Critical Evaluation, Sudoku

Algorithms & Data Structures

SET08122

Andrew Dickinson
Matriculation: 40538519
Word | Page count: 1107 | 2.5
(excl. references & appendices)
Submission date: 27th April 2023

Table of Contents

Introduction

This report sets out to reflect upon and critically evaluate the ‘Sudoku’ program, developed as a project for Algorithms and Data Structures, a module at Edinburgh Napier University. The project was to design and implement a command-line only version of a Sudoku game. As mentioned in the design report (Dickinson, 2023), this is my first program in C (the beginning of the trimester was spent self teaching), so any feedback regarding coding style especially C related would be much appreciated.

The features successfully implemented are as follows:

- Main menu.
- Settings menu.
- Generate a variety of game board sizes (5 available), in a reasonable time.
- One unique solution per game.
- Logical solution to all puzzles (no guessing if using candidates).
- Take input from user and check valid.
- Undo / Redo functionality.
- Colour aid feature.
- Varying difficulty levels (easy, medium, hard).
- Save / load games.
- Recognize a completed board.

Critical Evaluation

Adaptive for size:

From the beginning, the code has been designed to be adaptive for grid sizes. All of the functions to generate a board are based off the global variable SIZE which is set when the menu function is called when the game starts, and changed from the board size menu. Grids can be produced for 4x4, 6x6, 9x9, 12x12 or 16x16 sizes (Appendix A).

Grid generation:

The grids are stored in a 2D array, the code to generate the 2D arrays in C has been adapted from (Multidimensional arrays in c, n.d). Zeros are stored in the grid to denote empty cells. These are output when displaying the board as underscore characters to assist readability of the game board. Structs are used to store potential candidate values for each row, column and box. These are doubly linked lists pointing at the next row / column / box. The attribute ‘cand_list’ is a pointer to another struct, which is also a doubly linked list of all the potential values for a particular row / column / box.

To start with, the first box (index 0) is randomly populated (by forwarding the candidate pointer); this ensures each gameboard is different, producing a variety of games. The remaining cells are populated by looking at a cells row, column and box candidates and iterating over the lists - shortest first. It was the initial intention to populate on the diagonal, however, this produced a bug which on occasion meant boxes could be populated in a way to make a cell not have a solution. Appendix B shows the test output of the solution grid after populating on the diagonal, note there is no possible value for C2 (among others). Populating only the first box randomly appears to have overcome this problem. The remaining boxes are populated with candidate matches using a backtracking algorithm to obtain a completed solution_grid. Due to only using candidate matches, the values are never required to be checked if valid (because if the candidates match for row / column / box, it must be valid).

Once fully populated, the numbers are removed to create the original_player_grid. After each value is removed from the completed grid, a timer is started and the board is attempted to be solved for all remaining candidates for the cell, except the value removed. If a solution is found, it can't have a unique solution, so the number is returned to the grid and that cell is not checked again. The frequency of remaining number values are maintained so only 1 value at max can be reduced to 0 clues. This ensures there is only one unique solution.

User input:

At every point the user is requested for input (menus/gameplay etc.), the input is validated. This is done for the most part by using a Boolean variable to stay in a loop until valid and outputting a feedback message to the user if incorrect. Cells displayed in white cannot be modified by the user. These are original values and the original_player_grid is checked for 0 before accepting a users input.

Undo/Redo:

After every move, the grid references along with inserted and previous values are entered into a stack. When undo is performed, this is popped from undo, and pushed to redo. If the stacks are empty, then no undo or redo action is permitted. When a new move is performed, the redo stack is reset.

Save/Load:

Game variable data is logged to a txt file (Appendix C), automatically named by date and time. The user can select files from the menu based on the menu number.

When the game save or load menus are accessed, the save-files directory is checked for, and created if it does not exist. Additionally, if there have been changes made to a game, the user will be prompted when attempting to quit without saving.

