GRAPHICAL USER INTERFACE BASED SOLVER FOR SU-DO-KU PUZZLES WITH MULTIPLE SOLUTIONS

[1]MOHAMED ASIF, [2]MOHAMED RASEEN

[1]Student (Bachelor of Engineering – CSE), Aalim Muhammed Salegh College of Engineering, Chennai, Tamil Nadu, India
[2]Programmer, Tamil Nadu, India
Email: [1]mohamedasif74@gmail.com, [2]dr.raseen@gmail.com

Abstract- This paper describes a Graphical User Interface based solver for the SU DO KU puzzles with multiple solutions. If a puzzle has multiple correct solutions, then, manual solving will be almost impossible. The described solver provides one correct solution for puzzles that have multiple solutions. The solver implements the elimination, random assignment and backtracking search algorithm to solve the puzzle. The solver is developed using Visual C++ and it is easy to use. The solver is also powerful enough to detect puzzles that do not have even a single solution, and the user can redesign the puzzle. For solvable puzzles, the solution is found in less than a second, which is and an advantage. The solver is encapsulated in an installer along with some puzzles for users to install and practice. One major advantage is that the solver by itself is only 96 KB and the complete installer is only 140 KB. The solver is uploaded in the internet and can be downloaded and used any time.

Keywords- SU DO KU, Multiple Solution Problems, Backtracking Search.

I. INTRODUCTION

SU DO KU is one of the most popular puzzle games, goal of which is to fill a 9×9 grid with numbers so that each row, column and 3×3 section contain all of the digits between 1 and 9 [1]. Some SU DO KU puzzles have many solutions [2]. Certain puzzles have no solution [2]. Backtracking is a general algorithm for finding all (or some) solutions to some computational problems [3, 4]. Backtracking algorithm incrementally builds candidates to a prospective solution, and immediately abandons the candidate, if it determines that the added candidate cannot possibly lead to a valid solution [3, 4].
This paper explains a Graphical Based Interface (GUI) based SU DO KU solver that implements the elimination, and backtracking search algorithms. Section 2 provides some preliminary information about the topics that the paper implements. Section 3 explains the GUI components of the implemented SU DO KU Solver. It also gives a tutorial about using the GUI. Section 4 describes the algorithm for error checking and solving the puzzle. Section 5 lists the advantages of the solver. Final section concludes the paper.

II. PRELIMINARIES

Figure 1 depicts a sample SU DO KU puzzle. As in figure 1, a SU DO KU puzzle has 2 dimensional arrays of 9 rows and 9 columns. The 81 elements of the array are grouped into 9 squares, of 3 rows and 3 columns each. Initially any number, from 1 to 9, is filled at some random locations of the array. The numbers are filled in such a way that no number is repeated in any row, column or square [1]. For example, consider puzzle in figure 1, it has numbers filled in red color. Number 9 is filled in 2nd row, 3rd column and is present in top-left square. The number 9 is not repeated in 2nd row, 3rd column or in top-left square. Once some locations are filled the puzzle is ready and has to be solved.

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

Fig.1. Sample SU DO KU puzzle

The solution of the puzzle is obtained by filling the blank locations with any number, from 1 to 9, in such a way that no number is repeated in any row, column or square [1]. For example, figure 2 shows the solution for puzzle in figure 1. The solution numbers are in black color. In figure 2, no number is repeated in any row, column or square.
A well formulated SU DO KU puzzle has only one solution. Certain puzzles can have multiple solutions [2]. Figure 3 illustrates another solution for the same puzzle is figure 1. Though solutions in figure 2 and figure 3 look identical, they are different. In figure 3 (compared to figure 2), the numbers 2 and 9 are swapped in middle-left and the middle squares. This

difference is highlighted in figure 3 using dark, black rectangles.

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

Fig.2. Solution for the sample SU DO KU puzzle

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

Fig.3. Another solution for the same SU DO KU puzzle

Since the numbers (2 and 9) are swapped inside the same square they will not be repeated inside the square. Again, since the numbers (2 and 9) are swapped inside the same column they will not be repeated in the column. As the same numbers (2 and 9) are swapped twice (in two different squares, middle-left and middle) but on the same lines (4 and 6), they will not be repeated in the lines. As there is no repetition of numbers in a row, column or square even after swapping, the condition of a valid solution is still met. Hence figure 3 is also a valid solution for puzzle in figure 1.

