Indian Institute of Technology, Bombay

CS 101: Computer Programming and Utilisation

Project Report

Sudoku Auto-solver

B Chaitanya Rajesh	140050073
Chitraang Murdia	140050023
K V N Sreenivasulu	140050078

Slot: 11 Group: 11

Table of Contents

Abstract

This projects aims to design a C++ implementation of a Sudoku Auto-solver.

Sudoku Auto-solver is a program that can solve a normal Sudoku of any degree of toughness as desired by the user.

Our program can also solve its variants like diagonal sudoku, window sudoku and jigsaw sudoku.

We have included certain other features like displaying only one more number tile and checking a solution given by the user.

The program can also be used to check if a given Sudoku has a solution or not.

Acknowledgements

We would like to express the deepest gratitude to our instructors Professor Deepak B Phatak and Professor Supratik Chakraborty, who have shown the attitude and the substance of a genius: they continually and persuasively conveyed a spirit of adventure in regard to the subject matter, and an excitement in regard to teaching. Without their supervision and constant help this project would not have been possible.

We are also highly indebted to Miss Firuza Aibara and Mister Nagesh Karmali for their guidance and constant support. Their willingness to motivate us contributed tremendously to our project.

We would also like to thank to our CLTA , Miss Prerna Gupta for the valuable guidance and advice. She inspired us greatly to work in this project.

Furthermore we would like to acknowledge with much appreciation the crucial role of the staff of CSE department, IIT Bombay who gave the permission to use all required equipment and the necessary materials to complete the project

Besides, we would like to thank the authority of Indian Institute of Technology (IIT), Bombay for providing us with a good environment and facilities to complete this project.

We would also thank the developers of simplecpp.

Finally, an honorable mention goes to all the TAs and friends for their understandings and supports on us in completing this project. Without helps of the particular that mentioned above, we would face many difficulties while doing this.

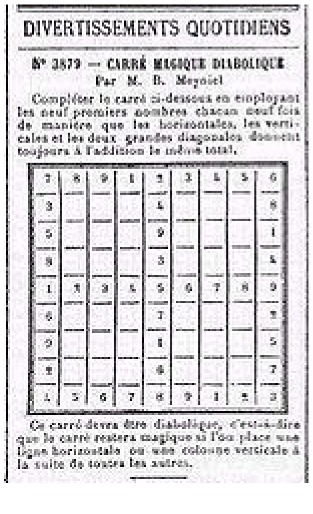
Introduction

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

History of Sudoku

In late nineteenth century number puzzles appeared in newspaper when French puzzle setters began experimenting with removing numbers from magic squares. *Le Siècle*, a Paris-based daily, published a partially completed 9×9 magic square with 3×3 sub-squares on November 19, 1892. It was not a Sudoku because it contained double-digit numbers and required arithmetic rather than logic to solve, but it shared key characteristics: each row, column and sub-square added up to the same number.

On July 6, 1895, *La France*, refined the puzzle so that it was almost a modern Sudoku. It simplified the 9×9 magic square puzzle so that each row, column and broken diagonals contained only the numbers 1-9, but did not mark the sub-squares. Although they are unmarked, each 3×3 sub-square does indeed comprise the numbers 1-9 and the additional constraint on the broken diagonals leads to only one solution.



DIVERTISSEMENTS QUOTIDIENS

N° 3879 — CARRÉ MAGIQUE DIABOLIQUE
Par M. B. Meyniel

Compléter le carré ci-dessous en employant les neuf premiers nombres chacun neuf fois de manière que les horizontales, les verticales et les deux grandes diagonales donnent toujours à l'addition le même total.

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

Ce carré devra être diabolique, c'est-à-dire que le carré restera magique si l'on place une ligne horizontale ou une colonne verticale à la suite de toutes les autres.

From *La France* newspaper, July 6, 1895.

The puzzle was introduced in Japan by Nikoli in the paper Monthly Nikolist in April 1984 as Sūji wa dokushin ni kagiru , which also can be translated as "the digits must be single". At a later date, the name was abbreviated to Sudoku by Maki Kaji taking only the first characters of compound words to form a shorter version.Sudoku is a registered trademark in Japan and the puzzle is generally referred to as Number Place.

General Description of Sudoku

Sudoku is a puzzle game which tests the logical capabilities of the solver. Sudoku is played over a 9x9 grid each element of which is called a tile, divided to 3x3 sub grids or boxes. The objective is to fill a grid with digits so that each column, each row, and each of the nine boxes that compose the grid contains all of the digits from 1 to 9 and each digit appears once.

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

A typical Sudoku puzzle

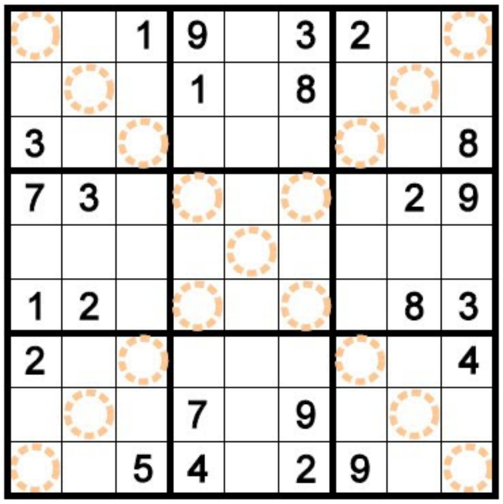
1	6	4	3	2	9	7	8	5
5	2	3	8	7	6	1	4	9
9	7	8	4	5	1	6	3	2
8	9	5	6	4	3	2	1	7
6	4	7	2	1	5	8	9	3
3	1	2	9	8	7	4	5	6
2	3	1	5	6	8	9	7	4
7	5	6	1	9	4	3	2	8
4	8	9	7	3	2	5	6	1

The same puzzle with solution

Variants of Sudoku

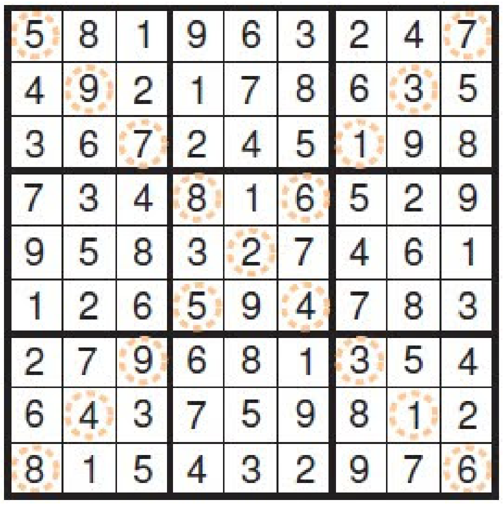
Diagonal Sudoku

Diagonal Sudoku is also played over a 9x9 grid divided to 3x3 sub grids or boxes. The objective is to fill a grid with digits so that each column, each row, and each of the nine boxes that compose the grid contains all of the digits from 1 to 9 and each digit appears once. Also the main diagonals of the grid contains all of the digits from 1 to 9 and each digit appears once.