Colour-aid:

The colour aid feature (defaults to off), has been added to check grid values during the game. When off, values which can be modified are displayed in yellow. When on, values are either red – incorrect, or green – correct. This works by checking the value in the player_grid against the value in the solution_grid for correctness, or if the value in the cell in question in the original_player_grid is 0 for modification.

Varying difficulties:

The difficulties (easy, medium and hard) are based on the target number of values to remove from the completed grid when generating a game. The target values for each game board and difficulty can be seen in Appendix D.

Completed game:

A completed game is recognized by maintaining an integer variable for the remaining number of empty cells. When this reaches zero, the player_grid is validated against the solution_grid. If they match, then a "Congratulations" message is output to the user before being able to return to the main menu by pressing enter.

Recommendations

Due to time restraints, the following was not possible to implement, however they would make good extensions to the program:

- Play against a clock / timer. This has been implemented in so far as the options in the menu are availiable (though commented out in user interface line 89 and line 138 along with changing 'max' parameter line 101 and line 150).
- Watch playthrough. This has also been implemented in the menu options, but not further (line 40, along with modifying 'max' parameter line 52).
- Delete file function. This would complete file access allowing files to be removed from within the game.
- Simplify functions / refactor. Some of the functions are long and potentially too complicated. Time has been a factor in being able to revisit and correct this.

Overall, I feel the project is a success, and the game looks and functions well.

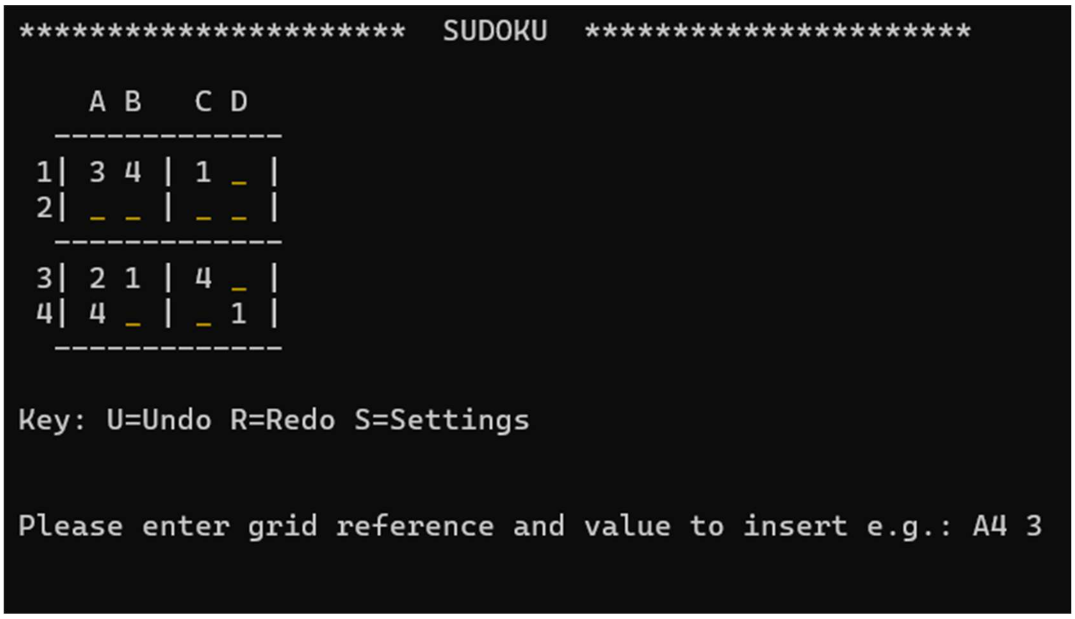
References

1. Dickinson, A. (2023). *Initial Design; Sudoku in C*. (Edinburgh Napier University: Algorithms and Data Structures – SET08122).
2. No name or date. *Multidimensional arrays in C*. www.Uio.no/studier/emner/matnat/ifi/IN3200/v19/teaching-material/multidimarrays.pdf