It is also possible to have puzzles that do not have any valid solution. For example, consider the puzzle in figure 4. The number 5 is present in row 1 (highlighted in top-left square) and row 2 (top-middle square). Hence the number 5 cannot be filled in blank locations in rows 1 and 2. Ultimately, in the top-right square, number 5 cannot be filled in rows 1 and 2 and row 3 is fully occupied (with 1, 7 and 4). Hence number 5 cannot be filled anywhere is the top-right square, which make the puzzle impossible to solve.

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

Fig.4. SU DO KU puzzle with no solution

The complexity in finding a solution for a SU DO KU puzzle depends on the number of valid solutions. In case of puzzle in figure 1, two solutions (figures 2 and 3) were shown. Only two numbers (2 and 9) were involved in the difference between the solutions. If number 2 was chosen first, for the solution, then obviously number 9 would fill be blank location later, and vice-versa. If the number of choices available to fill a blank location was 6 and there were 20 blank locations, a random selection of choice may not lead to correct solution.

For instance, if a random choice is made, it may lead to a situation as in Figure 4, and choice has to be reverted. As the number of valid solutions increases the number of choices to fill the blank location increases, which in turn increases the complexity to find a solution. This process of randomly selecting a choice and reverting (the choice, if incorrect) is the backtracking algorithm.

III. GRAPHICAL USER INTERFACE OF SU DO KU SOLVER

Figure 5 illustrates the GUI of the SU DO KU solver. A new puzzle is started by clicking the "NEW PUZZLE" button, marked as 'A' in figure 5. Initially all 81 location are blank. To fill a location by a number, select the corresponding number in the list marked as 'B'. For example, in figure 5, number 4 was selected and hence only square with number 4 is highlighted.

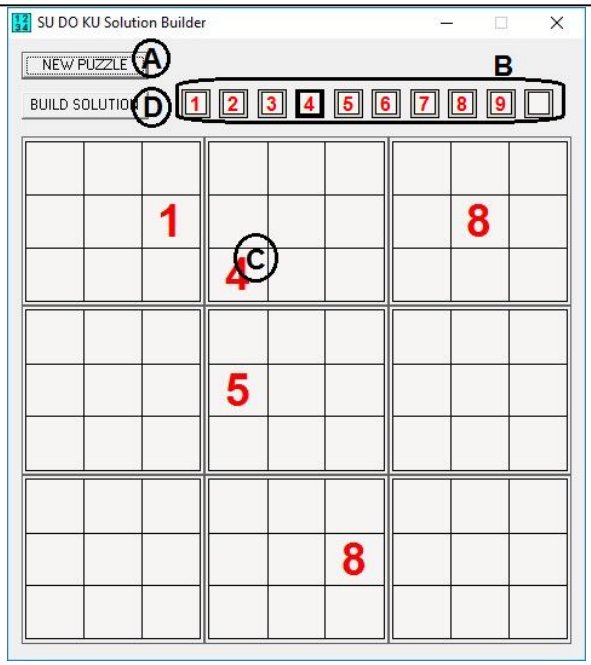


Fig.5. GUI of SU DO KU solver

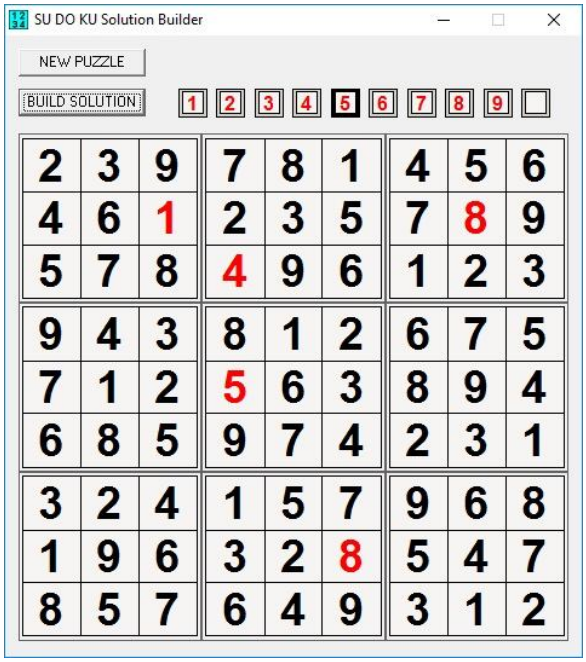


Fig.6. Solution generated by the SU DO KU solver

After selecting a number, click on any 81 locations and the selected number will be displayed in the location. As an example, in figure 5, number 4 was selected and location in row 3, column 4 was clicked (marked as 'C') and the number 4 is displayed in that location in red color. Same procedure can be used to overwrite a location with a different number. If a location needs to be cleared, then select the blank square, which is rightmost in the list marked as 'B', and then click on the location. Once all the elements of the puzzle are entered and have to be solved, click on the "BUILD SOLUTION" button marked as 'D'. The puzzle will be solved and the solution will be displayed as in figure 6. In figure 6, the puzzle input numbers are in red and the blank spaces filled by the solver are in black.