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

Diagonal Sudoku puzzle

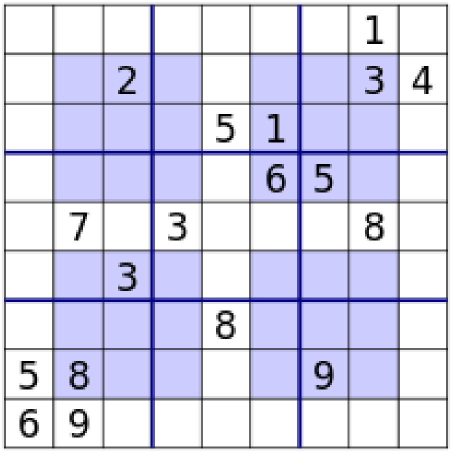


5	8	1	9	6	3	2	4	7
4	9	2	1	7	8	6	3	5
3	6	7	2	4	5	1	9	8
7	3	4	8	1	6	5	2	9
9	5	8	3	2	7	4	6	1
1	2	6	5	9	4	7	8	3
2	7	9	6	8	1	3	5	4
6	4	3	7	5	9	8	1	2
8	1	5	4	3	2	9	7	6

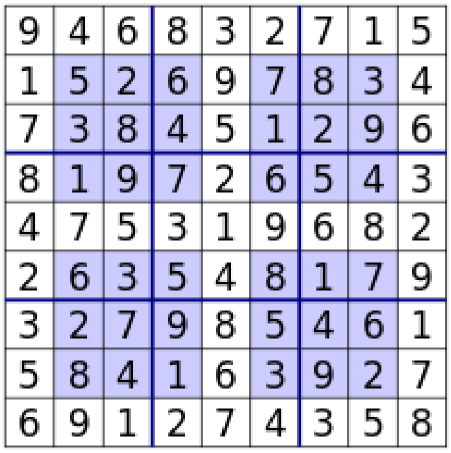
The same puzzle with solution

Window Sudoku (Hypersudoku)

Window Sudoku is also played over a 9x9 grid divided to 3x3 sub grids or boxes. The objective is to fill a grid with digits so that each column, each row, and each of the nine boxes that compose the grid contains all of the digits from 1 to 9 and each digit appears once. Also there are four additional boxes in the grid that contain all of the digits from 1 to 9 and each digit appears once.



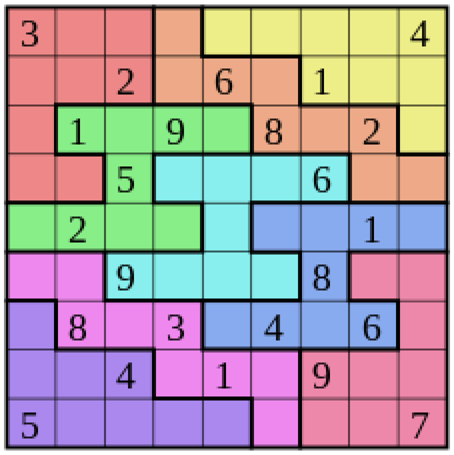
Window Sudoku puzzle



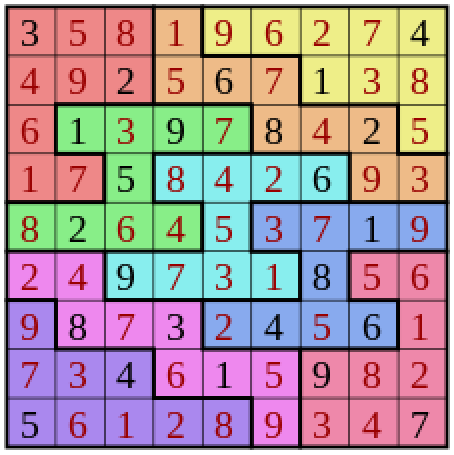
The same puzzle with solution

Jigsaw Sudoku (Nonomino)

Jigsaw Sudoku is also played over a 9x9 grid, divided to nine irregular shaped sub grids or boxes of 9 tiles each. The objective is to fill a grid with digits so that each column, each row, and each of the nine irregular boxes that compose the grid contains all of the digits from 1 to 9 and each digit appears once.



Jigsaw Sudoku puzzle



The same puzzle with solution

Design of Algorithm

The algorithm is based on backtracking that incrementally builds candidates to the solutions, and abandons each partial candidate *c* as soon as it determines that *c* cannot possibly be completed to a valid solution.

The basic algorithm is

1. Set up an iteration that repeats itself till the sudoku is completely solved.
2. Find the first empty square
3. Set up another iteration that repeats for each of the numbers 1 to 9
4. Check if the number is valid i.e. it should not be present in the corresponding row, column or 3x3 box
5. Call the sudoku solving function recursively
6. If at any point the number is invalid or the callee function returns false then go for next step of iteration.
7. If none of the numbers 1 to 9 satisfy above condition then return false
8. Finally if the sudoku gets solved then return true
9. Uniqueness of solution can be checked by filling the numbers in ascending and descending order
10. If the two solutions are same then the problem has a unique solution
11. If true value is returned by the first caller function then print the solved sudoku and if false value is returned then print that given sudoku cannot be solved.

