Sudoku Maker & Solver Application

A project submission for a Computer Science A level

by Edward Boulderstone

Bourne Grammar School

2016/17

Contents

1 Analysis – Sudoku Maker/Solver

1.1 Introduction

A Sudoku puzzle is a logic-based, combinatorial number-placement puzzle. The objective is to fill a 9x9 grid with digits so that each column, each row, and each of the nine 3x3 sub-grids that compose the grid (of tiles) 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`)

4			8	1	5	6		
		7			3		1	
8			4	7				
2			1			7	9	8
1	7	5				2	4	6
6	8	9			7			1
				8	1			2
	2		9			1		
		4	5	6	2			3

1				9				3
2	8	9		1		5	7	4
7		6		4		1		2
		7	9	8	3	4		
5	2		1		4		8	9
	4	8	2	5	7	6	3	
4			6		1			8
8		5		2		3		7
				7				

1.2 Aims

The objectives of this project are to create a program that generates Sudoku puzzles that are solvable with a defined set of puzzle solving techniques; to provide a Graphical User Interface that uses these puzzles to create a interesting puzzle solving environment for the user; to provide a consistent and robust scoring method for puzzle solving.

1.3 Background

I chose this task because I enjoy solving Sudoku puzzles and have found there is 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 puzzles at various levels of difficulty, providing hints to solve the puzzle at any point in the game and other user support tools to make puzzle solving more enjoyable.

1.4 Research

1.4.1 Similar Programs

`http://www.sudokuwiki.org/sudoku.htm`

This is an expert-level Sudoku puzzle maker/solver with a cluttered and confusing user interface. However, what sets this maker/solver apart is the range of puzzle solving techniques that can be selected to complete a Sudoku puzzle and the documentation provided. These puzzle solving techniques 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 puzzle 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.

1.4.2 Other Programs

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. A list of other Sudoku applications follows.

- https://www.sudoku-solutions.com/
- https://play.google.com/store/apps/details?id=com.jdamcd.sudokusolver&hl=en_GB
- https://play.google.com/store/apps/details?id=com.alkobyshai.sudokusolver&hl=en
- https://play.google.com/store/apps/details?id=com.enigon.sudokusolver&hl=en
- https://play.google.com/store/apps/details?id=de.georgwiese.sudokusolver&hl=en

1.5 Anticipated Difficulties

`http://stackoverflow.com/a/7280623`

'Unless P = NP, there is no polynomial-time algorithm for generating general Sudoku puzzles with exactly one solution. In his master's thesis, Takayuki Yato defined The Another Solution puzzle (ASP), where the goal is, given a puzzle and some solution, to find a different solution to that puzzle or to show that none exists. Yato then defined ASP-completeness, puzzles 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 digits) 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 is 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 Transformations 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 Grids 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.

1.6 Objectives

1. To create a program that generates solvable Sudoku puzzles with the maximum number of blank tiles having only one solution.
2. To create an automated method of solving Sudoku puzzles.
3. To support a wide selection of Sudoku puzzle solving techniques in puzzle formation and solution.
4. 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.
5. To create a database of completed Sudoku grids which can be used as the basis for Sudoku puzzles.

6. To provide, maintain and display a high scores table (moves/mistakes/time taken) so that users can track their puzzle solving performance.

7. To provide a consistent scoring method for similar Sudoku puzzles.

1.7 Proposed Solution

I decided to split the solution to these objectives into three distinct modules. These modules are the client application that contains the user interface and resides on the client machine; the Sudoku puzzle generator that resides on the server and the administration module that also resides on the server. The administration module manages the communication between the components, the high-score table and provides access to the relational database that stores the Sudoku puzzles, high-score information and other ancillary information.

The client application will contain the options menu and the main Sudoku playing area. The options menu will allow the user to set the features for puzzle playing. These features include the general difficulty level of the puzzles presented for solving, the specific Sudoku techniques that will be required to solve puzzles presented to the user and an option to permit the user to enter a Sudoku puzzle of their own from another source. If this final option is selected, a blank Sudoku grid will be presented to the user so that the initial digits of the external Sudoku puzzle can be entered.

Once the button to start play on the user interface has been pressed, the predefined digits of the Sudoku puzzle will be locked in place and the user will be able to input digits into the Sudoku Grid. The user can write Dummy Values (which appear in a small font) into blank tiles to assist the user in solving the puzzle, ask for help (which will be a step-by-step guide on what Technique they should be used next) and a save and exit option. If the user chooses to save and exit, the next time the application is started, the user will be asked whether they want to continue with the previous puzzle, if this option is selected the previous puzzle will be retrieved from the local storage and presented to the user.

The server will host the puzzle generator and the administration module. The administration module uses a mySQL database and a Raspberry PI as the hardware platform. This provides a cost effective database environment to support the server processing.

2 Documented Design: Sudoku Maker/Solver

2.1 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.

The architecture consists of a server and one or more clients. In the server a Sudoku puzzle generator creates valid Sudoku grids along with a key to identify each puzzle. This key is used to identify each puzzle and to eliminate duplicate grids. The grid is stored in a SQL database that is resident on the server. This 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. Then, using the games' difficulty setting, it removes 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.