Appendices

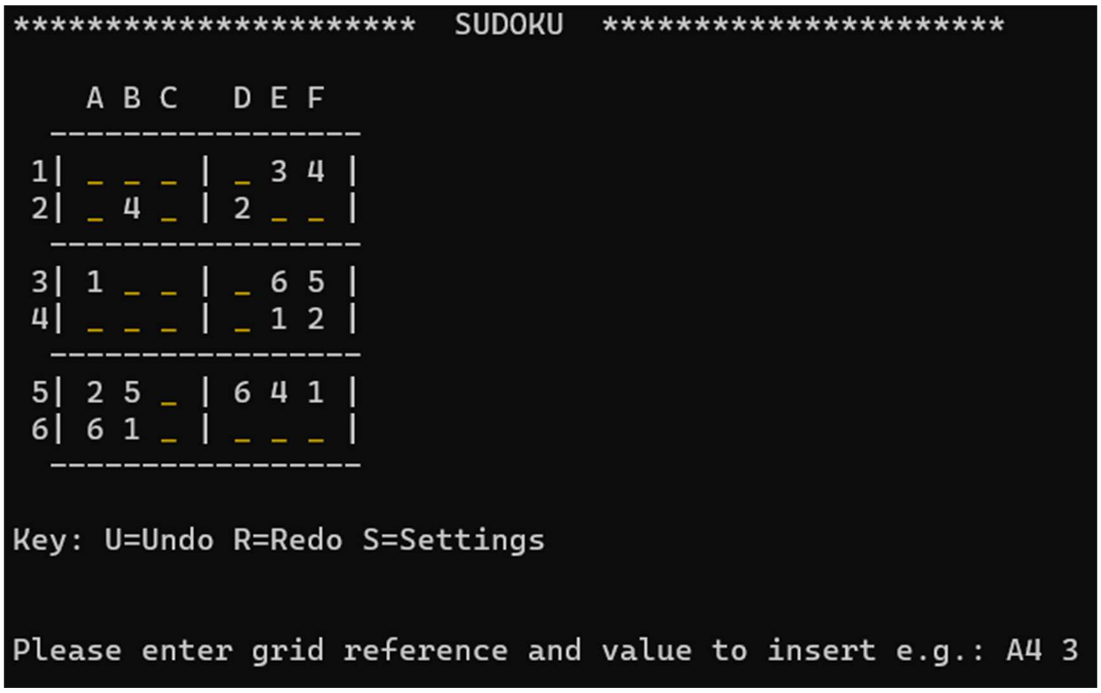
Appendix A – Board sizes.