Detailed Algorithm

Here we have given the detailed algorithm with complete details of variables and functions. Minor changes may be done in the course of coding the algorithm.

```
1. main (type int)
    a. Print “What type of sudoku do you wish to solve?”
    b. Give option to solve normal sudoku, diagonal sudoku, window sudoku and jigsaw sudoku by displaying appropriate message
    c. Call the required function
    d. Return the value returned by function

2. Normal (type int)
    a. Declare a 9x9 int array called sudoku_grid
    b. Initialize all elements of sudoku_grid to 0
    c. Declare a local int variable inp
    d. Call int function InputGrid  with formal parameter sudoku_grid and store its value in inp
    e. If inp equals -1 then return -1
    f. Validate sudoku_grid using ValidN, if invalid return -1
    g. Start an iteration
    h. Give option to solve sudoku, show a tile, check sudoku and exit by displaying appropriate message
    i. If user selects to solve sudoku
        i. Declare a int variable sudoku_solve with value obtained by calling function NSudoku with formal parameter sudoku_grid
        ii. If sudoku_solve is 1 then print “Solution is unique” and call function DisplaySudoku with formal parameter sudoku_grid and return 0
        iii. If sudoku_solve is 0 then print “Solution is not unique” and call function DisplaySudoku with formal parameter sudoku_grid and return 0
        iv. If sudoku_solve is -1 then print “No solution exists” and return -2
    j. If user selects to show a tile
        i. Print “specify row and coloumn of tile to be displayed”
        ii. Declare two int variables row and col
        iii. Input values of row and col
        iv. If row or col does not lie between 1 to 9 print “Invalid input” and start next iteration
        v. Declare a 9x9 int array called solution_grid same as sudoku_grid
        vi. Declare an int variable sudoku_solve with value obtained by calling function NSudoku with formal parameter solution_grid
        vii. If sudoku_solve is 1
```

```
                    1. If element in sudoku_grid corresponding to row - 1 and col - 1 is not zero print “Tile already displayed” and start next iteration
                    2. Else copy value of that element from solution_grid to sudoku_grid
                    3. Call function DisplaySudoku with formal parameter sudoku_grid
            viii. If sudoku_solve is 0 then print “Solution is not unique” and return 0
              ix. If sudoku_solve is -1 then print “No solution exists” and return -2
        k. If user selects to check sudoku
               i. Print “Give solution”
              ii. Input and validate solved_grid
             iii. Declare a bool variable check_sol with value obtained by calling function CheckNSudoku with formal parameter sudoku_grid and solved_grid
              iv. If check_sol is true then  print “Solution is correct” and return 0
               v. If check_sol is false then print “Solution is incorrect” and return 1
        l. If user selects to exit return 0

3. Diagonal (type int)
        a. Declare a 9x9 int array called sudoku_grid
        b. Initialize all elements of sudoku_grid to 0
        c. Declare a local int variable inp
        d. Call int function InputGrid  with formal parameter sudoku_grid and store its value in inp
        e. If inp equals -1 then return -1
        f. Validate sudoku_grid using ValidD, if invalid return -1
        g. Start an iteration
        h. Give option to solve sudoku, show a tile, check sudoku and exit by displaying appropriate message
        i. If user selects to solve sudoku
               i. Declare an int variable sudoku_solve with value obtained by calling function DSudoku with formal parameter sudoku_grid
              ii. If sudoku_solve is 1 then print “Solution is unique” and call function DisplaySudoku with formal parameter sudoku_grid and return 0
             iii. If sudoku_solve is 0 then print “Solution is not unique” and call function DisplaySudoku with formal parameter sudoku_grid and return 0
              iv. If sudoku_solve is -1 then print “No solution exists” and return -2
        j. If user selects to show a tile
               i. Print “specify row and coloumn of tile to be displayed”
              ii. Declare two int variables row and col
             iii. Input values of row and col
              iv. If row or col does not lie between 1 to 9 print “Invalid input” and start next iteration
               v. Declare a 9x9 int array called solution_grid same as sudoku_grid
              vi. Declare an int variable sudoku_solve with value obtained by calling function DSudoku with formal parameter solution_grid
            vii.  If sudoku_solve is 1
                    1. If element in sudoku_grid corresponding to row - 1 and col - 1 is not
                       zero print “Tile already displayed” and start next iteration
                    2. Else copy value of that element from solution_grid to sudoku_grid
                    3. Call function DisplaySudoku with formal parameter sudoku_grid
           viii.  If sudoku_solve is 0 then print “Solution is not unique” and return 0
             ix.  If sudoku_solve is -1 then print “No solution exists” and return -2
       k. If user selects to check sudoku
              i.  Print “Give solution”
             ii.  Input and validate solved_grid
            iii.  Declare a bool variable check_sol with value obtained by calling function
                  CheckDSudoku with formal parameter sudoku_grid and solved_grid
             iv.  If check_sol is true then  print “Solution is correct” and return 0
              v.  If check_sol is false then print “Solution is incorrect” and return 1
       l. If user selects to exit return 0

4. Window (type int)
       a. Declare a 9x9 int array called sudoku_grid
       b. Initialize all elements of sudoku_grid to 0
       c. Declare a local int variable inp
       d. Call int function InputGrid  with formal parameter sudoku_grid and store its value in
          inp
       e. If inp equals -1 then return -1
       f. Validate sudoku_grid using ValidW, if invalid return -1
       g. Start an iteration
       h. Give option to solve sudoku, show a tile, check sudoku and exit by displaying
          appropriate message
       i. If user selects to solve sudoku
              i.  Declare an int variable sudoku_solve with value obtained by calling function
                  WSudoku with formal parameter sudoku_grid
             ii.  If sudoku_solve is 1 then print “Solution is unique” and call function
                  DisplaySudoku with formal parameter sudoku_grid and return 0
            iii.  If sudoku_solve is 0 then print “Solution is not unique” and call function
                  DisplaySudoku with formal parameter sudoku_grid and return 0
             iv.  If sudoku_solve is -1 then print “No solution exists” and return -2
       j. If user selects to show a tile
              i.  Print “specify row and coloumn of tile to be displayed”
             ii.  Declare two int variables row and col
            iii.  Input values of row and col
             iv.  If row or col does not lie between 1 to 9 print “Invalid input” and start next
                  iteration
              v.  Declare a 9x9 int array called solution_grid same as sudoku_grid
```

```
        vi. Declare an int variable sudoku_solve with value obtained by calling function WSudoku with formal parameter solution_grid
        vii. If sudoku_solve is true
            1. If element in sudoku_grid corresponding to row - 1 and col - 1 is not zero print “Tile already displayed” and start next iteration
            2. Else copy value of that element from solution_grid to sudoku_grid
            3. Call function DisplaySudoku with formal parameter sudoku_grid
        viii. If sudoku_solve is 0 then print “Solution is not unique” and return 0
        ix. If sudoku_solve is -1 then print “No solution exists” and return -2
    k. If user selects to check sudoku
        i. Print “Give solution”
        ii. Input and validate solved_grid
        iii. Declare a bool variable check_sol with value obtained by calling function CheckWSudoku with formal parameter sudoku_grid and solved_grid
        iv. If check_sol is true then  print “Solution is correct” and return 0
        v. If check_sol is false then print “Solution is incorrect” and return 1
    l. If user selects to exit return 0


5. Jigsaw (type int)
    a. Declare an object type jig consisting of 9x9 int array called box, a 9x9 int array called grid and a 9x10 bool array called present
    b. Declare a jig variable named sudoku
    c. Initialize all elements of sudoku.box and sudoku.grid to 0 and sudoku.present to false
    d. Call int function InputBox  with formal parameter sudoku.box
    e. If InputBox returns -1 then return -1
    f. Call bool function ValidBox  with formal parameter sudoku.box
    g. If ValidBox returns false then return -2
    h. Declare a local int variable inp
    i. Call int function InputGrid  with formal parameter sudoku.grid and store its value in inp
    j. If inp equals -1 then return -3
    k. Validate sudoku.grid using ValidJ, if invalid return -3
    l. Call bool function InputPresent  with formal parameter sudoku
    m. If InputPresent returns false then return -4
    n. Start an iteration
    o. Give option to solve sudoku, show a tile, check sudoku and exit by displaying appropriate message
    p. If user selects to solve sudoku
        i. Declare an int variable sudoku_solve with value obtained by calling function JSudoku with formal parameter sudoku
        ii. If sudoku_solve is 1 then print “Solution is unique” and call function DisplaySudoku with formal parameter sudoku.grid and return 0
```

```
            iii. If sudoku_solve is 0 then print “Solution is not unique” and call function
                 DisplaySudoku with formal parameter sudoku.grid and return 0
            iv.  If sudoku_solve is -1 then print “No solution exists” and return -2
        q. If user selects to show a tile
            i.    Print “specify row and column of tile to be displayed”
            ii.   Declare two int variables row and col
            iii.  Input values of row and col
            iv.   If row or col does not lie between 1 to 9 print “Invalid input” and start next
                  iteration
            v.    Declare a jig variable called solution same as sudoku
            vi.   Declare an int variable sudoku_solve with value obtained by calling function
                  JSudoku with formal parameter solution
            vii.  If sudoku_solve is 1
                      1. If element in sudoku.grid corresponding to row - 1 and col - 1 is not
                         zero print “Tile already displayed” and start next iteration
                      2. Else copy value of that element from solution.grid to sudoku.grid
                      3. Change the value of element in sudoku.present corresponding to
                         required box and value to true
                      4. Call function DisplaySudoku with formal parameter sudoku.grid
            viii. If sudoku_solve is 0 then print “Solution is not unique” and return 0
            ix.   If sudoku_solve is -1 then print “No solution exists” and return -2
        r. If user selects to check sudoku
            i.    Print “Give solution”
            ii.   Declare a jig variable named solved
            iii.  Initialize all elements of solved.grid to 0 and solved.present to false
            iv.   Copy sudoku.box to solved box
            v.    Declare a local int variable inp
            vi.   Call int function InputGrid  with formal parameter solved.grid and store its
                  value in inp
            vii.  If inp equals -1 then return -3
            viii. Call bool function InputPresent  with formal parameter solved
            ix.   If InputPresent returns false then return -4
            x.    Declare a bool variable check_sol with value obtained by calling function
                  CheckJsolved with formal parameter sudoku and solved
            xi.   If check_sol is true then  print “Solution is correct” and return 0
            xii.  If check_sol is false then print “Solution is incorrect” and return 1
        s. If user selects to exit return 0

    6. InputGrid (type int; formal parameter sudoku_grid)
        a. Print “Input the sudoku grid (0 for blank box)”
        b. Set up an iteration with index1 initialized to 0 and increases by 1 upto 8
        c. Set up an iteration with index2 initialized to 0 and increases by 1 upto 8
        d. Take input
```