2.2 Server Side

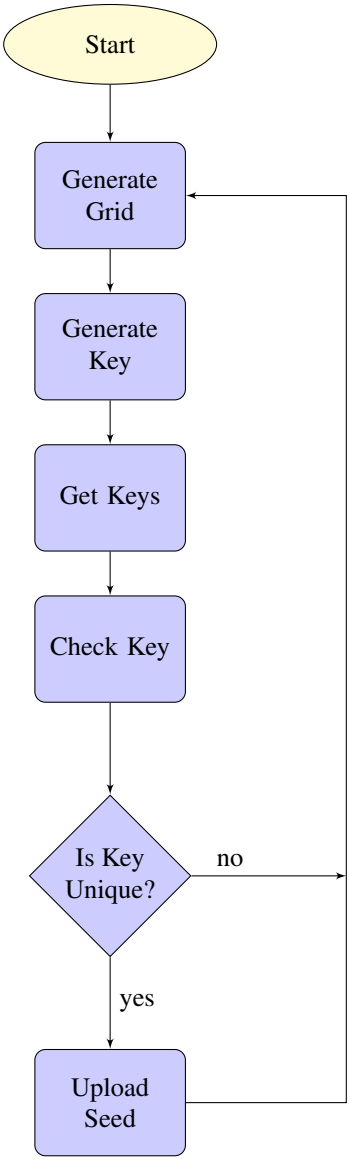
2.2.1 Class: Node

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. These trivial variations are created by simply swapping digits in the grid so can be regarded as the 'same' puzzle from a mathematical standpoint. A rotation is obtained by rotating all digits in the grid by 90° (swapping x and y values); a transformation is obtained by swapping digits (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 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 the previously computed keys to ensure that the new Sudoku grid is unique (except for the trivial variations). If any of the leaf nodes match the grid is discarded.

2.3 Key procedures

This flowchart shows the high=level operation of processes to create a complete Sudoku grid.



2.3.1 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 (digital location) a randomly shuffled list of digits (0 to 9) is used to select the next candidate for insertion. This list is maintained with each tile location. 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 it successfully fills the entire grid.

2.3.2 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 trivial transformations of the Sudoku seed.

2.3.3 Get Keys

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

2.3.4 Upload Grid

This function 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 an ‘in order traversal’ algorithm.

2.3.5 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.

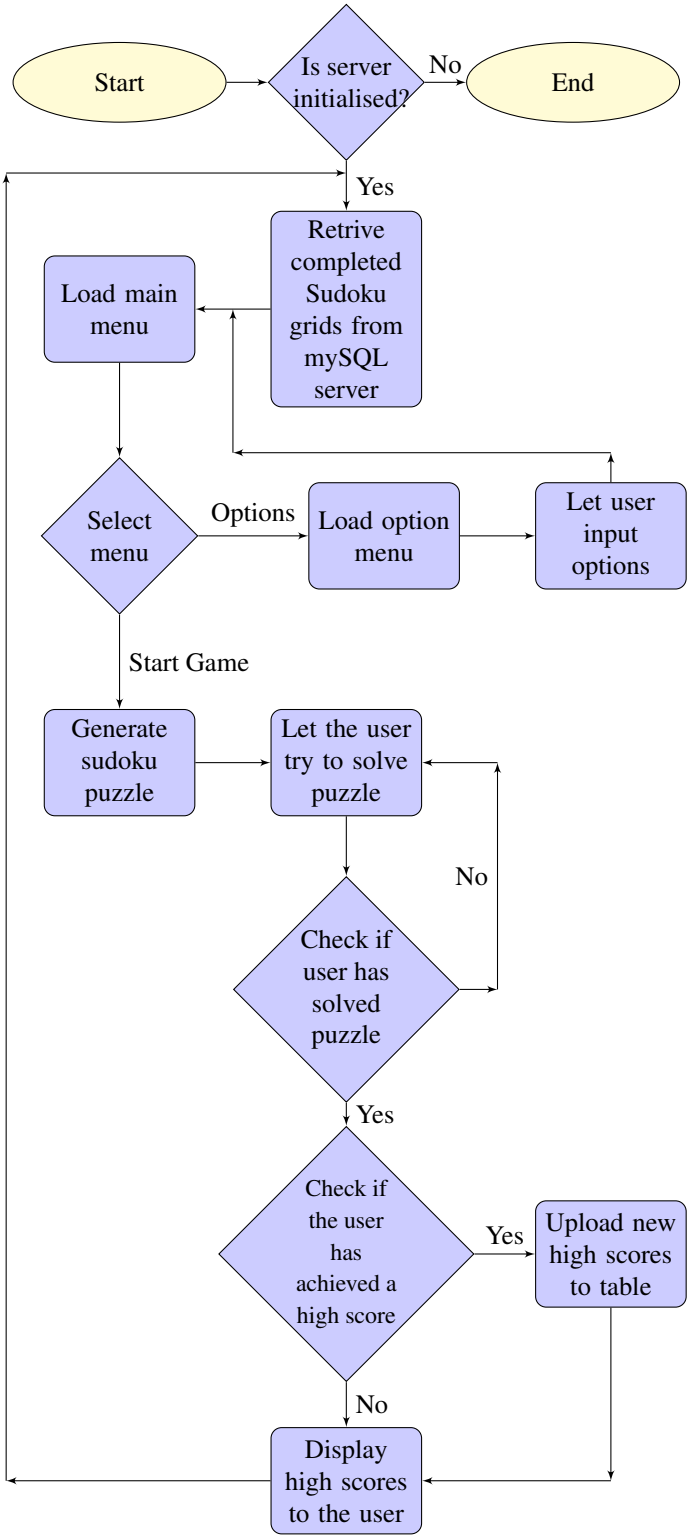
2.4 SQL server (Raspberry pi)

