# Sudoku Solver Architecture Document

## Revision History

| Date | Version | Description | Author |
|---|---|---|---|
| 2024-04-07 | 1.0 | Initial document | Boren Zang |

## Overview

This document outlines the design and implementation decisions for the Sudoku Solver, a Java-based application that uses a backtracking algorithm to solve Sudoku puzzles. The solver is designed to be efficient, maintainable, and easily extendable.

## Design Choices

## Data Structure: 2D Integer Array

The Sudoku grid is represented as a 2D integer array (`int[][]`), mirroring the puzzle's structure. This choice allows for straightforward access and manipulation of cells within the grid, essential for both solving the puzzle and validating the solution.

## Algorithm: Backtracking

The backtracking algorithm was selected for its suitability to the problem domain. Sudoku solving requires exploring different number placements in a depth-first manner, with the ability to backtrack upon reaching an invalid state. This algorithm effectively navigates the solution space of a Sudoku puzzle.

## Implementation Details

## Modularity and Separation of Concerns

The application is divided into methods that encapsulate specific functionalities:

- `isValidPlacement()`: Checks if placing a number in a given cell is valid according to Sudoku rules.
- `isNumberInBox()`: Checks if a number is present in a 3x3 subgrid.
- `isNumberInColumn`: Checks if a number is present in a column.
- `isNumberInRow`: Checks if a number is present in a row.
- `isValidSudoku()`: Checks if the solution of sudoku is correct.
- `solveBoard()`: Recursively attempts to fill the grid, backtracking when necessary.
- `printBoard()`: Displays the current state of the Sudoku grid.

This modular design enhances readability and maintainability, allowing for isolated testing and modification of individual components.

## Recursive Solution Approach

The `solveBoard()` method implements the backtracking algorithm through recursion. This approach naturally fits the problem, allowing the solver to progress depth-first across the solution space and backtrack when it encounters a dead-end.

## Validation Checks

To ensure the validity of number placements, the solver performs comprehensive checks for each number it attempts to place:

- Row uniqueness
- Column uniqueness
- Box (3x3 subgrid) uniqueness

These checks are crucial for the solver's correctness, ensuring that each step adheres to Sudoku's rules.

## Challenges and Solutions

## Handling Backtracking

Implementing efficient backtracking posed a challenge, particularly in managing the state of the grid when backtracking from an invalid placement. This was addressed by carefully resetting cells to their initial state (empty, denoted by `0`) upon backtracking, ensuring the solver's state remains consistent.

## Setup Instructions

## Requirements

- Java Development Kit (JDK) installed on your machine.
- An Integrated Development Environment (IDE) like IntelliJ IDEA for development, or a terminal for compilation and execution.

## Compiling and Running the Application

Using the Command Line

**Compilation & Execution**:

- Navigate to the `src` directory.
- Run command `chmod -x runSudokuSolver.sh` to make the bash file executable
- Run the `runSudokuSolver.sh` bash script to compile and execute the application

Using IntelliJ IDEA

**Running the Application**:

- Right-click within the code editor and select "Run 'SudokuSolver.main()'". Alternatively, use the run button in the toolbar.