    e. If any input does not lie between 0 to 9 then display error message and return -1
    f. Otherwise store it in the element in sudoku_grid corresponding to index1 and index2
    g. Start next iteration
    h. Print “Given sudoku is:”
    i. Call function DisplaySudoku with formal parameter sudoku_grid and return 0

7. InputBox (type int; formal parameter box_grid)
    a. Print “Input the jigsaw sudoku boxes”
    b. Set up an iteration with index1 initialized to 0 and increases by 1 upto 8
    c. Set up an iteration with index2 initialized to 0 and increases by 1 upto 8
    d. Take input
    e. If any input does not lie between 1 to 9 then display error message and return -1
    f. Otherwise store it in the element in box_grid corresponding to index1 and index2
    g. Start next iteration
    h. Print “Given sudoku boxes are:”
    i. Call function DisplaySudoku with formal parameter box_grid and return 0

8. ValidBox (type bool; no formal parameter; member of structure jig)
    a. Declare four bool variables bool1, bool2, bool3 and bool4
    b. Declare a 9 element int array box_tile with all elements zero
    c. Set up an iteration with index1 initialized to 0 and increases by 1 upto 8
    d. Set up an iteration with index2 initialized to 0 and increases by 1 upto 8
    e. Increment element in box_tile corresponding to value of element in box_grid corresponding to index1 and index2 by one
    f. bool1 is true if index1 is more than zero and element in box_grid corresponding to index1-1 and index2 is same as element corresponding to index1 and index2
    g. bool2 is true if index2 is more than zero and element in box_grid corresponding to index1 and index2-1 is same as element corresponding to index1 and index2
    h. bool3 is true if index1is than eight and element in box_grid corresponding to index1+1 and index2 is same as element corresponding to index1 and index2
    i. bool4 is true if index2 is less than eight and element in box_grid corresponding to index1 and index2+1 is same as element corresponding to index1 and index2
    j. If are all false then print “Invalid jigsaw sudoku boxes” and return false
    k. Start next iteration
    l. Check if all elements of box_tile are 9. If not return false.
    m. If no value has been returned then return true