A Raspberry pi is used for the server in this project including hosting the SQL server. Consideration was given to internet hosting however since the compute load is relatively light this was not thought necessary.

2.5 Client Side

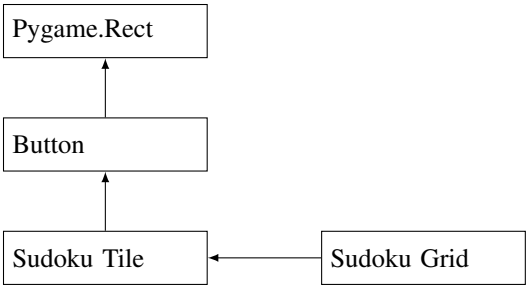
2.5.1 Flowchart

The following is a flow chart of the major functions of the client application.



2.5.2 Classes

2.5.2.1 Inheritance Diagram



2.5.2.2 Button

This class will directly inherit from Pygame.Rect. Pygame.Rect inherits from Pygame which is provided by the Python libraries. The Button class will:

- Take a text string and outline flag that creates a text-based button with an optional outline easily.
- Take a function as an argument, allowing the button to transfer control to the function when the button is pressed.

The Button class is used for both normal game control and the digits within the Sudoku grid.

2.5.2.3 Sudoku tile

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

- A text argument (entered and displayed to the user) to assist solving the puzzle (this could be used to display possible digits as small digits in each tile.)
- Functions to change the colour of the button/tile and to allow user input when selected.

2.5.2.4 Sudoku Grid

This class is a composition of Sudoku tiles, this class keeps track of:

- The current Sudoku puzzle.
- High score information.

This class also provides a method to enumerate the tiles in a row or column so that checks can be made to determine if certain moves are legal, and for the help function.

2.5.3 Key procedures

2.5.3.1 Main

This function provides the main loop for the program.

2.5.3.2 Main Menu

The user will be greeted by the main menu when launching the game. From this menu 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 the main menu when they finish a Sudoku puzzle.

2.5.3.3 Draw Screen

This function draws the main screen, options menu and the Sudoku puzzle; the function is used to redraw the screen when the game is in progress.

2.5.3.4 Options

This function manages the options menu. The design goal for the options menu is to allow the user to change as many aspects of the UI (colours/settings) as possible. This includes the Difficulty settings of the game, the hints the user can obtain and the colours of all the graphical elements. In addition, this menu allows the user to enter complete Sudoku puzzles from other sources.

The Difficulty settings, within the options menu, controls various aspects of the game. These include the relative difficulty of solving the Sudoku puzzles presented to the user as calculated by the Sudoku generator; the amount of help the user is given, i.e. displaying all possible correct digits in tiles (hence eliminating the trivial incorrect digits); whether an incorrect move is accepted (this is important because an incorrect move in Sudoku makes the rest of the puzzle impossible to solve).

2.5.3.5 Game Menu

From this window:

- The user can select tiles in the 9x9 grid with the left mouse button and input digits through the user's Keyboard. By using the right mouse button the user can choose to input dummy values.
- The user can ask for help, this will cause a message box to appear which will instruct the user how to continue with the Sudoku puzzle. This message box can be dismissed at any point if the user is satisfied with the help provided. Using this feature will disable recording of high scores, and a prompt will tell the user this.
- The user can ask the application to solve a Sudoku puzzle for them. The Game Menu will allow the user to input an external puzzle from another source (e.g. newspaper). The application will allow the user to solve this puzzle in the normal way or solve it automatically. This feature also disables recording of 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 puzzle or start a new one.

2.5.3.6 Import Seed

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

2.5.3.7 Generate puzzle

This will take a completed Sudoku grid from the SQL server together with the Difficulty level and generate a Sudoku puzzle for the user to complete. It will do this by first performing a trivial transformation of the puzzle then randomly removing tiles and checking to determine whether the puzzle can still be solved. This uses the difficulty settings to determine which solving strategies must be used to solve the puzzle. The higher the difficulty setting the more advanced the set of strategies that will be required to solve the puzzle.

2.5.3.8 Submit Score to server

This will upload the users' name, score and the Sudoku Key that was completed on to the server. The score will include how quickly the puzzle was completed (time) and how many mistakes were made.

