# Sudoku Solvers Design Document

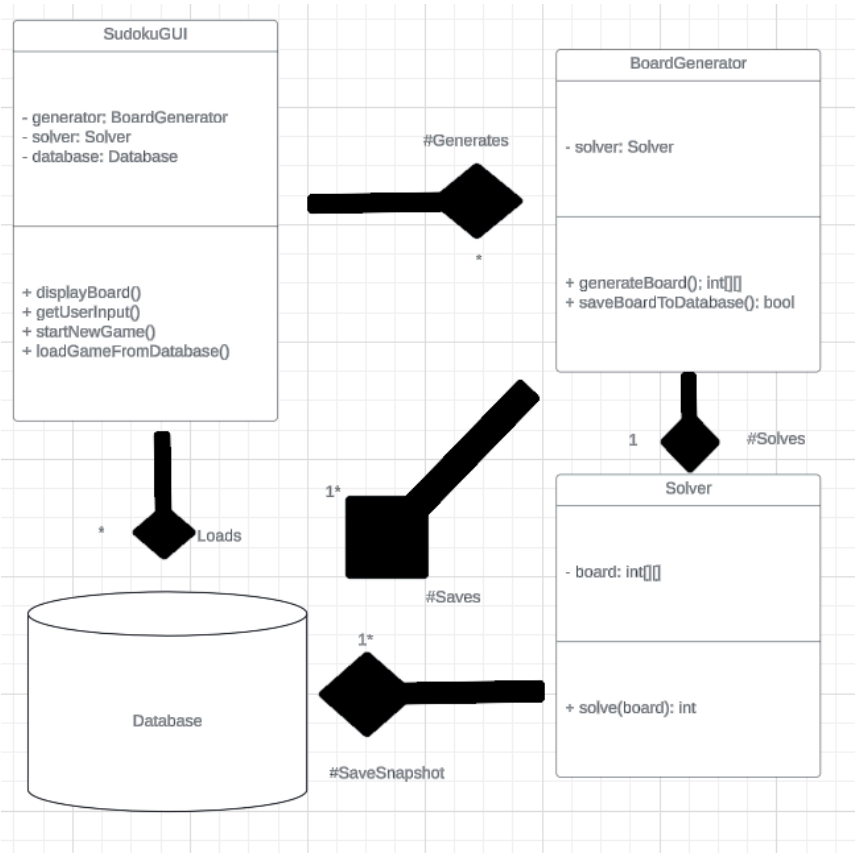
## 1. Introduction

### 1.1 Purpose & Scope

This document will define the architecture, external tools, and UI design for the Sudoku Solvers application. The structure of the application will reference the requirements given in the Requirement Document.