9. InputPresent (type bool; no formal parameter; member of structure jig)
    a. Initialise the array present to all zeroes
    b. Declare int varibles grid_ num and box_num
    c. Set up an iteration with index1 initialized to 0 and increases by 1 upto 8
    d. Set up an iteration with index2 initialized to 0 and increases by 1 upto 8

```
        e. If element in sudoku.grid corresponding to index1 and index2 is zero start next
           iteration
        f. Else
              i. Define grid_num equals to element in sudoku.grid corresponding to index1 and
                 index2
             ii. Define box_num equals to element in sudoku.box corresponding to index1 and
                 index2
            iii. If element in sudoku.present corresponding to box_num and grid_num is false
                 then make it true
             iv. If it is already true print "Invalid jigsaw sudoku grid" and return false
        g. Start next iteration
        h. If no value has been returned then return true

    10. SolveNSudoku (type bool; formal parameter sudoku_grid and a)
        a. Declare two local int variables row and col
        b. Call bool function FindBlank with formal parameters sudoku_grid, row and col (passed
           by reference)
        c. If FindBlank returns false then return true as sudoku is solved
        d. If FindBlank returns true
              i. Declare an int variable num
             ii. Set up an iteration with num initialized to 5-4a and changes by a upto 5+4a
            iii. Call bool function CheckNumN with formal parameters sudoku_grid, row, col
                 and num
             iv. If CheckNumN is true then change the value of element in sudoku_grid
                 corresponding to row and col to num
              v. Call the function SolveNSudoku recursively with formal parameter
                 sudoku_grid
             vi. If SolveNSudoku is true then return true
            vii. If SolveNSudoku is false then change the value of element in sudoku_grid
                 corresponding to row and col back to 0
           viii. Start next iteration
        e. If after iteration for all 9 values of num sudoku has not been solved return false

    11. SolveDSudoku (type bool; formal parameter sudoku_grid and a)
        a. Declare two local int variables row and col
        b. Call bool function FindBlank with formal parameters sudoku_grid, row and col (passed
           by reference)
        c. If FindBlank returns false then return true as sudoku is solved
        d. If FindBlank returns true
              i. Declare an int variable num
             ii. Set up an iteration with num initialized to 5-4a and changes by a upto 5+4a
            iii. Call bool function CheckNumD with formal parameters sudoku_grid, row, col
                 and num
```

    iv. If CheckNumD is true then change the value of element in sudoku_grid corresponding to row and col to num
    v. Call the function SolveDSudoku recursively with formal parameter sudoku_grid
    vi. If SolveDSudoku is true then return true
    vii. If SolveDSudoku is false then change the value of element in sudoku_grid corresponding to row and col back to 0
    viii. Start next iteration
  e. If after iteration for all 9 values of num sudoku has not been solved return false

12. SolveWSudoku (type bool; formal parameter sudoku_grid and a)
    a. Declare two local int variables row and col
    b. Call bool function FindBlank with formal parameters sudoku_grid, row and col (passed by reference)
    c. If FindBlank returns false then return true as sudoku is solved
    d. If FindBlank returns true
        i. Declare an int variable num
        ii. Set up an iteration with num initialized to 5-4a and changes by a upto 5+4a
        iii. Call bool function CheckNumW with formal parameters sudoku_grid, row, col and num
        iv. If CheckNumW is true then change the value of element in sudoku_grid corresponding to row and col to num
        v. Call the function SolveWSudoku recursively with formal parameter sudoku_grid
        vi. If SolveWSudoku is true then return true
        vii. If SolveWSudoku is false then change the value of element in sudoku_grid corresponding to row and col back to 0
        viii. Start next iteration
    e. If after iteration for all 9 values of num sudoku has not been solved return false