3 Technical Solution

3.1 Server

The architecture consists of a server and one or more clients. In the server a Sudoku puzzle generator creates a valid 9 by 9 Sudoku grid along with a unique key to identify each puzzle. The grid and unique key are stored in a SQL database that is resident on the server.

3.1.1 Place cell

This code places one digit into the cell indexed by the parameter c. This function is called recursively to fill the entire grid. Firstly a list of numbers 1 to 9 are shuffled (line 3 & 4). The function checks whether the selected digit (Number) is in the row (line 6) and then in the column (line 7). If successful it tests to see whether the digit is in the sub-grid (line 8-10). If all tests are passes it sets the value into the Suduko grid (line 11). The termination condition is evaluated on line 12. If the end of the grid has been reached the completed Sudoku grid is returned. If any of the tests fail, the location in the grid is set to 'None' and the for loop continues (line 15).

```
1  def place_cell(sudoku_grid, c=0):
2      column_number, row_number = divmod(c, 9)  # returns column,row
3      numbers = [count for count in range(1, 10)]
4      random.shuffle(numbers)
5      for Number in numbers:
6          if ((Number not in sudoku_grid[round_(c, 9):round_(c, 9) + 9]) and
7                  (Number not in sudoku_grid[row_number::9]) and
8                  all(Number not in sudoku_grid[round_(row_number, 3) + ((round_(column_number, 3) + count) * 9):
9                      round_(row_number, 3) + 3 + ((round_(column_number, 3) + count) * 9)]
10                     for count in range(0, 3))):  # checks grid
11             sudoku_grid[c] = Number
12             if c + 1 >= 81 or place_cell(sudoku_grid, c + 1):
13                 return sudoku_grid
14     else:
15         sudoku_grid[c] = None
16         return None
```

3.1.2 Rotation & Turn

This code does matrix rotation of a Sudoku grid represented by a one-dimensional array (array) at 90°, 180° or 270° using the following formula.

$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \end{bmatrix}$$

The function rotation rotates the grid, stored as a one dimensional array (array) by the angle of rotation to map a set of x-y coordinates to the rotated version of these co-ordinates (lines 6-18). Then it calls the turn function. The turn function uses the mapping function rotate_to to create a new one-dimensional array (rotated_array) of the rotated values.

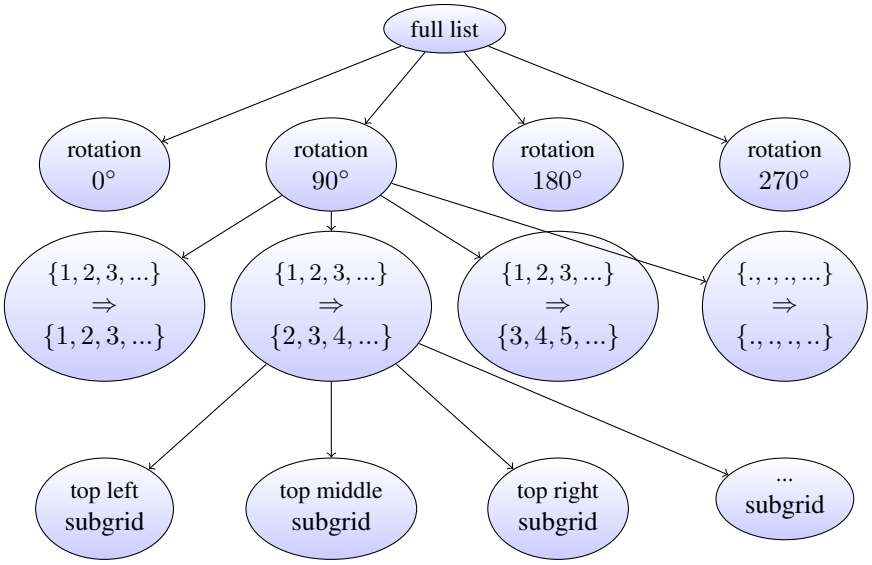
```
1  def rotation(array, rotation_angle, length):  # uses matrix rotation
2      try:
3          length = length[0]
4      except (AttributeError, TypeError):
5          pass
6      if rotation_angle == 90:
7          def rotate_to(x, y):
```

```
8               return x, length - y
9
10          rotated_array = turn(array, rotate_to, length)
11      elif rotation_angle == 180:
12          def rotate_to(x, y):
13              return length - y, length - x
14
15          rotated_array = turn(array, rotate_to, length)
16      elif rotation_angle == 270:
17          def rotate_to(x, y):
18              return length - x, y
19
20          rotated_array = turn(array, rotate_to, length)
21      try:
22          return rotated_array
23
24      except NameError:
25          return array
26
27
28  def turn(array, rotate_to, length):
29      rotated_array = [None for _ in range((length + 1) ** 2)]
30      for index, index_value in enumerate(array):
31          y, x = divmod(index, length + 1)
32          new_y, new_x = (rotate_to(x, y))
33          new_index = new_y * (length + 1) + new_x
34          rotated_array[new_index] = index_value
35      return rotated_array
```

3.1.3 Generate Key

This function creates a data structure that contains all the information to match trivial versions of the passed Sudoku grid. The data structure consists of 3 levels, the highest level supports the 4 rotated versions of the grid ($0°$, $90°$, $180°$ and $270°$), the second level supports all of the possible digit swapping permutations (9 factorial in total). The third level supports all of the valid sub-grid (3x3) positions within the grid (9 positions). This data structure therefore could be used to construct all possible permutations of the passed Sudoku grid and can therefore be used to uniquely identify a Sudoku grid.

