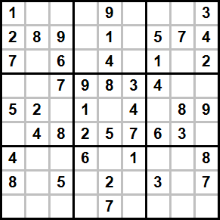
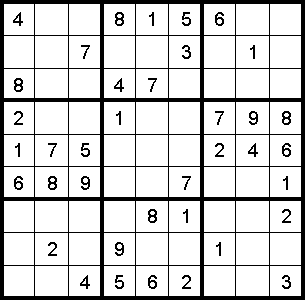
# Analysis – Sudoku Maker/Solver

## Introduction

A Sudoku puzzle is a logic-based, combinatorial number-placement puzzle. The objective is to fill a 9×9 grid with digits so that each column, each row, and each of the nine 3×3 subgrids that compose the grid (also called "boxes", "blocks", "regions", or "subsquares") contains all of the digits from 1 to 9. The puzzle setter provides a partially completed grid, which for a well-posed puzzle has a unique solution. (reference: https://en.wikipedia.org/wiki/Sudoku)



## Aims

The objective of this project is to create **Sudoku puzzle**s with a specific **Difficulty** for a user to solve with a GUI. To guide the user to a solution the program must be able to solve the puzzles in a systematic manner. This approach will allow hints to be provided to the user and will ensure that the puzzle is solvable using a well defined set of techniques.

## Background

I chose this task because I enjoy solving Sudoku puzzles and have found a lack of Sudoku applications that use sophisticated solving techniques and have a rich set of user features. I am particularly interested in puzzle solving strategies such as <http://www.sudokuwiki.org/Jelly_Fish_Strategy> and <http://www.sudokuwiki.org/WXYZ_Wing>. The user features of interested include creating Sudoku problems at various levels of difficulty, providing hints to solve the puzzle at any point in the game and other user support tools to make the puzzle solving experience more enjoyable.

## Research

### Similar Programs

<http://www.sudokuwiki.org/sudoku.htm>

This is an expert-level **Sudoku problem** maker/solver with a cluttered and confusing user interface. However, what sets this maker/solver apart is the level of “Strategies” that can be selected to complete a **Sudoku problem** and the documentation provided. These puzzle solving strategies form the basis of my analysis of the difficulty level of Sudoku puzzles which is used in my solution.

<https://play.google.com/store/apps/details?id=com.brainium.sudoku.free&hl=en>

This is a **Sudoku problem** maker that is available from the Google play store. The special feature provided by this application is an intuitive help button. I plan to use a similar approach to providing help that operates at the different levels of difficulty provided by my application.

Others

<List of urls>

Many other Sudoku makers/solvers are available on the Internet. However, they have a similar set of basic features that allow a user to solve a limited set of Sudoku puzzles. The two applications described above provide distinct ideas which are used in my solution.

### Anticipated Difficulties

<http://stackoverflow.com/a/7280623>

‘Unless P = NP, there is no polynomial-time algorithm for generating general Sudoku problems with exactly one solution.

In his master's thesis, Takayuki Yato defined [The Another Solution Problem](http://www-imai.is.s.u-tokyo.ac.jp/~yato/data2/MasterThesis.pdf) (ASP), where the goal is, given a problem and some solution, to find a different solution to that problem or to show that none exists. Yato then defined ASP-completeness, problems for which it is difficult to find another solution, and showed that Sudoku is ASP-complete. Since he also proves that ASP-completeness implies NP-hardness, this means that if you allow for arbitrary-sized Sudoku boards, there is no polynomial-time algorithm to check if the puzzle you've generated has a unique solution (unless P = NP).’’

This discussion from stackoverflow suggests that the fastest way to generate a **Sudoku puzzle (ie a partially completed grid that a user can solve) from scratch (ie starting with a blank grid) is by** brute force (try every combination of values) as the puzzles are intractable. As this is a very time consuming process ( there are 9^81 possible combinations – although clearly many of these are invalid) a more efficient method of generating Sudoku puzzles was required.

The method chosen to create new **Sudoku puzzles** is to work backwards from a completed Sudoku grid. These completed grids are created by a trial and error, back-tracking algorithm. Once completed grids are available individual tiles can be blanked out and the Sudoku Solver can be used to determine how difficult the puzzle has become. If the puzzle has become too difficult the algorithm can back-track one step and attempt to remove an additional tile to achieve the level of difficulty required. This process is continued until a puzzle with the appropriate set of characteristics (level of difficulty, number of remaining tiles) has been created.