13. SolveJSudoku (type bool; formal parameter sudoku and a)
    a. Declare two local int variables row and col
    b. Call bool function FindBlank with formal parameters sudoku.grid, row and col (passed by reference)
    c. If FindBlank returns false then return true as sudoku is solved
    d. If FindBlank returns true
        i. Declare an int variable num
        ii. Define int variable box_num as element in sudoku.box corresponding to row and col
        iii. Set up an iteration with num initialized to 5-4a and changes by a upto 5+4a
        iv. Call bool function CheckNumJ with formal parameters sudoku, row, col and num

```
        v.    If CheckNumJ returns true and element in sudoku.present corresponding to
              box_num and num is false then change the value of element in sudoku.grid
              corresponding to row and col to num
       vi.    Also change the value of element in sudoku.present corresponding to respective
              box and num to true
      vii.    Call the function SolveJSudoku recursively with formal parameter sudoku
     viii.    If SolveJSudoku is true then return true
       ix.    If SolveJSudoku is false then change the value of element in sudoku.grid
              corresponding to row and col back to 0
        x.    Also change the value of element in sudoku.present corresponding to respective
              box and num back to false
       xi.    Start next iteration
    e.  If after iteration for all 9 values of num sudoku has not been solved return false

14. FindBlank (type bool; formal parameters sudoku_grid, row and col - passed by reference)
    a.  Set up an iteration with row initialized to 0 and increases by 1 upto 8
    b.  Set up an iteration with col initialized to 0 and increases by 1 upto 8
    c.  If elements in sudoku_grid corresponding to row and col has value 0 then return true
    d.  Start next iteration
    e.  If no value has been returned so far then no element in sudoku_grid is 0 and so return
        false

15. CheckNumN (type bool; formal parameters sudoku_grid, row, col and num)
    a.  Call bool function CheckRow with formal parameters sudoku_grid, row, col   and num
    b.  Call bool function CheckCol with formal parameters sudoku_grid, row, col and num
    c.  Declare local int variables rowstart equal to 3*(row/3) and colstart equal to  3*(col/3)
    d.  Call bool function CheckBox with formal parameters sudoku_grid, row, col, rowstart,
        colstart and num
    e.  If all functions return false then return true else return false

16. CheckNumD (type bool; formal parameters sudoku_grid, row, col and num)
    a.  If tile lies on diagonal1 i.e. row equals col
        i.    Call bool function CheckRow with formal parameters sudoku_grid, row, col,
              and num
       ii.    Call bool function CheckCol with formal parameters sudoku_grid, row, col,
              and num
      iii.    Declare local int variables rowstart equal to 3*(row/3) and colstart equal to
              3*(col/3)
       iv.    Call bool function CheckBox with formal parameters sudoku_grid, row, col,
              rowstart, colstart and num
        v.    Call bool function CheckDia1 with formal parameters sudoku_grid, row, col
              and num
       vi.    If all functions return false then return true else return false
    b. If tile lies on diagonal2 i.e. row + col equals 8
        i. Call bool function CheckRow with formal parameters sudoku_grid, row, col,
           and num
        ii. Call bool function CheckCol with formal parameters sudoku_grid, row, col,
            and num
        iii. Declare local int variables rowstart equal to 3*(row/3) and colstart equal to
             3*(col/3)
        iv. Call bool function CheckBox with formal parameters sudoku_grid, row, col,
            rowstart, colstart and num
        v. Call bool function CheckDia2 with formal parameters sudoku_grid, row, col
           and num
        vi. If all functions return false then return true else return false
    c. Else
        i. Call bool function CheckRow with formal parameters sudoku_grid, row, col,
           and num
        ii. Call bool function CheckCol with formal parameters sudoku_grid, row, col,
            and num
        iii. Declare local int variables rowstart equal to 3*(row/3) and colstart equal to
             3*(col/3)
        iv. Call bool function CheckBox with formal parameters sudoku_grid, row, col,
            rowstart, colstart and num
        v. If all functions return false then return true else return false

17. CheckNumW (type bool; formal parameters sudoku_grid, row, col and num)
    a. If tile lies on window1 i.e. row equals 1, 2 or 3 and col equals 1, 2 or 3
        i. Call bool function CheckRow with formal parameters sudoku_grid, row, col,
           and num
        ii. Call bool function CheckCol with formal parameters sudoku_grid, row, col,
            and num
        iii. Declare local int variables rowstart equal to 3*(row/3) and colstart equal to
             3*(col/3)
        iv. Call bool function CheckBox with formal parameters sudoku_grid, row, col,
            rowstart, colstart and num
        v. Redefine rowstart equal to 1 and colstart equal to 1
        vi. Call bool function CheckBox with formal parameters sudoku_grid, row, col,
            rowstart, colstart and num
        vii. If all functions return false then return true else return false
    b. If tile lies on window2 i.e. row equals 1, 2 or 3 and col equals 5, 6 or 7
        i. Call bool function CheckRow with formal parameters sudoku_grid, row, col,
           and num
        ii. Call bool function CheckCol with formal parameters sudoku_grid, row, col,
            and num
        iii. Declare local int variables rowstart equal to 3*(row/3) and colstart equal to 3*(col/3)
        iv. Call bool function CheckBox with formal parameters sudoku_grid, row, col, rowstart, colstart and num
        v. Redefine rowstart equal to 1 and colstart equal to 5
        vi. Call bool function CheckBox with formal parameters sudoku_grid, row, col, rowstart, colstart and num
        vii. If all functions return false then return true else return false
    c. If tile lies on window3 i.e. row equals 5, 6 or 7 and col equals 1, 2 or 3
        i. Call bool function CheckRow with formal parameters sudoku_grid, row, col, and num
        ii. Call bool function CheckCol with formal parameters sudoku_grid, row, col, and num
        iii. Declare local int variables rowstart equal to 3*(row/3) and colstart equal to 3*(col/3)
        iv. Call bool function CheckBox with formal parameters sudoku_grid, row, col, rowstart, colstart and num
        v. Redefine rowstart equal to 5 and colstart equal to 1
        vi. Call bool function CheckBox with formal parameters sudoku_grid, row, col, rowstart, colstart and num
        vii. If all functions return false then return true else return false
    d. If tile lies on window4 i.e. row equals 5, 6 or 7 and col equals 5, 6 or 7
        i. Call bool function CheckRow with formal parameters sudoku_grid, row, col, and num
        ii. Call bool function CheckCol with formal parameters sudoku_grid, row, col, and num
        iii. Declare local int variables rowstart equal to 3*(row/3) and colstart equal to 3*(col/3)
        iv. Call bool function CheckBox with formal parameters sudoku_grid, row, col, rowstart, colstart and num
        v. Redefine rowstart equal to 5 and colstart equal to 5
        vi. Call bool function CheckBox with formal parameters sudoku_grid, row, col, rowstart, colstart and num
        vii. If all functions return false then return true else return false
    e. Else
        i. Call bool function CheckRow with formal parameters sudoku_grid, row, col, and num
        ii. Call bool function CheckCol with formal parameters sudoku_grid, row, col, and num
        iii. Declare local int variables rowstart equal to 3*(row/3) and colstart equal to 3*(col/3)
        iv. Call bool function CheckBox with formal parameters sudoku_grid, row, col, rowstart, colstart and num
```

    v. If all functions return false then return true else return false

18. CheckNumJ (type bool; formal parameters sudoku, row, col and num)
    a. Call bool function CheckRow with formal parameters sudoku.grid, row, col, and num
    b. Call bool function CheckCol with formal parameters sudoku.grid, row, col, and num
    c. If all functions return false then return true else return false

19. CheckRow (type bool; formal parameters sudoku_grid, row, col and num)
    a. Declare local int variable index
    b. Set up an iteration with index initialized to 0 and increases by 1 upto 8
    c. If element in sudoku_grid corresponding to row and index has value num and it does not correspond to row and col then return true
    d. Else start next iteration
    e. If no value has been returned so far then return false

20. CheckCol (type bool; formal parameters sudoku_grid, row, col and num)
    a. Declare local int variable index
    b. Set up an iteration with index initialized to 0 and increases by 1 upto 8
    c. If element in sudoku_grid corresponding to index and col has value num and it does not correspond to row and col then return true
    d. Else start next iteration
    e. If no value has been returned so far then return false

21. CheckBox (type bool; formal parameters sudoku_grid, row, col, rowstart, colstart and num)
    a. Declare local int variables index1 and index2
    b. Set up an iteration with index1 initialized to rowstart and increases by 1 upto rowstart + 2
    c. Set up an iteration with index2 initialized to colstart and increases by 1 upto colstart + 2
    d. If element in sudoku_grid corresponding to index1 and index2 has value num and it does not correspond to row and col then return true
    e. Else start next iteration
    f. If no value has been returned so far then return false

