Sudoku

1. Problem

Write a program to solve any given *easy to medium* difficulty *level* Sudoku puzzle.

2. Play the game

If you have not solved a Sudoku puzzle you can go to sudoku9x9.com to solve one. You'll need to know how to solve any given puzzle before you can teach the computer how to solve it. sudoku9x9.com can also show you step by step how to solve it. You can also enter your own puzzle.

3. Here's an easy puzzle

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

Enter this puzzle into sudoku9x9.com and try to solve it, first on your own, and then with the solver for step by step instructions if you can't solve it.

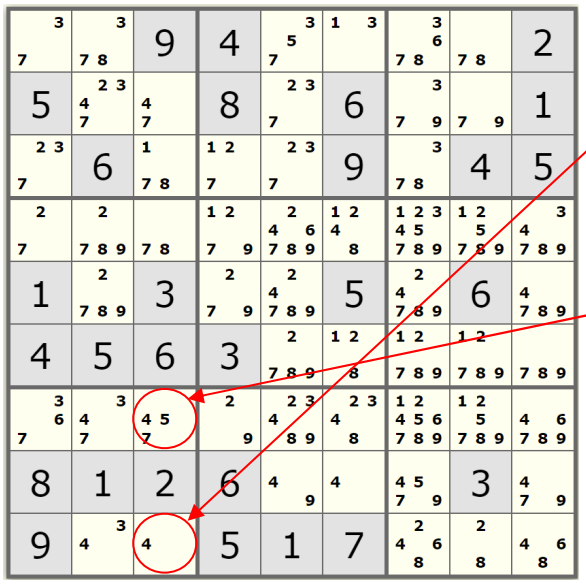
Note that your program doesn't have to follow the given step by step instructions. This is just to give you an idea of how to solve the puzzle.

Step 1

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

Find all possible solutions for all the empty cells, i.e., cells with no solutions yet.

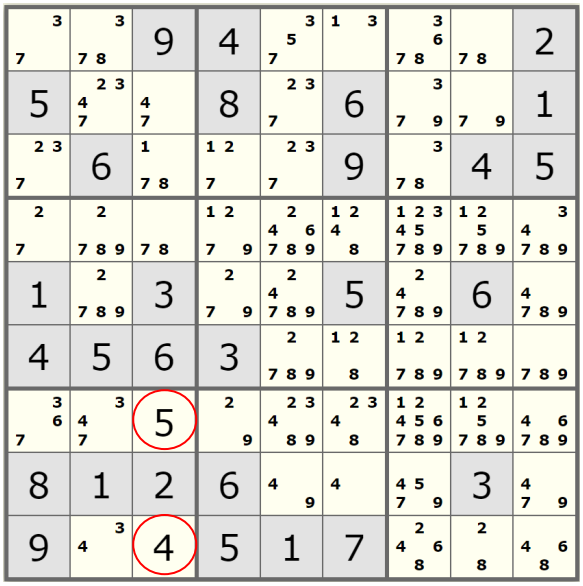
Step 2



Two different things to look for in all the possible solutions:

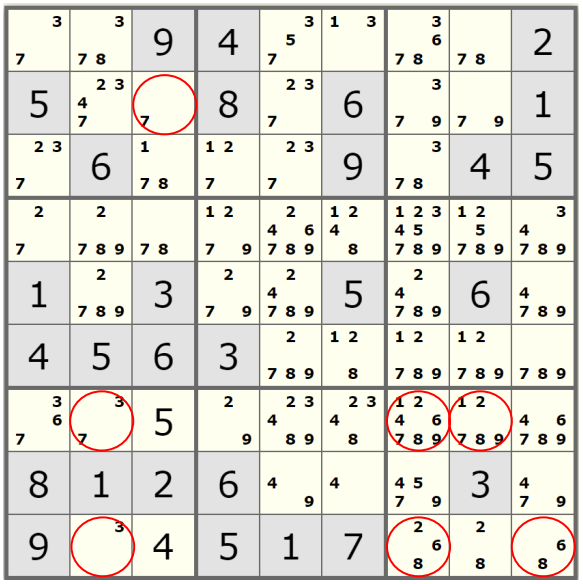
1) If there's only one possible solution for a cell then that number is a solution for that cell. Ex. Cell row 8, column 2 contains only a 4, so 4 is a solution for that cell. – **Naked Single**

