# Technical Solution

Check for temp work arounds like those in SudokuGrid.generate\_new\_puzzle

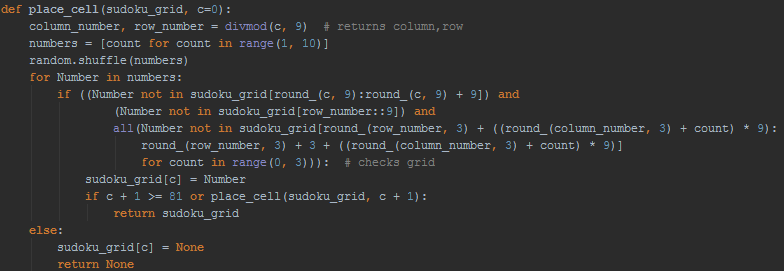
And for non-formal comments

Not included: Sudoku grid, options menu, give hint, anything SQL related, or save(highscore)

Put in my initial stuff

## Server

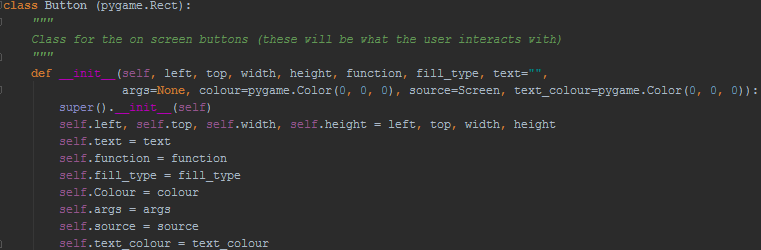
### Place cell



This code generates the completed Sudoku grid recursively by first shuffling a list of numbers 1 to 9. Then it iterates through the list, and it will place the first valid number it reaches. If there is no legal place-able number for that tile, it will backtrack and continue from where it left off on a previous tile

## Client

### Class: Button



This class is used to display everything on the screen. The arguments have the following use:

* Left, top, width, height are used to located the button on the screen
* Text is the text that will be displayed on the button
* Function is the function that will be run when the button is pressed
* Fill\_type is states wheter or no the button will be filled or outlined
* Colour is the Colour of the outline
* Args are the arguments passed into the function
* Source is the screen that the button will be rendered onto
* Text\_colour is the colour of the text rendered onto the button

#### Draw

Draw is the method that draws the button onto the source surface. It first draws a rectangle the same colour as the background over its current position. Then if it is an outline a rectangle width 5 drawn like so, otherwise, it fills in a rectangle in the place, then it renders the text to the tile. Finally updating the screen

#### Initialise render font to rect

Initialise render font to rect tries to render the text on to the text box.  
Firstly it counts the number of new lines are in the text, then if there are any it goes through every line and tries to fit it into the box with the largest font size. Once it has found all the font sizes for each line it chooses the smallest one and sets that as the font size for each line.

Then it goes through each line and renders it to the screen, by choosing the text up to the line break to render and the text after the first line break to repeat, if there are no more line breaks indexing the string will throw an error, and then it renders the last line.

Finally if there was no line breaks it gets the font size and then renders the font

#### Render font to rect

Render font to rect takes the text, the font size and the location of text and renders it concentrically.

#### Calculate font size

Calculate font size takes the text to render and the space it has and gets the maximum font size that allows the text to fit in it.

Firstly, smallest font size is set to nine .Then if the text can fit in the rectangle the font size is incremented, if that is not the case the font size is decrease and that is returned as the largest font that can fit in the rectangle

### Input handle

This is the main loop. Every tick it will get the most recent input from the user. If the user clicked their mouse, the handle\_click function is called. If the user pressed a numerical button then it will try to edit the tile’s value; if the tile cannot be edited i.e. the edit procedure is not defined for that object then it will call an Attribute Error, which I handle by just skipping the step.

### Handle click

This function takes the arguments: left\_click – a Boolean statement to determine whether the left button was pressed and mouse\_pos – the postion of the mouse on the screen. The code checks which button on the screen was clicked. If the game is currently in the Game menu, it will remove the tile highlighting in the Sudoku grid. It stores the last tile clicked on, so the user can edit its value in the main loop. Finally, the function of the object will be run with arguments if it has them.

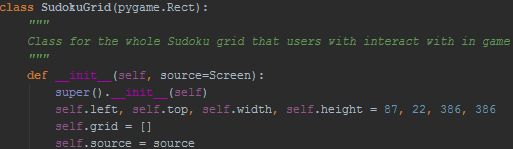
### Rotation

This takes a 1D array that represents a 2d square (with length being the length of one of the sides) and rotates it through 90o, 180o, 270o it does this by using a Cartesian representation of matrix rotation.

This shows the matrix rotation of a 2d vector through, which is the equation I have used (defined in rotate\_to)

In the turn function it takes the rotate\_to function, the array and the length of one of the sides. It firstly makes a blank list of the same length of the array, and then iterates through the array keeping track of its index, next, the index is converted into a set of co-ordinates, which are then transformed by the rotate\_to function then the new\_index is made and the value is written to that location in the new array.

### Class: SudokuGrid



The sudoku grid class stores every tile in the grid in it, its location, the source screeen

#### Initialise values

Is the first method called, as the Sudoku grid is initialised at the very start of running the game, however the user may want to change the difficulty setting before generating a problem (as the difficulty settings have a direct effect on the problem made.

#### Get values

These methods take either a tile index or a row/column/ sub grid index, these make finding all the tiles that can affect a single tile very easy.

All the return statements are tuples to allow for hashing.