# Documented Design

## High level Overview

The objective of this project is to create, automatically solve and grade (according to difficulty) randomly generated Sudoku puzzles. An easy to use graphical user interface is provided which allows a user to solve these puzzles, provide hints and maintain user scores.

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)

The architecture consists of a server and one or more clients. In the server a Sudoku puzzle generator creates valid 9 by 9 Sudoku grids along with a unique key to identify each puzzle. This unique key is used to identify each puzzle and to eliminate duplicate grids. The grid and unique key are stored in a SQL database that is resident on the server. This SQL database is also used to record a user high scores table.

The application in the client provides the user interface for the Sudoku puzzle. This client application retrieves completed Sudoku grids from the server and then, using the games’ difficulty setting, it eliminates digits from the Sudoku grid to create a unique Sudoku puzzle of the appropriate level of difficulty.

Once the user has started to solve the puzzle, the partial solution is stored on persistent storage in the client machine so that the puzzle can be resumed after a break in play. The user can also request hints to solve the puzzle. Once a puzzle has been completed the user can submit their results to the server which will include: level of difficulty, time taken, guesses made and hints used.

## Server Side

### Class: Node

This class (Node) is used to calculate a unique key for each Sudoku grid. The class uses a tree structure to hold all the trivial variations of a Sudodu grid including rotations, transformations, column and row switching. A rotation is obtained by rotating all digits in the grid by a right angle (swapping x and y values); a transformation is obtained by swapping digit values (eg. All one digits swapped with all two digits); column and row swapping is achieved by swapping adjacent sets (sub squares) of 3 columns or rows.

When a new Sudoko grid has been created, a new tree structure that contains all trivial variations of this new grid is computed, this is called the key. The leaf nodes of this key are compared with all the previously computed keys to ensure that the new Sudoku grid is unique. If any of the leaf nodes match the grid is discarded.

#### Key procedures

##### Generate Grid

This procedure will generate a complete **Sudoku Grid**.  
The procedure uses a back-tracking algorithm to create a valid grid. It starts at the first row and column and inserts a random digit (0 to 9). For each subsequent tile a randomly shuffled list of all digits is used to select the next candidate for insertion. If the next candidate produces a valid grid the digit is inserted and the next tile is processed. Each digit from the list is tested, if no digit can be found that produces a valid grid the algorithm back-tracks to the previous tile and tries the next digit from the list for this tile until if successfully fills the entire grid.

##### Generate Key

This will generate a **Key (all trivial variations of the grid)** from a **Sudoku grid**. The **Key** will be generated by making a Tree using the Node class of all of the possible transformations of the **Sudoku seed**.

##### Get Keys

This will return all of the Sudoku **Key**s from the SQL server. This is used to test for uniqueness of a new key.

##### Check Keys

##### Function: Upload Grid

This function takes uploads the grid and its variations from the tree to the SQL server. The variations are converted from the tree structure to a plain text format by use of ‘in order traversal’.

##### Function: High Score Manager

This function manages the high score table. When the high score table is updated the records are sorted by score and only the top five scores are retained.

## SQL server (Raspberry pi)

A Raspberry pi will be used for the server in this project. This includes to host ting the SQL server on a raspberry pi run from home, this is because for this projects' sake, the load will not be too intensive. If the load were to become more intensive, I would switch to a SQL server hosted off of the internet

## Client Side

### Classes

#### Inheritance diagram

Pygame.Rect

Button

Sudoku tile

Sudoku Grid

#### Button

This will directly inherit from Pygame.Rect a class from Pygame an external module which handles GUI. This will add:

* The text argument and outline argument, allowing me to easily make text boxes.
* A function argument allowing me to easily handle when the buttons are pressed

#### Sudoku tile

Sudoku **Tile** is a class that will directly inherit from Button and will be exclusively used in the **Sudoku grid** class. This will add:

* A limit on the text argument, making it automatically only allow 1 single digit number at a time
* A secondary text argument for **Dummy Values**
* A Producer to change color and allow inputs when clicked on