4x4



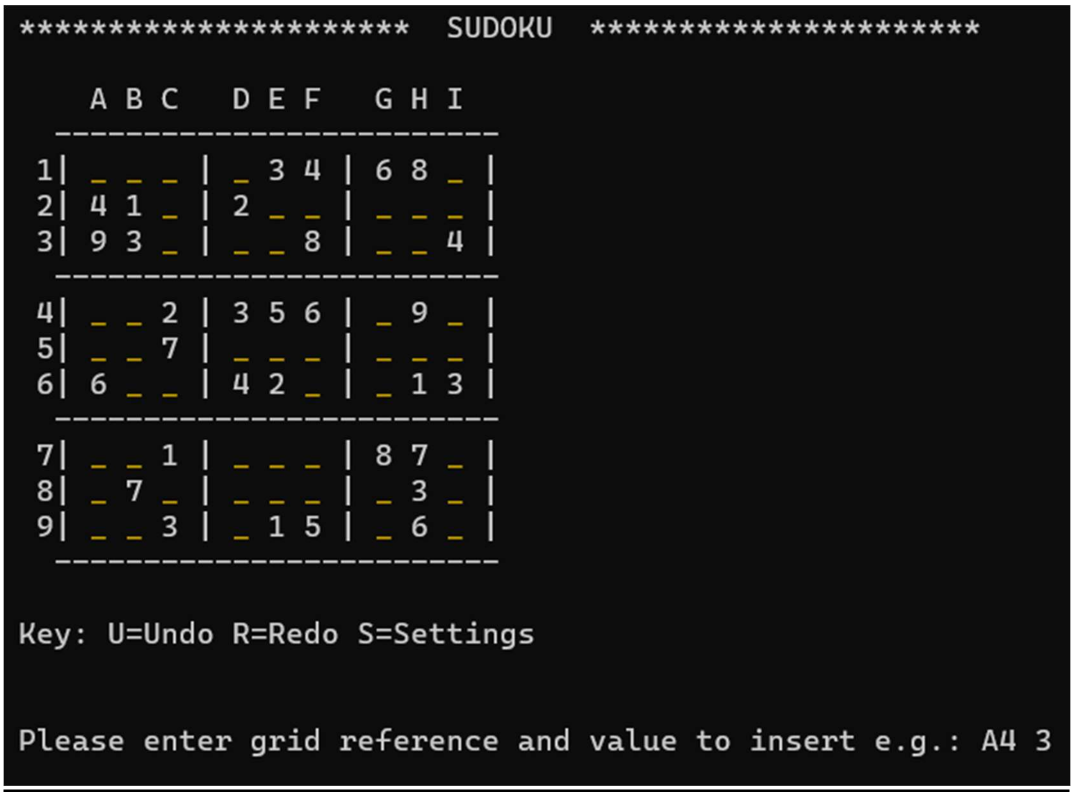
```
**********************  SUDOKU  **********************

    A B   C D
  -------------
 1| 3 4 | 1 _ |
 2| _ _ | _ _ |
  -------------
 3| 2 1 | 4 _ |
 4| 4 _ | _ 1 |
  -------------


Key: U=Undo R=Redo S=Settings


Please enter grid reference and value to insert e.g.: A4 3
```

6x6



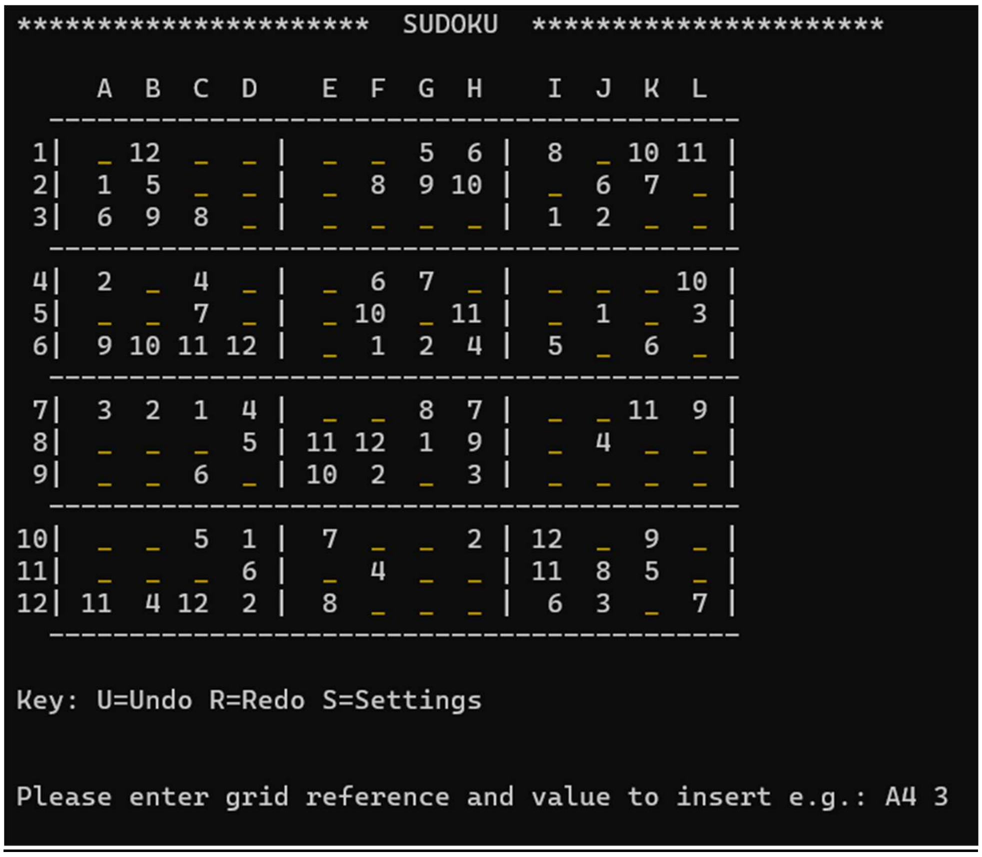
```
**********************  SUDOKU  **********************

    A B C   D E F
  -----------------
 1| _ _ _ | _ 3 4 |
 2| _ 4 _ | 2 _ _ |
  -----------------
 3| 1 _ _ | _ 6 5 |
 4| _ _ _ | _ 1 2 |
  -----------------
 5| 2 5 _ | 6 4 1 |
 6| 6 1 _ | _ _ _ |
  -----------------

Key: U=Undo R=Redo S=Settings


Please enter grid reference and value to insert e.g.: A4 3
```