The diagram below provides a visual representation of this data structure. It shows a tree with 'full list' as the parent node that has the 4 rotations as immediate children. Each rotation has 9 factorial children (with only one set of nodes shown here) under the $90°$ Rotation node. These child nodes represent the digit swapping permutations. These digit swapping nodes also have child nodes that store the sub-grid locations (with only 4 of the 9 shown in the diagram. The sub-grid nodes contain a list of 5 sets of sub-grid digits that defines the individual permutation.



The code at lines 4-20 create all the valid sub-grids for the passed Sudoku grid (level 3). While the code at lines 21-25 creates the 'digit-swapping' permutations and the rotations are constructed at line 26. The output from the function is a list of all possible variations (full_list).

```
1  def generate_key(sudoku_grid):
2      full_list = []
3      check_list = []
4      for Index_Tile in range(0, 9):
5          for Row in range(0, 3):
6              for Tile_Row in range(3):
7                  check_list[Row * 9 + Tile_Row * 3:Row * 9 + (Tile_Row + 1) * 3] = \
8                      sudoku_grid[((((Index_Tile // 3) + Row) * 9 + Tile_Row * 3 + Index_Tile % 3) * 3:
9                           (((Index_Tile // 3) + Row) * 9 + Tile_Row * 3 + 1 + Index_Tile % 3) * 3]
10         for Column in range(1, 3):
11             for Tile_Row in range(3):
12                 check_list[27 + (Column - 1) * 9 + Tile_Row * 3:27 + (Column - 1) * 9 + (Tile_Row + 1) * 3] = \
13                     sudoku_grid[((Index_Tile // 3) * 9 + Tile_Row * 3 + (Index_Tile + Column) % 3) * 3:
14                          ((Index_Tile // 3) * 9 + Tile_Row * 3 + 1 + (Index_Tile + Column) % 3) * 3]
15         check_lists = [check_list,
16                        [check_list[9:18] + check_list[:9] + check_list[18:]][0],
17                        [check_list[18:27] + check_list[:18] + check_list[27:]][0],
18                        [check_list[27:36] + check_list[:27] + check_list[36:]][0],
19                        [check_list[36:] + check_list[:36]][0]]
20         tile_indexed = []
21         for StartIndex in range(len(check_lists)):
22             index = [x for x in range(1, 10)]
23             for _ in itertools.repeat(None, 3):
24                 change_dict = {index[x]: x_value for x, x_value in enumerate(check_lists[StartIndex][:9])}
25                 tile_indexed.append([change_dict[int(x)] for x in check_lists[StartIndex][9:]])
26                 index = rotation(index, 90, 2)
27         full_list.append(tile_indexed)
28     return full_list
29     }
```

3.1.4 Node

The Node class is a holder for the full list parent node described in the Generate Key class above.

```
class Node:
    # this will be used for the ID of the sudoku grids
    def __init__(self, children):
        self.children = children

    def get_children(self):  # This will return all of this node's child nodes
        return self.children

    @staticmethod
    def get_type():  # This function will test (when comparing two Trees)
        # if the current object is a node class or a list
        return True
```

3.1.5 identical_trees

The identical_trees function tests to determine whether two Nodes are identical. It is used to ensure unique Sudoku grids are created.

```
def identical_trees(root1, root2):  # This will check if the two nodes have identical children and
    # thus are identical Sudoku puzzles
    try:
        if root1.get_type() == root2.get_type():  # If the two objects are part of the Node class,
            #  if not trigger an AttributeError
            for child1, child2 in itertools.product(root1.get_children(), root2.get_children()):
                # Checks if they share the same children
                if child1 == child2:
                    return True
                else:
                    if identical_trees(child1, child2):
                        return True
    except AttributeError:
        if root1 == root2:
            return True
```

3.1.6 generate_upload_grid

The generate_upload_grid function stores a new Sudoku grid (created by generate_completed_grid - line 10) if it is not a duplicate of a grid that is already in the database. The function selects all grids from the table (line 6) and tests to determine whether the uploaded_grid is equal to new grid (line 14). If no match is found the grid is stored in the database with the insert SQL command (line 20).

```
def generate_upload_grid(connection, table):
    db = connection[0]
    cur = connection[1]
    cur.execute("SELECT grid FROM " + table)
    uploaded_grids = cur.fetchall()
    uploaded_grids = [uploaded_grid[0] for uploaded_grid in uploaded_grids]
    while True:
        grid = generate_completed_grid()
        if VERBOSE:
            print("Grid generated")

        if not any(identical_trees(generate_seed(uploaded_grid), generate_seed(grid))
```

```
                for uploaded_grid in uploaded_grids):
            if VERBOSE:
                print("No copy of grid uploaded")
            grid = str("".join(map(str, grid)))
            uploaded_grids.append(grid)
            cur.execute("INSERT INTO " + table + "(grid) VALUES ('" + str("".join(map(str, grid))) + "')")
            db.commit()
        elif VERBOSE:
            print("Grid already uploaded")
            print()
```