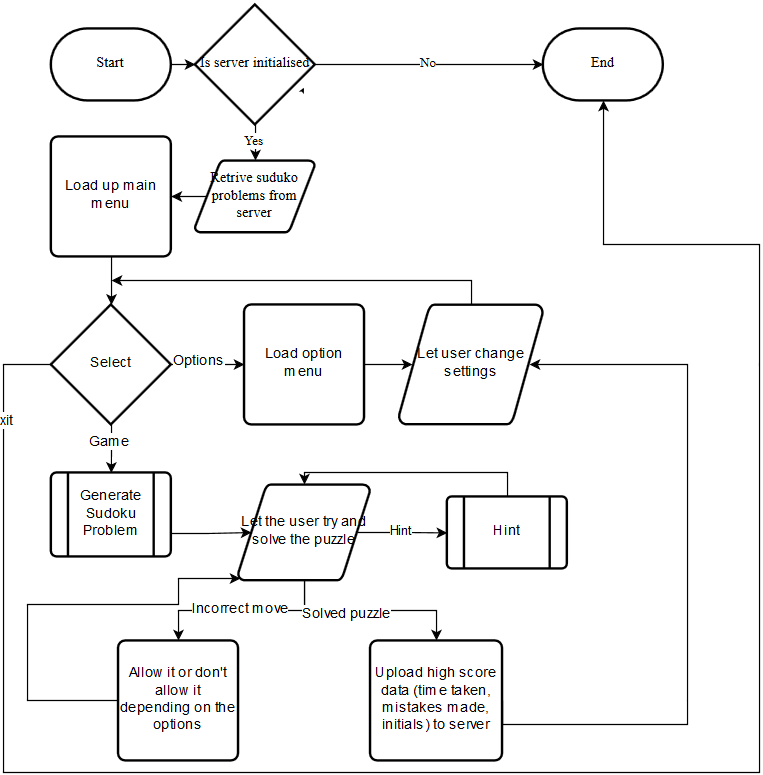
#### Sudoku Grid

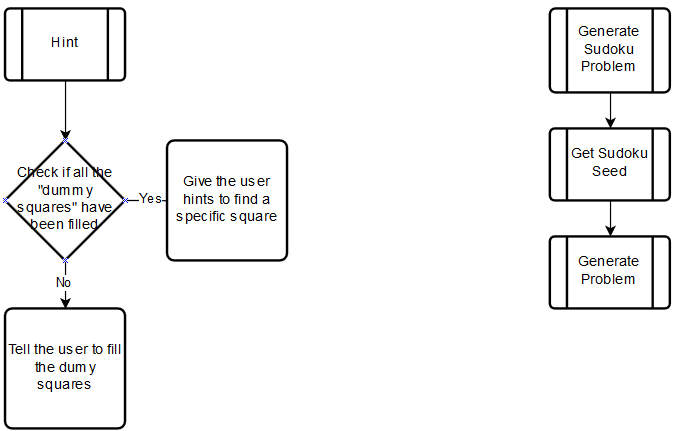
This will be a class is a Composition of Sudoku tiles. By implementing it as a function I can easily keep track of:

* Its **ID**
* High score info

It also gives me an easy way to get the rows and columns for the grid to check if certain moves are legal, and for the help function.

### Key procedures





#### Main

This will be the loop for the main menu; the user will be greeted by this window when launching the game. From here the user will be able to start a game, enter the options menu or exit out of the game. The user will also return to here when they finish a **Sudoku problem**.

#### Draw Screen

This will be the function to draw the screen; I will be able to re-use this for: my option menu, the main menu and the game whilst in progress.

#### Options

This will be the function to run the options menu, from the options menu I intend for the user to be able to change as many aspects of the UI as possible, to be able to change the **Difficulty** settings of the game, and to be able to input their own problems.

The user will be able to change the UI through re-coloring all the different elements of the game.

The **Difficulty** settings will include the **Difficulty** of the problem presented; another way to change the **Difficulty** is to change the amount of starting help the user is given, i.e. by starting the game will every cell having all the possible **Values** able to entered into them already listed; also if the game prevents them from inputting incorrect moves, in Sudoku an incorrect move can often be easily done and then it ruins the rest of the puzzle, these are often very hard to spot, so by disabling that feature entirely, it makes it significantly easier.

#### Game

This will be the loop to run the game, from this window:

* The user can select tiles in the 9x9 grid with left mouse button and input numbers with the on screen number selector, or the user can use the Keyboard. By using the right mouse button the user can choose to input notes, this will allow the user to clearly see all the available **Values** for that tile.
* The user can ask for help, this will prompt a message box to appear and talk the user through step by step how to continue with the **Sudoku puzzle**. This can be dismissed at any point if the user thinks they understand. Using this will disable high scores, and a prompt will tell the user this
* The user can ask the code to solve the puzzle for them, if the user already has a problem from say a newspaper, the can input the puzzle, get the code to solve it for them and then check to see if their answer is correct, this will also disable the high scores.
* The user can choose to save and exit, if the user does this, the next time they start a puzzle they will be asked if they want to load their old one.

#### Import Seed

This will fetch the **Completed puzzle** from the SQL server and feed it into Generate problem.

#### Generate Problem

This will take a solved puzzle from the SQL server and a set **Difficulty** and generate a **Sudoku puzzle** for the user to complete from them. It will do this by first **Transforming** the puzzle then randomly removing tiles and checking if it can find their value from the remaining values.

#### Submit Score to server

This will upload the users’ name, score, **Difficulty** and the Sudoku **Key** that it was completed on to the server along. The score will include how quickly the problem was completed (time) and how many mistakes were made

## Key Words

* **Sudoku grid**

A **Sudoku puzzle/problem** is a 9x9 grid of Sudoku tiles

* **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

* **Key/ID**

A **Key/ID** is a unique identifier for a **Sudoku puzzle/problem** that can catch transformations of that **Sudoku puzzle**

* **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

* **Tile** (with reference to **Sudoku grid**)

A Sudoku **Tile** is an element of a **Sudoku grid** that can contain one non-zero single digit number.

* **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 e.g **Dummy Values**

* **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)

* **Transforming** (a **Sudoku grid**)

**Transforming** or Transformations of **Sudoku grid**s are the different ways one **Sudoku puzzle** can be seen. i.e. a rotation of 90o

* A **unique Sudoku seed** (with reference to the SQL server)

A **Sudoku seed** that cannot be transformed into any of the already uploaded seeds