9x9



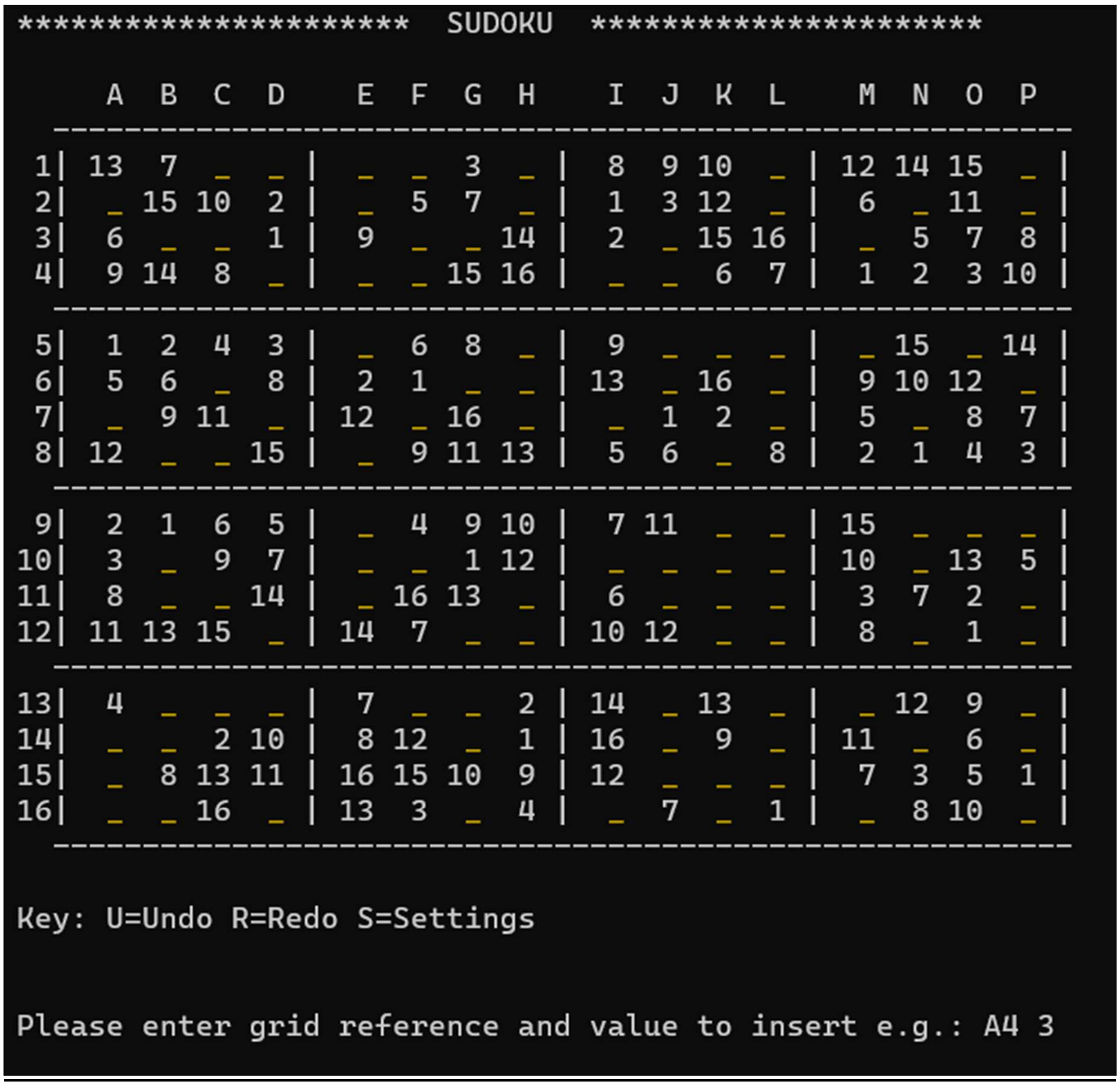
```
**********************   SUDOKU   **********************

    A B C   D E F   G H I
  -------------------------
 1| _ _ _ | _ 3 4 | 6 8 _ |
 2| 4 1 _ | 2 _ _ | _ _ _ |
 3| 9 3 _ | _ _ 8 | _ _ 4 |
  -------------------------
 4| _ _ 2 | 3 5 6 | _ 9 _ |
 5| _ _ 7 | _ _ _ | _ _ _ |
 6| 6 _ _ | 4 2 _ | _ 1 3 |
  -------------------------
 7| _ _ 1 | _ _ _ | 8 7 _ |
 8| _ 7 _ | _ _ _ | _ 3 _ |
 9| _ _ 3 | _ 1 5 | _ 6 _ |
  -------------------------

Key: U=Undo R=Redo S=Settings


Please enter grid reference and value to insert e.g.: A4 3
```