3.1.7 establish_connection

The establish_connection function attempts connect to a local MySql database. If this connection fails it attempts to connect to port-forwarded remote database hosted on the Raspberry Pi. If successful it displays a message and prepares the database for transactions.

```
def establish_connection(connection_address):
    while True:
        try:
            db = mysql.connector.Connect(**connection_address)
        except mysql.connector.errors.InterfaceError:
            if connection_address['host'] != HOSTS[1]:
                connection_address['host'] = HOSTS[1]
                print("Server unreachable, trying local IP")
                continue
            else:
                print("Server down please try again")
                exit()
        break
    print("Connection established")
    db.start_transaction(isolation_level='READ COMMITTED')
    cur = db.cursor(buffered=True)
    return [db, cur]
```

3.2 Client

The client application provides the user interface through which the user selects the puzzle options, solves the puzzle and provides a facility to enter an external Sudoku puzzle. When the user presses the start game button, the application retrieves a completed Sudoku grid 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.

3.2.1 Button

The Button class inherits from an external library (Pygame) and is used to display the Sudoku grid and tiles, menu buttons and high scores table. It supports different text colours, automatic font selection to fit the text onto the button and generically processes button events.

```
class Button(pygame.Rect):
#
```

```
3  # Button(pygame.Rect)
4  # Class for the on screen buttons (these will be what the user interacts with)
5  #
6
7      def __init__(self, left, top, width, height, function, fill_type, text="",
8                   args=None, colour=pygame.Color(0, 0, 0), source=Screen,          text_colour=pygame.Color(0, 0,
9          super().__init__(self)
10         self.left, self.top, self.width, self.height = left, top, width, height
11         self.text = text
12         self.function = function
13         self.fill_type = fill_type
14         self.Colour = colour
15         self.args = args
16         self.source = source
17         self.text_colour = text_colour
18 #
19 # draw(self)
20 # function to render the button onto the screen including text
21 #
22     def draw(self):
23         pygame.draw.rect(self.source, (190, 190, 190), self)
24         if self.fill_type == "Outline":
25             pygame.draw.rect(self.source, self.Colour, self, 5)
26         else:
27             pygame.draw.rect(self.source, self.Colour, self)
28         self.init_render_font_to_rect()
29         self.update()
30 #
31 # init_render_font_to_rect(self)
32 # function to render text onto the button, splits the text into separate lines,
33 # and uses calc_font_size for each line to select the appropriate font to fit the
34 # button
35 #
36     def init_render_font_to_rect(self):
37         font_sizes = []
38         breaks = self.text.count("\n")
39         temp_text = self.text
40         if breaks:
41             for _ in range(breaks + 1):
42                 render_text, temp_text = temp_text[:temp_text.index("\n")], temp_text[temp_text.index("\n"):]
43                 font_sizes.append(self.calc_font_size(render_text, breaks))
44             font_size = min(font_sizes)
45             temp_text = self.text
46             for count in range(1, breaks + 2):
47                 try:
48                     render_text, temp_text = temp_text[:temp_text.index("\n")], temp_text[temp_text.index("\n") +
49                     self.render_font_to_rect(render_text, font_size, count, breaks)
50                 except ValueError:
51                     self.render_font_to_rect(temp_text, font_size, count, breaks)
52
53         else:
54             font_size = self.calc_font_size(self.text, 0)
55             self.render_font_to_rect(self.text, font_size, 1, 0)
56 #
```

```
# render_font_to_rect(self, text, font_size, count, breaks)
# function to draw the text onto the button
#
    def render_font_to_rect(self, text, font_size, count, breaks):
        font_rect = FONT.get_rect(text, size=font_size)
        FONT.render_to(self.source, (((self.left + self.width / 2) - font_rect.width / 2),
                                     (self.top + (self.height * count / (breaks + 2))) - (font_rect.height / 2)),
                       text, self.text_colour, size=font_size)
#
# calc_font_size(self, text, breaks)
# function to calculate the appropriate font size for the text
#
    def calc_font_size(self, text, breaks):
        end_while = False
        increment = 10
        while not end_while:
            font_rect = FONT.get_rect(text, size=increment)
            if font_rect.height > (self.height / (breaks + 1)) or font_rect.width > (2 / 3) * self.width:
                increment -= 1
                end_while = True
            else:
                increment += 1
        return increment
#
# update(self)
# function to update screen at button location
#
    def update(self):
        pygame.display.update(self)
#
# check_clicked_on(self, loc) - function to handle button events
#
    def check_clicked_on(self, loc):
        if self.collidepoint(loc):
            return self, self.function, self.args
        else:
            return False, None, None
#
# change_text(self, new_text)
# function to update the text shown on the button
#
    def change_text(self, new_text):
        self.text = str(new_text)
```