22. CheckDia1 (type bool; formal parameters sudoku_grid, row, col, and num)
    a. Declare local int variable index
    b. Set up an iteration with index initialized to 0 and increases by 1 upto 8
    c. If element in sudoku_grid corresponding to index and index has value num and it does not correspond to row and col then return true
    d. Else start next iteration
    e. If no value has been returned so far then return false

23. CheckDia2 (type bool; formal parameters sudoku_grid, row, col, and num)
    a. Declare local int variable index

b. Set up an iteration with index initialized to 0 and increases by 1 upto 8
c. If element in sudoku_grid corresponding to index and 8 - index has value num and it does not correspond to row and col then return true
d. Else start next iteration
e. If no value has been returned so far then return false

24. DisplaySudoku (type void; formal parameter sudoku_grid)
    a. Declare local int variables index1 and index2
    b. Set up an iteration with index1 initialized to 0 and increases by 1 upto 8
    c. Set up an iteration with index2 initialized to 0 and increases by 1 upto 8
    d. Print the element in sudoku_grid corresponding to index1 and index2
    e. Print “ “ (space character)
    f. Start next iteration for index2
    g. Start new line
    h. Start next iteration for index1

25. CheckNSudoku (type int; formal parameters sudoku_grid and solved_grid)
    a. Validate solution by calling ValidN with formal parameters solved_grid
    b. If ValidN returns false return 0
    c. Set up an iteration with row initialized to 0 and increases by 1 upto 8
    d. Set up an iteration with col initialized to 0 and increases by 1 upto 8
    e. If elements in sudoku_grid corresponding to row and col has value 0 then return -1 start next loop
    f. Else check if elements corresponding to row and col in solved_grid and sudoku_grid are same
    g. If they are different return 0
    h. Start next iteration
    i. If no value has been returned so far then return 1

26. CheckDSudoku (type int; formal parameters sudoku_grid and solved_grid)
    a. Validate solution by calling ValidD with formal parameters solved_grid
    b. If ValidD returns false return 0
    c. Set up an iteration with row initialized to 0 and increases by 1 upto 8
    d. Set up an iteration with col initialized to 0 and increases by 1 upto 8
    e. If elements in sudoku_grid corresponding to row and col has value 0 then return -1 start next loop
    f. Else check if elements corresponding to row and col in solved_grid and sudoku_grid are same
    g. If they are different return 0
    h. Start next iteration
    i. If no value has been returned so far then return 1

27. CheckWSudoku (type int; formal parameters sudoku_grid and solved_grid)
    a. Validate solution by calling ValidW with formal parameters solved_grid
    b. If ValidW returns false return 0
    c. Set up an iteration with row initialized to 0 and increases by 1 upto 8
    d. Set up an iteration with col initialized to 0 and increases by 1 upto 8
    e. If elements in sudoku_grid corresponding to row and col has value 0 then return -1 start next loop
    f. Else check if elements corresponding to row and col in solved_grid and sudoku_grid are same
    g. If they are different return 0
    h. Start next iteration
    i. If no value has been returned so far then return 1

28. CheckJSudoku (type int; formal parameters sudoku and solved)
    a. Validate solution by calling ValidJ with formal parameters solved
    b. If ValidJ returns false return 0
    c. Declare two local int variables row and col
    d. Set up an iteration with row initialized to 0 and increases by 1 upto 8
    e. Set up an iteration with col initialized to 0 and increases by 1 upto 8
    f. If elements in sudoku.grid corresponding to row and col has value 0 then return -1 start next loop
    g. Else check if elements corresponding to row and col in solved.grid and sudoku.grid are same
    h. If they are different return 0
    i. Start next iteration
    j. If no value has been returned so far then return 1

29. ValidN (type bool; formal parameters sudoku_grid)
    a. Set up an iteration with i initialized to 0 and increases by 1 upto 8
    b. Set up an iteration with j initialized to 0 and increases by 1 upto 8
    c. Call bool function CheckNumN with formal parameters sudoku_grid, i, j and element in sudoku_grid corresponding to i and j
    d. If CheckNumN returns false return false
    e. Start next iteration
    f. If no value has been returned so far then return true

30. ValidD (type bool; formal parameters sudoku_grid)
    a. Set up an iteration with i initialized to 0 and increases by 1 upto 8
    b. Set up an iteration with j initialized to 0 and increases by 1 upto 8
    c. Call bool function CheckNumD with formal parameters sudoku_grid, i, j and element in sudoku_grid corresponding to i and j
    d. If CheckNumN returns false return false
    e. Start next iteration

    f. If no value has been returned so far then return true

31. ValidW (type bool; formal parameters sudoku_grid)
    a. Set up an iteration with i initialized to 0 and increases by 1 upto 8
    b. Set up an iteration with j initialized to 0 and increases by 1 upto 8
    c. Call bool function CheckNumW with formal parameters sudoku_grid, i, j and element in sudoku_grid corresponding to i and j
    d. If CheckNumN returns false return false
    e. Start next iteration
    f. If no value has been returned so far then return true

32. ValidJ (type bool; formal parameters sudoku)
    a. Set up an iteration with i initialized to 0 and increases by 1 upto 8
    b. Set up an iteration with j initialized to 0 and increases by 1 upto 8
    c. Call bool function CheckNumJ with formal parameters sudoku.grid, i, j and element in sudoku.grid corresponding to i and j
    d. If CheckNumJ returns false return false
    e. Start next iteration
    f. Call InputPresent as a member of solved
    g. If InputPresent returns false return false
    h. If no value has been returned so far then return true

33. NSudoku (type int; formal parameters sudoku_grid)
    a. Validate sudoku_grid using ValidN and if invalid return -1
    b. Call function SolveNSudoku with parameters sudoku grid and 1/-1
    c. If SolveNSudoku is false return -1
    d. Otherwise if the two solution grids match return 1
    e. If two solution grids do not match return 0

34. DSudoku (type int; formal parameters sudoku_grid)
    a. Validate sudoku_grid using ValidD and if invalid return -1
    b. Call function SolveDSudoku with parameters sudoku grid and 1/-1
    c. If SolveDSudoku is false return -1
    d. Otherwise if the two solution grids match return 1
    e. If two solution grids do not match return 0