12x12



```
**********************   SUDOKU   **********************

     A  B  C  D    E  F  G  H    I  J  K  L
   ---------------------------------------------
 1|  _ 12  _  _ |  _  _  5  6 |  8  _ 10 11 |
 2|  1  5  _  _ |  _  8  9 10 |  _  6  7  _ |
 3|  6  9  8  _ |  _  _  _  _ |  1  2  _  _ |
   ---------------------------------------------
 4|  2  _  4  _ |  _  6  7  _ |  _  _  _ 10 |
 5|  _  _  7  _ |  _ 10  _ 11 |  _  1  _  3 |
 6|  9 10 11 12 |  _  1  2  4 |  5  _  6  _ |
   ---------------------------------------------
 7|  3  2  1  4 |  _  _  8  7 |  _  _ 11  9 |
 8|  _  _  _  5 | 11 12  1  9 |  _  4  _  _ |
 9|  _  _  6  _ | 10  2  _  3 |  _  _  _  _ |
   ---------------------------------------------
10|  _  _  5  1 |  7  _  _  2 | 12  _  9  _ |
11|  _  _  _  6 |  _  4  _  _ | 11  8  5  _ |
12| 11  4 12  2 |  8  _  _  _ |  6  3  _  7 |
   ---------------------------------------------

Key: U=Undo R=Redo S=Settings


Please enter grid reference and value to insert e.g.: A4 3
```