3.2.2 TextBox

The TextBox class inherits from the Button class, it provides text input and edit support to allow the user to input and modify text, in particular it supports delete and escape functions.

```
class TextBox(Button):
    def __init__(self, left, top, width, height, function, args=None, max_values=10, text="",
                 colour=pygame.Color(0, 0, 0), source=Screen, text_colour=pygame.Color(0, 0, 0)):
        self.left, self.top, self.width, self.height = left, top, width, height
        self.text = text
```

```
6           self.Colour = colour
7           self.source = source
8           self.text_colour = text_colour
9           self.max_values = max_values
10          self.function = function
11          self.args = args
12          super().__init__(self.left, self.top, self.width, self.height, None, "Fill",
13                           colour=self.Colour, text_colour=self.text_colour)
14
15      def edit(self, event):
16          if (event.key == 8 or event.key == 127 or event.key == 266) and len(self.text) > 0:
17              self.text = "".join(list(self.text)[:-1])
18          elif event.key in range(49, 58) and len(self.text) != 10:
19              self.text += event.unicode
20          elif event.key == 13:
21              self.function(self.text, self.args)
22
23              swap_screen("High_score_Tables")
24
25      def set_args(self, new_args_list):
26          self.args = new_args_list
```

4 Evaluation

4.1 Objectives

The following is a list of objectives for the project and the progress that was made against each objective.

1. *To create a program that generates solvable Sudoku puzzles with the maximum number of blank tiles having only one solution.*

   Fully Completed – This was achieved by randomly blanking out tiles from a complete(all tiles contain digits) Sudoku grid. The complete Sudoku grid was created on the server by the place_cell function. These grids are then stored in the mySQL database for retrieval by the client. The client retrieves the completed Sudoku grid, performs a random combination of trivial transformations (rotation, number swap, shuffles) and then randomly removes digits from the grid, to form an appropriate Sudoku puzzle. These processes include a step to ensure that the Sudoku puzzle is solvable using a defined set of Sudoku solving techniques.

2. *To create an automated method of solving Sudoku puzzles.* See item 3 below.

3. *To support a wide selection of Sudoku puzzle solving techniques in puzzle formation and solution.*

   Fully Completed – This was achieved by using a set of Sudoku puzzle solving techniques for which a set of functions was written. The solving functions were used iteratively as each time a dummy value (a possible digit in a Sudoku tile) is removed from the puzzle, one or more of the other solving techniques may enable removal of more dummy values leading ultimately to a solution of the Sudoku tile and puzzle. The Sudoku puzzle solving techniques that were used are:

   (a) Trivial Dummy Values

   The Trivial Dummy values functions removes dummy values based on the resolved digits by eliminating all dummy values that are the same as the resolved digits in their respective row, column or sub-grid.

   (b) Tuples

   The Tuples functions obtains the set of 9 lists of dummy values from a row, column or sub-grid that is passed to it. The function then iterates through the solve functions, which look for sets of digits that are restricted to one, two or three tiles and look for a set of tiles where only one, two or three digits can be located (these are called hidden 1 singles, pairs, triples and naked 2 singles, pairs, triples). If a hidden or naked tuple is found all dummy variables that are no longer valid are removed from the remaining tiles (See the functions: remove naked tuples, remove hidden tuples below)

   (c) Intersection Removal

   The Intersection Removal functions attempts to use the method Pointing Pairs, Pointing Triples and Box Line Reduction to remove dummy values from tiles. The Pointing Pairs and Pointing Triples methods identifies pairs or triples of aligned digits in the sub-grid, since these aligned digits must occur in the sub-grid, they can be eliminated from the corresponding row or column in the full Sudoku grid. The Box Line Reduction method identifies pairs or triples of digits in a row, that only occur in the same sub-grid. These digits can be eliminated from the other tiles not on that row in the sub-grid.

4. *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.*

   Partially Completed - A set of classes were written (Button, TextBox, SudokuGrid & SudokuTile) to provide the Graphical User Interface. To maintain consistency and ease of use the same class (Button) was used for all user event handling, and as the base class for both TextBox and SudokuTile. This inheritance of the functionality in Button ensured that the TextBox and SudokuTile classes respond the same way to user interaction as the Button class itself.

   A high level description of each GUI class

   (a) Button

   The Button class inherits from an external library Pygame and is used to display the Sudoku grid and tiles, menu buttons and high scores table. It supports different text colours, automatic font selection to fit the text onto the button and generically processes button events.

(b) TextBox

The TextBox class inherits from the Button class, it provides text input and edit support to allow the user to input and modify text, in particular it supports delete and escape functions.

(c) SudokuGrid

The SudokuGrid class is the main class of the client application. It contains the current Sudoku puzzle including: the Sudoku tile information, display location of the grid, start and elapsed time and number of changes made to the puzzle. It provides functionality to: create a Sudoku puzzle, render the puzzle, store and retrieve the puzzle on local disk, respond to user events and detect when the puzzle has been completed.