35. WSudoku (type int; formal parameters sudoku_grid)
    a. Validate sudoku_grid using ValidW and if invalid return -1
    b. Call function SolveWSudoku with parameters sudoku grid and 1/-1
    c. If SolveWSudoku is false return -1
    d. Otherwise if the two solution grids match return 1
    e. If two solution grids do not match return 0

36. JSudoku (type int; formal parameters sudoku_grid)
    a. Validate sudoku_grid using ValidJ and if invalid return -1
    b. Call function SolveJSudoku with parameters sudoku grid and 1/-1
    c. If SolveJSudoku is false return -1
    d. Otherwise if the two solution grids match return 1
    e. If two solution grids do not match return 0

Evaluations

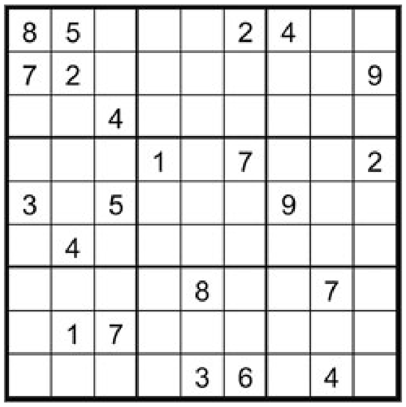
So far we have been able to code the normal sudoku solver and it is able to solve all valid sudoku puzzles given to it. Note that valid here refers to puzzles having unique solutions.

We feeded several easy puzzles and each of them was solved in less than 0.04 seconds.
The following is an easy puzzle and that our program solved.

```
0 0 3 | 0 2 0 | 6 0 0
9 0 0 | 3 0 5 | 0 0 1
0 0 1 | 8 0 6 | 4 0 0
0 0 8 | 1 0 2 | 9 0 0
7 0 0 | 0 0 0 | 0 0 8
0 0 6 | 7 0 8 | 2 0 0
0 0 2 | 6 0 9 | 5 0 0
8 0 0 | 2 0 3 | 0 0 9
0 0 5 | 0 1 0 | 3 0 0
```

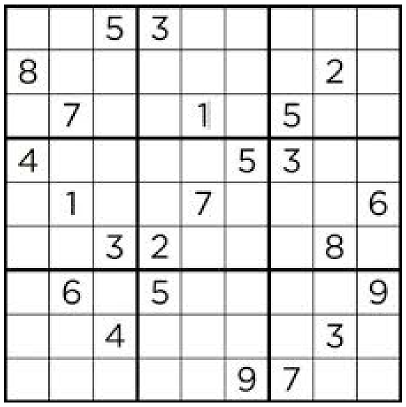
```
4 8 3 | 9 2 1 | 6 5 7
9 6 7 | 3 4 5 | 8 2 1
2 5 1 | 8 7 6 | 4 9 3
5 4 8 | 1 3 2 | 9 7 6
7 2 9 | 5 6 4 | 1 3 8
1 3 6 | 7 9 8 | 2 4 5
3 7 2 | 6 8 9 | 5 1 4
8 1 4 | 2 5 3 | 7 6 9
6 9 5 | 4 1 7 | 3 8 2
```

Finnish mathematician Arto Inkala described his 2006 puzzle as "the most difficult sudoku-puzzle known so far" and his 2010 puzzle as "the most difficult puzzle I've ever created." Our program solves them in under 0.35 seconds each.



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

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



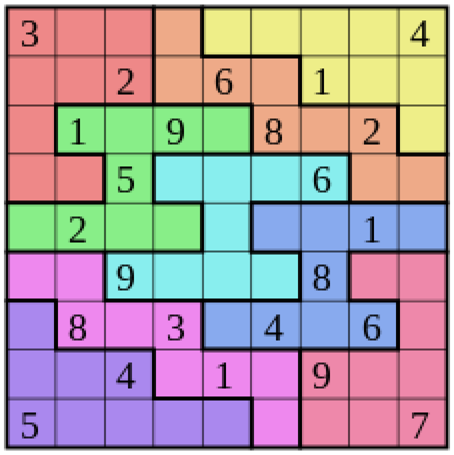
		5	3					
8							2	
	7			1		5		
4					5	3		
	1			7				6
		3	2				8	
	6		5					9
		4					3	
					9	7		

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

In case of sudoku puzzles that possess multiple solutions our program gives any one possible solution and tells whether unique solution is possible or not.
On giving blank grid (all zeroes) a solution is given in less than 0.03 seconds.

In case of diagonal sudoku and window sudoku also it takes almost the same amount of time.However for jigsaw sudoku it requires more time owing to larger number of steps to be performed.

The following is a jigsaw sudoku puzzle and was solved in 0.08 seconds



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

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

The algorithm that we have used is based on the idea of brute force and hence it involves more number of computations as compared to other more advanced algorithms. It is possible that if an extremely tough version of a variant is given then it may lead to segmentation faults or other errors.

Conclusions and Future Scope

- The project designed is capable of solving normal sudoku and its variants like diagonal sudoku, window sudoku and jigsaw sudoku of any degree of toughness.
- This model can also be extended to solve larger grids and more complicated variants.
- We have incorporated a graphical user interface so that the program becomes more user-friendly.

References

- CS101
    - http://www.cse.iitb.ac.in/~cs101/project.html
    - http://www.cse.iitb.ac.in/~cs101/Project/Manual_Code::Blocks_Simplecpp.pdf
    - Past year projects
- wikipedia
    - http://en.wikipedia.org/wiki/Sudoku
    - http://en.wikipedia.org/wiki/Sudoku_solving_algorithms
    - http://en.wikipedia.org/wiki/Backtracking
- http://www.geeksforgeeks.org/backtracking-set-7-suduku/
- norvig.com/sudoku.html
- http://rohanrao.blogspot.in/2010_04_01_archive.html
- http://usatoday30.usatoday.com/news/offbeat/2006-11-06-sudoku_x.htm
- http://www.mirror.co.uk/news/weird-news/worlds-hardest-sudoku-can-you-242294
- http://www.tifr.res.in/~cccf/index.php/interns/77-general/127-how-to-write-a-structured-project-report