import cp_model

def solve_grid_puzzle(size=6, gold_cells=None):
    model = cp_model.CpModel()

    if size == 6:
        box_rows, box_cols = 2, 3
    elif size == 9:
        box_rows, box_cols = 3, 3
    else:
        raise ValueError("Size must be 6 or 9")

    num_boxes = (size // box_rows) * (size // box_cols)
    cells = {(r, c): model.NewIntVar(1, size, f'cell_{r}_{c}')
             for r in range(size) for c in range(size)}

    def get_box_index(r, c):
        return (r // box_rows) * (size // box_cols) + (c // box_cols)

    def get_box_position(r, c):
        return (r % box_rows) * box_cols + (c % box_cols) + 1

    for r in range(size):
        model.AddAllDifferent([cells[(r, c)] for c in range(size)])
    for c in range(size):
        model.AddAllDifferent([cells[(r, c)] for r in range(size)])
    for box in range(num_boxes):
        model.AddAllDifferent([cells[(r, c)]
            for r in range(size) for c in range(size)
            if get_box_index(r, c) == box])

    for r in range(size):
        for c in range(size):
            if (r, c) in gold_cells:
                opts = []
                for v in {r + 1, c + 1, get_box_position(r, c)}:
                    if 1 <= v <= size:
                        b = model.NewBoolVar(f'opt_{r}_{c}_{v}')
                        model.Add(cells[(r, c)] == v).OnlyEnforceIf(b)
                        opts.append(b)
                model.AddExactlyOne(opts)
            else:
                for v in {r + 1, c + 1, get_box_position(r, c)}:
                    if 1 <= v <= size:
                        model.Add(cells[(r, c)] != v)

    solver = cp_model.CpSolver()
    return solver.Solve(model), solver, cells
```