<http://zhangroup.aporc.org/images/files/Paper_3485.pdf>

This paper describes how all the trivial **Transform**ations of an individual **Sudoku Grid** can be computed. (Re-labelling, swapping rows, columns, rotations etc.). This information is used to dramatically reduce the number of **Sudoku Grid**s that are hosted on the server and also increase the number of Sudoku puzzles that can be presented to a user to be solved. This is because a computationally trivial change to a Sudoku grid will look like a completely different puzzle to the un-tutored eye.

## Objectives

* To create a program that generates solvable **Sudoku puzzles** with the maximum number of blank tiles having only one solution.
* To create **Sudoku puzzles with a certain level of Difficulty (defined by the methods required to solve them)**
* To create an automated method of solving **Sudoku puzzles.**
* **To provide a simple and elegant graphical user interface (gui) that provides a rich set of tools to a user(s) to solve Sudoku puzzles.**
* To provide relevant and useful hints that will assist the user in solving the Sudoku puzzles.
* To support the user importation of Sudoku puzzles from other sources (eg Times newspaper).
* To create a database of completed **Sudoku Grid**s which can be used as the basis for **Sudoku puzzles.**
* To provide, maintain and display a high scores table (moves/mistakes/time taken) so that users can track their puzzle solving performance.
* To support a wide selection of Sudoku puzzle solving techniques in puzzle formation and solution.

## Proposed Solution

I will make three pieces of code: One client piece that the user will use and interact with; one for the server back end to generate new **Sudoku problem**s and one for distribution.

For my client code, I will have an options menu and a place where you can play the game. In my options menu I will have choices such as: general **Difficulty** level, specific **Techniques** used and If the user will input their own problem or not, if this option is chosen, then instead of being prompted with a puzzle when the go onto the game UI, they will be prompted with a blank **Sudoku Grid**, where they will first put in the initial **Values**, then press a button and start playing. In the play the game screen the user will be able to input **Values** into the **Sudoku Grid**, input **Dummy Values** into the **Sudoku Grid**, ask for help (which will be a step-by-step guide on what **Technique** they should go to next, and how to use it) and save and exit. If the user chooses to save and exit, the next time they go to play the game, they will be asked if they want to continue, if not the game is deleted, otherwise the game continues.

On the server side I will be generating Completed **Sudoku problem**s and uploading them to my distribution platform. This may seem like very little, however from my research I understand that generating **Sudoku problem**s will be very difficult.

For my distribution I will be running a SQL server hosted off a raspberry pi. This way I do not have to pay for server hosting, it is very easy to customise and upgrade and I won’t have to deal with 3rd party problems.

## Key Words

* **Sudoku Grid**

(see introduction)

* **Sudoku puzzle/problem**

A **Sudoku puzzle/problem** is a Sudoku Grid with minimal values pre-filled in to make it only have one solution for the user to solve

* **Completed puzzle**/ **Sudoku seed**

A **Completed puzzle**/ **Sudoku seed** is a **Sudoku puzzle/problem** that has every single value filled in and obeying the Sudoku laws

* **Values** (with reference to Tiles)

A Value is the Value that is stored within the tile

* **Dummy Values** (with reference to Tiles)

A Dummy value is a **Technique** where the user lists all the possible **Values** for a Tile

* **Difficulty** (with reference to **Sudoku Grid**)

**Difficulty**, in the scope of my project will define the level of **Techniques** that is needed to use to complete the puzzle

* **Techniques**

Atechnique is a method of solving a **Sudoku problem** i.e.**Dummy Values**, [BUG](http://www.sudokuwiki.org/BUG), [X\_Cycles](http://www.sudokuwiki.org/X_Cycles), [Unit\_Forcing\_Chains](http://www.sudokuwiki.org/Unit_Forcing_Chains)& [Sword\_Fish\_Strategy](http://www.sudokuwiki.org/Sword_Fish_Strategy)

* **Transform**ing (a **Sudoku Grid**)

**Transform**ing or **Transform**ations of **Sudoku Grid**s are aprocess, which converts completed **Sudoku Grid**into a new one. I.e. a rotation of 90o

* A **uniqueSudoku seed** (with reference to the SQL server)

A **Sudoku seed** that through no **Transform**ation can result in an uploaded seed