## 2. System Architecture

### 2.1 UML



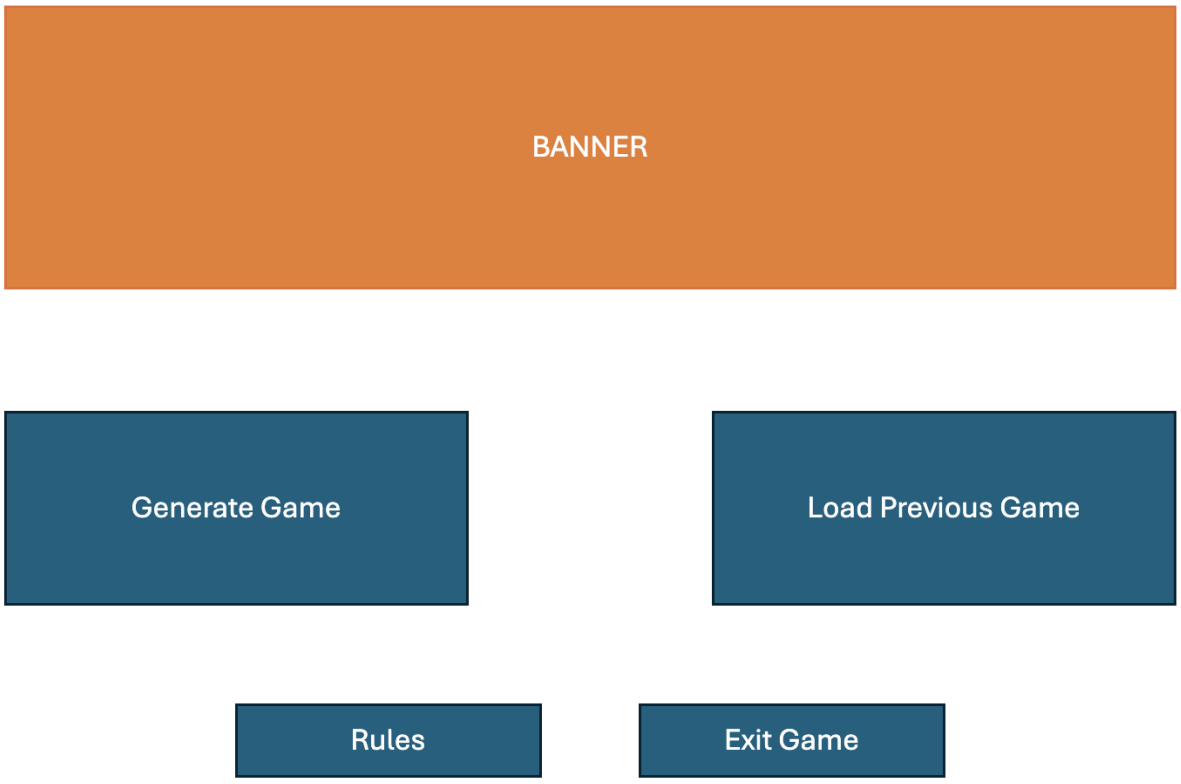
## 2.2 System Architecture

The main class is the SudokuGUI which creates instances of the Board Generator and Solver. The Board Generator interacts with the Solver to verify uniqueness, and as a result no other class needs to work with the Solver. The Solver will save the snapshot of the generating for future visual use, and therefore the GUI instead only has to work with the database to get the necessary information. When the solver verifies the uniqueness of the board, the Board Generator will save this board into the database. The database will be used to prevent lots of repeat work and will store generated boards, snapshots of generation process for visualization, and previous boards for loading history.