(d) SudokuTile

The SudokuTile class holds all the information contained within each tile in the Sudoku grid. This consists of the tile's value, if any, and the dummy values (remaining valid digits). This class also holds status and display information.

While the basic objective of providing a simple and elegant graphical user interface was achieved insufficient time was available to create some of the rich tools that were originally envisaged, this includes the comprehensive hints for solving Sudoku puzzles.

5. *To create a database of completed Sudoku grids which can be used as the basis for Sudoku puzzles.*

   Fully Completed - A database of completed Sudoku grids was created and stored in the mySQL database. This was achieved by the place_cell function. This function is called recursively to fill the entire grid. A list of possible digits is maintained for each tile. One digit is selected for each tile and various tests are performed to ensure that this digit is valid. If this digit is valid it is written into the Sudoku grid, otherwise the algorithm backtracks, selects the next digit and continues until completion. This algorithm always terminates with a valid Sudoku grid.

6. *To provide, maintain and display a high scores table (moves/mistakes/time taken) so that users can track their puzzle solving performance.*

   Fully completed - This was achieved by using a client server architecture with the high scores table stored in the mySQL database on the server. In addition functions were written (make new top 5, upload high score, upload new high score, check user high score, get high scores & initialise high scores) in the client to manage and display the high score table.

7. *To provide a consistent scoring method for similar Sudoku puzzles.*

   Fully Completed - Similar Sudoku puzzles were created by random trivial transformations of centrally created completed Sudoku grids. The software that creates the Sudoku grids ensures that they are unique by enumerating all trivial transformations of these grids and comparing them with grids already stored in the database. This comparison is done using a 4 level tree structure in the Node class. The Sudoku puzzle is created on the client application by randomly removing digits but checking that the puzzle is still solvable using the Sudoku solving techniques described above, this ensures that the difficulty level of the puzzle is consistent for different users. A separate high scores table is stored for each of the unique Sudoku grids which ensures that the high score table is a function of the difficulty of the Sudoku puzzle.

4.2 Future Improvements

This section describes potential future improvements for the Sudoku maker/solver. From the research of other Sudoku maker/solvers available on the internet, there are a wide variety of future improvements that could be suggested. The priority of these improvements should be determined by user feedback from their experience of using the application.

1. *Puzzle solving hints*

   As the application can solve the Sudoku puzzle, in-context hints could be provided to the user, throughout the puzzle solving process. These hints would provide the next logical step in solving the puzzle and would potentially teach the user new Sudoku solving techniques. However, if hints are provided to the user, the high scores functionally would not be used and the user would be warned.

2. *Include additional Sudoku puzzle solving techniques*

   The current application includes 9 puzzle solving techniques. However, the research in this report shows that a number of more complex solving techniques exist. These could be easily incorporated into the current application to create a richer, more complex set of Sudoku puzzles. To graduate the level of difficulty of Sudoku puzzles a difficulty level could be established for each of the puzzle solving techniques. The user could then select a level of difficulty for the Sudoku puzzle appropriate for them.

3. *Improvements to Graphical User Interface*

   Based on user feedback through survey, word of mouth and other sources, specific changes to the user interface could be defined. This could include:

   - Addition of a background image
   - Shape of graphical items(style)
   - Number highlighting
   - Other items found from alternate Sudoku websites

4. *Cross Platform Support*

   The initial version of the software was written as a client server application in python. This could be ported to the web and/or to mobile environment as many users are using mobile devices.

5. *Support for Multiple Sudoku Type Puzzles*

   There are a large number of variants of the basic Sudoku puzzle, these include: Mini Sudoku, Killer Sudoku, Alphabetical Sudoku, Kaodoku and Hypersudoku [1]. While the functionality to support these different types of Sudoku puzzle would have to be written the underlying Suoku solving logic is the same. Therefore, the software that has been written could be extended to support these different versions of Sudoku.

6. *History for Each User*

   To observe the development of Sudoku solving skills for a particular user, a history of the user's performance in solving Sudoku puzzles would be kept on users device. This could be shown to the user on demand.

7. *Monetisation*

   To generate revenue from this application either ads could be shown or a small subscription fee could be charged. These changes would have to be carefully introduced so that the users were not alienated from the service.

[1] https://en.wikipedia.org/wiki/Sudoku

5 Glossary

Dummy Values

Dummy Values are a list of the remaining possible digits for a tile.

Sub-grid

A 3x3 grid of tiles that does not contain duplicate digits.The sub-grids are located on the boundaries of the grid and in the centre.

Sudoku grid

A 9x9 grid of tiles.

Sudoku puzzle

A Sudoku puzzle is a Sudoku Grid that has some tiles with digits set. Only one arrangement of additional digits can create a valid completed Sudoku grid.

Techniques

A technique is a single method of solving a Sudoku puzzle, such as Last Remaining Cell in a Row[2].

Tile

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

Trivial Transformations

A Trivial Transformation of a Sudoku grid is a simple re-arrengement of the digits in the grid that maintains the level of complexity of the grid for example rotation by 90°.

[2] http://www.sudokuwiki.org/Getting_Started