The solver also checks for errors in input such are repeated numbers in a row, column or square. If there are repeated number in a row, error dialog similar to figure 7 is displayed. A dialog similar to figure 8 is displayed for a repeated number in a column. For a repeated number in a square, a dialog similar to figure 9 is displayed. If there are no repeated numbers in the input, but the puzzle has no solution as in figure 4, an error dialog as in figure 10 is displayed.

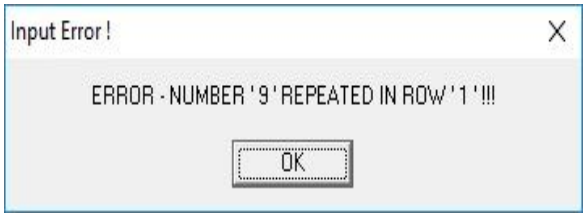


Fig.7. Input Error Indication – Number repeated in a Row

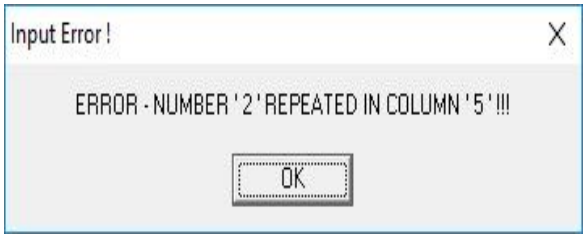


Fig.8. Input Error Indication – Number repeated in a Column

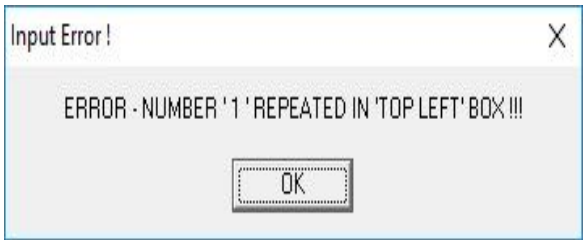


Fig.9. Input Error Indication – Number repeated in a Square

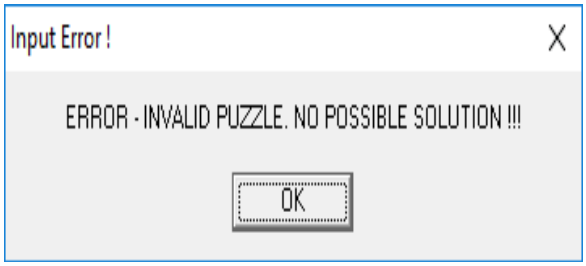


Fig.10. Input Error Indication – Puzzle with no Solution

IV. ALGORITHM OF THE SU DO KU SOLVER

The solver initially checks for duplication of numbers is rows, columns or squares. This is easy and straight forward. Then the solver proceeds to solve the puzzle. There are several sub-routines that are invoked while finding the solution. Figures 11, 12, 13 and 14 illustrate the "Initialization", "Recursive Elimination", "Solution Correctness Verification" and "Non Existence of Solution" Sub-Routines respectively. Figure 15 illustrate "The Main Solver".

1. *Create a 9 x 9 x 9 (3 Dimension) array to store solution of the puzzle. (9 x 9 is used for 81 locations and third dimension to store the 9 possible numbers in that location).*
2. *The numbers provided as 'Inputs' to the puzzle at different locations (rows and columns) cannot change; hence copy them at corresponding locations in the solution array (These locations use only one space in the third dimension).*
3. *Blank locations of the puzzle can end up with any number from 1 to 9, hence initialize the third dimension elements of solution array (for blank locations) to values 1 to 9 (All possible outcomes).*
4. *Set a Boolean variable 'Assumption Started' to 'False'*

Fig.11. Initialization Sub-Routine

1. *If any location in the solution array has only 'one' possible combination (say N) in the 3rd dimension, it has to be the final number; hence the other locations in same rows, columns and squares cannot take that value. Scan through all the locations in the same row, column and square and remove N, if present in their possible combinations.*
2. *After previous step some combinations (of possible outcome) in some locations may reduce, and some may end up with just one combination.*
   a. *If any one location ends up with one combination, then finalize it and go to step 1 (Since this finalization may further reduce other locations).*
   b. *If no location ends up with one combination, then return to the caller.*