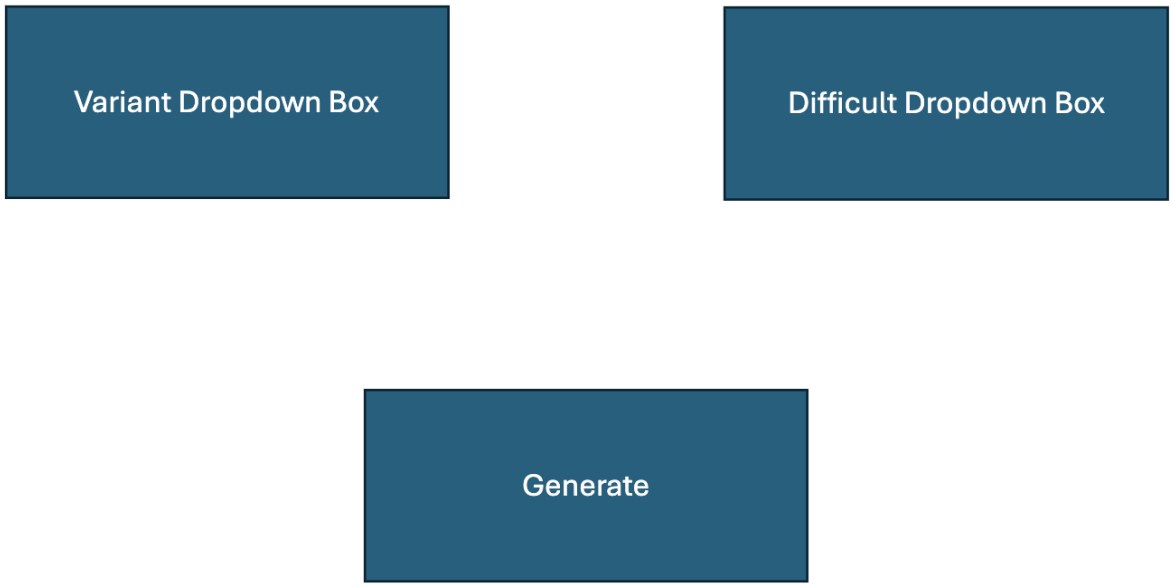
# 3. Detailed Design

## 3.1 User Interface

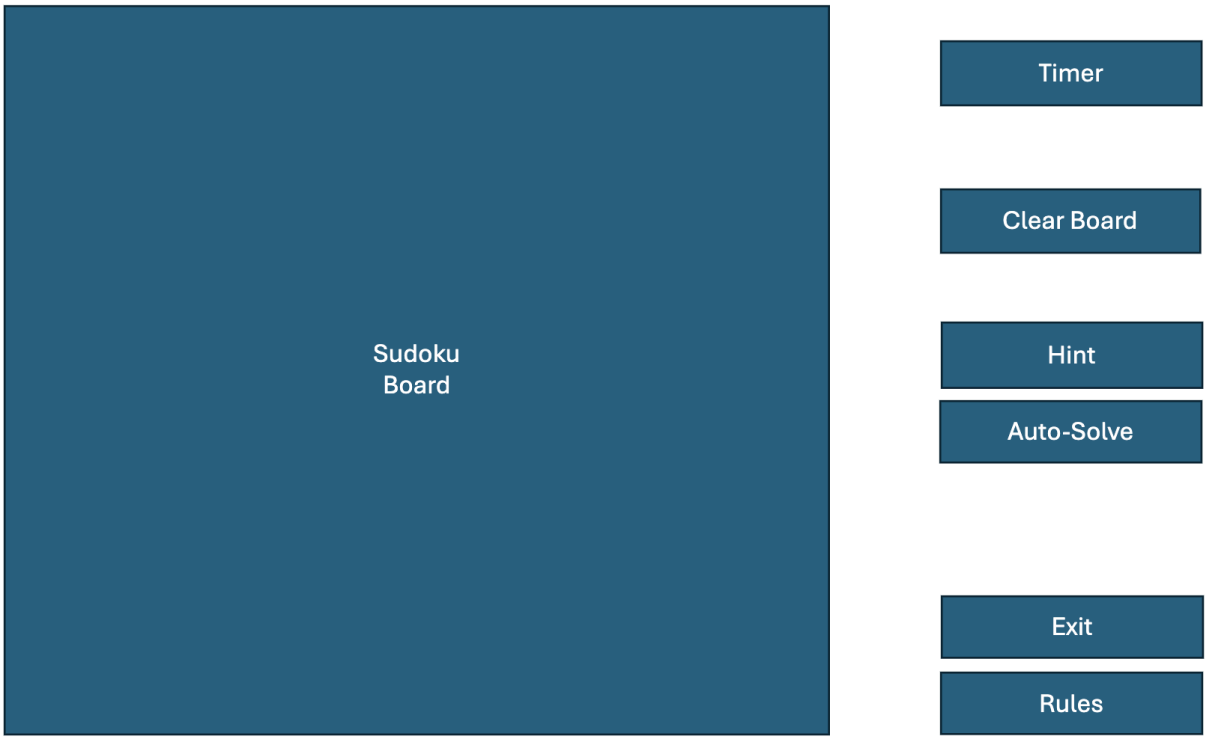
The starting screen will include a banner with a graphic or text symbolizing Sudoku. Two options will be given below to generate a new board or load a previous board from history. A rules button will be given that will redirect to a page of Sudoku rules.



In the Generate Game screen, the difficulty box dropdown will have three options: easy, medium, and hard. Difficulty will be determined by the number of boxes already filled in the board. An easier board will consist of more numbers and a harder board will consist of less numbers filled in after generation. The variant dropdown box will include all variants implemented for the current version of the application. The generate button will begin generation for the board with given settings and have a final board ready to play within three seconds.