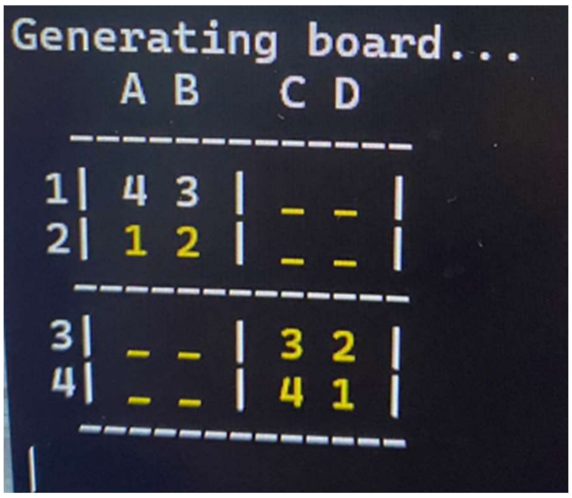
16x16



```
**********************  SUDOKU  **********************

     A  B  C  D    E  F  G  H    I  J  K  L    M  N  O  P
   ---------------------------------------------------------
  1| 13  7  _  _ |  _  _  3  _ |  8  9 10  _ | 12 14 15  _ |
  2|  _ 15 10  2 |  _  5  7  _ |  1  3 12  _ |  6  _ 11  _ |
  3|  6  _  _  1 |  9  _  _ 14 |  2  _ 15 16 |  _  5  7  8 |
  4|  9 14  8  _ |  _  _ 15 16 |  _  _  6  7 |  1  2  3 10 |
   ---------------------------------------------------------
  5|  1  2  4  3 |  _  6  8  _ |  9  _  _  _ |  _ 15  _ 14 |
  6|  5  6  _  8 |  2  1  _  _ | 13  _ 16  _ |  9 10 12  _ |
  7|  _  9 11  _ | 12  _ 16  _ |  _  1  2  _ |  5  _  8  7 |
  8| 12  _  _ 15 |  _  9 11 13 |  5  6  _  8 |  2  1  4  3 |
   ---------------------------------------------------------
  9|  2  1  6  5 |  _  4  9 10 |  7 11  _  _ | 15  _  _  _ |
 10|  3  _  9  7 |  _  _  1 12 |  _  _  _  _ | 10  _ 13  5 |
 11|  8  _  _ 14 |  _ 16 13  _ |  6  _  _  _ |  3  7  2  _ |
 12| 11 13 15  _ | 14  7  _  _ | 10 12  _  _ |  8  _  1  _ |
   ---------------------------------------------------------
 13|  4  _  _  _ |  7  _  _  2 | 14  _ 13  _ |  _ 12  9  _ |
 14|  _  _  2 10 |  8 12  _  1 | 16  _  9  _ | 11  _  6  _ |
 15|  _  8 13 11 | 16 15 10  9 | 12  _  _  _ |  7  3  5  1 |
 16|  _  _ 16  _ | 13  3  _  4 |  _  7  _  1 |  _  8 10  _ |
   ---------------------------------------------------------

Key: U=Undo R=Redo S=Settings


Please enter grid reference and value to insert e.g.: A4 3
```

Appendix B – Bug during development.

Fixed by changing to populating only the 1st box (index 0) rather than on the diagonal.



```
Generating board...
    A B   C D
  -------------
1| 4 3 | _ _ |
2| 1 2 | _ _ |
  -------------
3| _ _ | 3 2 |
4| _ _ | 4 1 |
  -------------
```

Appendix C – Save file line representation

The file lines represent variables as shown below:

- SIZE
- Difficulty setting
- Colour-aid setting
- Remaining grid_empties
- Undo_stack memory required to be allocated
- Undo_stack top
- Undo_stack moves – 4 per move; row, column, insert_value, previous_value
- Redo_stack memory required to be allocated
- Redo_stack top
- Redo_stack moves – 4 per move; row, column, insert_value, previous_value
- Solution_grid values – listed left to right row by row
- Player_grid values – listed left to right row by row
- Original_player_grid values – listed left to right row by row

Appendix D – Target values to be removed when generating a board

The target values to be removed for each game board and difficulty during generation can be seen below:

- Easy
  - 4x4; remove 6
  - 6x6; remove 15
  - 9x9; remove 49
  - 12x12; remove 60
  - 16x16; remove 90
- Medium
  - 4x4; remove 8
  - 6x6; remove 20
  - 9x9; remove 53
  - 12x12; remove 70
  - 16x16; remove 105
- Hard
  - Attempts to remove the maximum number (All cells are attempted to remove and check for solution within 20ms per cell). Only 1 number can be reduced to 0 clues remaining.