Fig.12. Recursive Elimination Sub-Routine

1. *If all locations have only 'one' possible combination of outcome number, then proceed to next step, else return 'Solution Incorrect' to the caller.*
2. *If any number is repeated in any 'Row', then return 'Solution Incorrect' to the caller, else proceed to next step.*
3. *If any number is repeated in any 'Column', then return 'Solution Incorrect' to the caller, else proceed to next step.*
4. *If any number is repeated in any 'Square', then return 'Solution Incorrect' to the caller, else proceed to next step.*
5. *Return 'Solution Correct' to the caller.*

Fig.13. Solution Correctness Verification Sub-Routine

After the recursive reduction subroutine, some outcome combinations might be eliminated from certain locations.

1. *If there is any location that has 'No' possible combinations of outcome, then, return 'No Possible Solution' to the caller.*
2. *Else return 'Solution Still Possible' to the caller.*

Fig.14. Non Existence of Solution Sub-Routine

1. *Call the 'Initialization Sub-Routine'*
2. *Call the 'Recursive Elimination Sub-Routine'*
3. *Call the 'Solution Correctness Verification Sub-Routine'.*
   a. *If the Sub-Routine Call returns 'Success in Solution', then print 'Solution' and Quit.*
   b. *Else Continue to next Step*
4. *Call the 'Non Existence of Solution Sub-Routine'.*
   a. *If the Sub-Routine Call returns 'No possible Solution', then print 'Error – No possible Solution' and Quit.*
5. *At this position, some locations should have more than 'One' combinations for possible outcome, but the 'Recursive Elimination Sub-Routine' will not re reduce further.*
   a. *If the Boolean Variable 'Assumption Started' is 'False' then, begin assuming that one of many combinations of 'Possible outcomes' for locations will lead to a correct solution.*
      i. *Set the Boolean 'Assumption Started' to 'True'*
      ii. *Record the current 'Solution Array'*
      iii. *Assume that any one possible combination of outcome will lead to correct solution, record it in 'Solution Array' and jump to 'Step 2'*
   b. *At this position 'Assumption has Begun,' But still 'Solution is not Obtained', hence*
      i. *Backtrack the assumption and move to previously recorded solution*
      ii. *If 'All possible combinations' of Assumptions are 'Exhausted', then, 'Print - No Possible Solution' and 'Exit'*
      iii. *Else Assume the 'Next' Possible outcome combination as correct solution and Jump to 'Step 2'*
6. *Exit the Program*

Fig.15. The Main Solver

ADVANTAGES

The advantages of the SU DO KU solver are,

1. It is powerful enough to detect puzzles that do not have a solution.
2. If a puzzle has at least one solution, the solution is found in less than a second.
3. Solver also comes with a bunch of puzzles for the user to practice solving them.
4. Solver is encapsulated in an installer along with the practice puzzles.
5. The size of complete installer is just 140 KB.
6. The standalone solver executable file is only 96 KB, which is very less compared to other applications.
7. The installer has been uploaded in the internet, and can be downloaded from the link - https://sites.google.com/site/drraseen/my-software-projects/SU_DO_KU_Install.rar

CONCLUSIONS

This paper explained a GUI based solver for SU DO KU puzzles with multiple solutions. Method to use the GUI was discussed. Algorithm used by the solver in deriving the solution was elaborated. Finally, several advantages of the SU DO KU solver were listed. The ease of use of GUI and advantages of the solver proves it to be useful tool.

ACKNOWLEDGMENTS

The authors wish to express their gratitude to Mr. Mohamed Reyas (EDRC Global Computers – Dubai, UAE) for sponsoring and making travel arrangements.

REFERENCES

1. Felgenhauer, Bertram, and Frazer Jarvis. "Mathematics of Sudoku I," Mathematical Spectrum, vol.39, no.1, pp.15-22, 2006.
2. Zhai, Gaoshou, and Junhong Zhang. "Solving Sudoku Puzzles Based on Customized Information Entropy," International Journal of Hybrid Information Technology, vol.6, no.1, pp.77-91, 2013.
3. Van Beek, Peter. "Backtracking search algorithms," Foundations of Artificial Intelligence, vol.2, pp.85-134, 2006.
4. Website on Backtracking search algorithm – https://en.wikipedia.org/wiki/Backtracking

★★★