The game interface will consist of the large Sudoku Board on the left side of the GUI. The board will have compatibility with arrow keys to allow easy navigation throughout the board without the necessity of using a mouse. A timer will be emplaced in the top right that begins automatically upon generation of the board. The clear button will wipe the board of all numbers inputted by the user. When the button is clicked another box will pop up asking the user to confirm they want to clear the board. The hint button will fill in the answer to the currently selected box. If no box is selected when the hint button is clicked, a random box will be filled in. The solve puzzle button will have a confirmation when clicked and will visually auto solve the puzzle in front of the user, showing the user the backtracking algorithm working. The exit button will open a small pop up window asking if the user would like to save the progress of the game started. The user's games will be auto saved after every entry or deletion to numbers on the Sudoku board, so if the application is exited through means other than the exit button, their Sudoku board will still be saved. The GUI will be powered by the library PyQt and will run locally on the user's machine.




## 3.2 Puzzle Generation

A valid puzzle for Sudoku must only have one solution and the difficulty of the generated puzzle will be calculated by the number of boxes initially given to the user. The puzzle generation algorithm will begin with a board completely filled out with numbers in compliance with Sudoku rules. The puzzle generator will remove a set number of boxes depending on the difficulty and then verify the uniqueness of the board through the Puzzle Solver algorithm, exhausting all options. If a second solution is found, either another number will be added and the process of verifying uniqueness will repeat, or puzzle generation will start at the beginning with a completely new starting set of numbers depending on confines to meet the required difficulty level.

## 3.3 Puzzle Solver

The Puzzle Solver will use a backtracking algorithm created by Knuth called Algorithm X, or Dancing Links, to iterate efficiently through possible combinations on the sudoku board. The puzzle solver will be used primarily to ensure uniqueness in Sudoku boards. Once a board has ensured to have a unique solution, this solution will be stored to limit unnecessary calls to the Puzzle Solver. As a result, when the user clicks the Auto-Solve button, it will not make a call to the Puzzle Solver but instead use stored information to improve response time to the user.

## 3.4 Hint System

When the user clicks the Hint button, the program will check for two criteria: whether a cell on the board is selected or unselected. A cell on the board will be able to be selected by clicking on it directly, or navigating to the cell with arrow keys. To deselect a cell, the user can click in dead space within the GUI that is outside the board and not a button. If the user clicks the hint button with a specific cell selected, the program will give the user the number to that specific cell by citing the stored answer for the board. If the user clicks the hint button with no specific cell selected, the program will randomly choose an unfilled cell on the board to fill. If all cells on the board are filled in, but the game is still ongoing because of an incorrect solution, a text box will appear notifying the user that the board they have is incorrect and to make space for a hint.

# 4. Database Design

## 4.1 Database Overview

The application will use MySQL as the database. Each database entry will consist of 2 main components:

- Sudoku Board
    - Raw Board: The initial unsolved Sudoku Puzzle
    - Solved Board: The final solved Sudoku Puzzle
- Snapshots of the Board Overtime
    - States of the board over time while the Puzzle Solving algorithm is running

The snapshots of the board overtime will be used to visually show the user how the board was solved with the implemented puzzle solving algorithm.

## 4.2 Schema Details

Table: Puzzle
Fields:

- Puzzle ID: Unique identifier for puzzle
- Raw Board: Initial unsolved Sudoku Puzzle
- Solved Board: Final solved Sudoku Puzzle
- Board State: Contains current board data filled in from user (to save history)

Table: Snapshot
Fields:

- Snapshot ID: Integer identifier for snapshot that increments by one
- Puzzle ID: Unique identifier for puzzle Snapshot is linked to
- Board State: Contains board data at time of snapshot

## 4.3 Data Flow

The order of database operation to generate a puzzle will begin when a tentative start board is created. This board will be saved into the Puzzle Table with a generated Puzzle ID. As the puzzle solver is running, snapshots will be created and entries will be made into the Snapshot Table with the linking Puzzle ID. If the board is determined to be unique, the initial Puzzle Table entry will be modified to include the solved puzzle. If the board is determined to have more than one solution, all entries made into the database with the matching Puzzle ID will be removed as it is not a valid board and the process will repeat to generate a new board. As the user makes edits to the board, entries into the Puzzle Table to the Board State will be made after every modification to autosave the users game progress.