# Documented Design

## High level Overview

My program will easily allow the user to play a Sudoku game or solve a **Sudoku problem** using a GUI.

On the game side of things, the code will have a server which is constantly randomly generating seeds, these seeds are completed unique **Sudoku problem**s; these seeds are then uploaded to a SQL server hosted by a raspberry pi. These seeds are then taken by the client and converted into a **Sudoku problem** which has one unique solution; the method used to convert the seed to problem will determine the **Difficulty**. During the course of the problem the user will be able to: save their progress, ask for a hint and exit back to the menu. After the problem has been completed by the user, the user can then submit their high score to the server which will include: time taken, guesses made and hints used.

On the solving side of things, the user can choose to have the code solve the problem step by step, showing the user each step (like the hint function) or just automatically solve the problem.

## Server Side

### Classes

#### Node

This class will imitate a tree where all the data is null apart from the leaf nodes. The leaf nodes hold a different rotation or transformation of the **Sudoku seed**. This way I will be able to check if two **Sudoku seed**s are Identical to each other through a defined transformation.

#### Uses

##### Identical trees

This will check if two **Sudoku seed**s are Identical through a transformation, by checking recursively if all of the children are Identical. I will use this when uploading a seed to make sure that I do not upload two of the same seed

##### Generate Key

This will generate a **Key** from a **Sudoku seed**. The **Key** will be generated through a brute force method, by making a Tree using the Node class of all of the possible transformations of the **Sudoku seed**. Other methods such as indexing corner tiles and corner 3x3 blocks would not catch all of the transformations.

#### Key procedures

##### Generate Seed

This procedure will generate a completed **Sudoku Grid**.  
It will do this by going line by line and recursively add a random number that is legal to be put in that tile, however if there is no legal number to put in the tile, the code will backtrack to a value that can be changed and hasn’t been used before and continue on from there

##### Generate Key

This will generate a **Key** from a **Sudoku seed**. The **Key** will be generated through a brute force method, by making a Tree using the Node class of all of the possible transformations of the **Sudoku seed**. Other methods such as indexing corner tiles and corner 3x3 blocks would not catch all of the transformations.

##### Get Keys

This will return all of the Sudoku **Key**s from the SQL server, this way I can check the **Key**s against my own to check if it is a unique seed, and thus check if I can upload it

##### Check Keys

This will iterate through all of the **Key**s from Get **Key**s and check if the seed trying to be uploaded is unique through use of the Identical trees function

##### Upload Seed

This will upload the **Sudoku seed** and its **Key** to the SQL server, the **Key** will be converted from the tree like format to a plain text one by use of in order traversal.

##### High Score Manager

This will go through every record and update the high scores to ensure that they are in order from best to worst, and delete any high score not in the top ten. This way memory can be saved on the SQL server. This will be run separately from the seed upload code

## SQL server (Raspberry pi)

I will be hosting 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

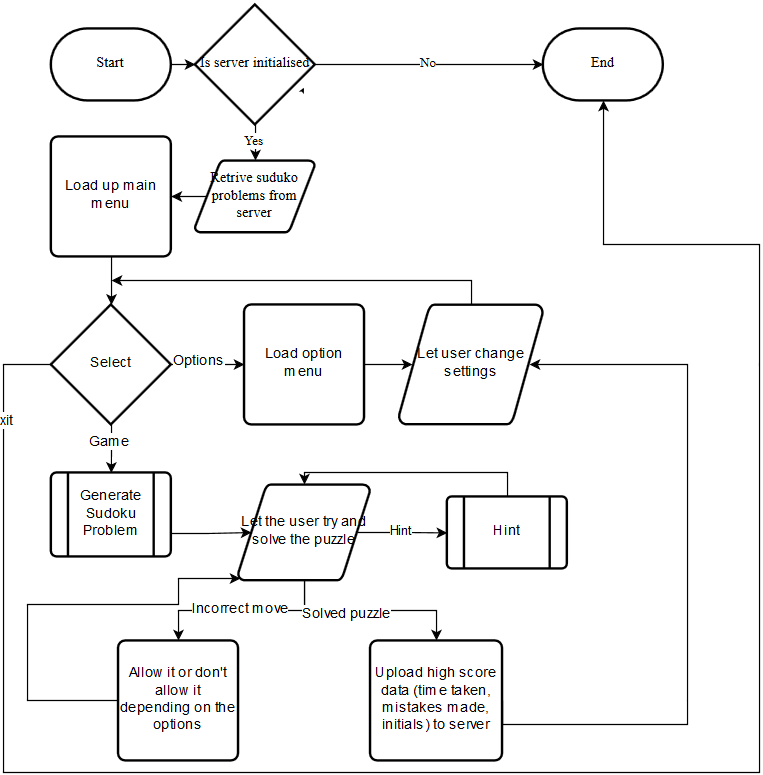
#### Sudoku Grid

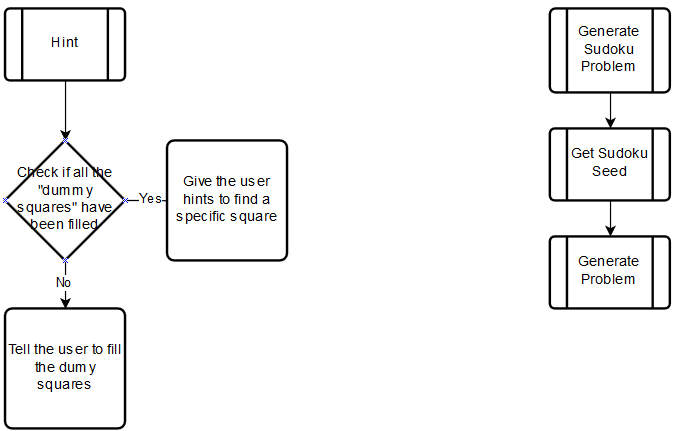
This will be a class is a Composition of Sudoku tiles. By implementing it as a function I can easily keep track of:

* 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