2) If a possible solution is the only number in the row, column, or quadrant, then that number is a solution for that cell. Ex. Cell row 6, column 2 contains the number 5 which is the only occurrence for that quadrant, so 5 is a solution for that cell. – **Hidden Single**



After filling in the numbers 5 and 4 for the solutions for the two cells

Step 3



Repeat from step 1 until no more changes. All the solutions should have been found (for easy puzzles).

This is the result from doing step 1 the second time around.

Notice the changes made for the possible solutions in the circled cells. There are now two cells with only one possible solution.

Doing the above 3 steps should be sufficient to solve all the easy to medium difficulty level puzzles.

Here's the final solution for this puzzle.

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

4. Grading

C – If your program can only print out possible solutions but cannot solve a puzzle completely.

B – If your program can solve these two puzzles. They can be solved by searching for **Naked Singles** only.

Puzzle 1

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

Using only Naked Singles will get 81 solutions

Puzzle 2

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

Using only Naked Singles will get 81 solutions

A – If your program can solve these two puzzles. Need to search for both **Naked Singles** and **Hidden Singles**.

Puzzle 3

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

Using only Naked Singles will get 41 solutions

Puzzle 4

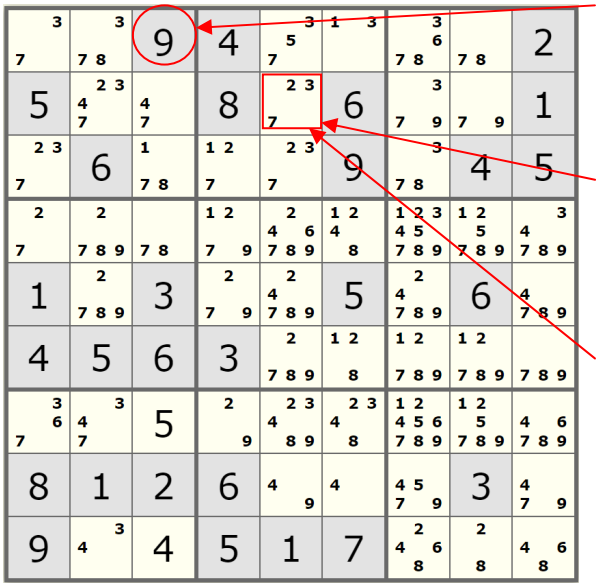
	3		6		9	5		
		1						
2	9						3	
8						4		
5			2	1			7	
	1				5	6		
6	2	9	5		8			
			7	2	6			

Using only Naked Singles will get 25 solutions

5. *Initial Template Code*

What follows are the data structures for storing the puzzle data and the initial template for your program. In addition to the member functions listed, there'll probably be many more that you'll need to have.

It is very important that you understand how to access the data that represent the solutions and the possible solutions for each cell. Here are some examples:



How do you assign the solution 9 to the red-circle cell?

board[0][2].solution = 9;

How do you assign the possible solution 2 to the red-square cell?

board[1][4].possible_solutions[2] = true;

How do you test if the number 3 is a possible solution in the red-square cell?

```
If (board[1][4].possible_solutions[3] == true) {
   // 3 is a possible solution
```

Sudoku.h

```
// Solves a Sudoku puzzle
// Copyright 2024 Enoch Hwang

#ifndef __SUDOKU__
#define __SUDOKU__

#include <iostream>
#include <string>
using namespace std;

class Sudoku {
private:
   // solution contains the solution number for the cell or 0 if no solution yet.
   // possible_solutions index 1 to 9 contains true if that index number is a
   // possible solution or false if that index number is not a possible solution.
   // possible_solutions[0] is not used.
   struct Board_Cell {
      int solution;
      bool possible_solutions[10];
   };

   // the main 9x9 Sudoku board
   Board_Cell board[9][9];

   // row,column coordinate of a cell on the board
   struct Coordinate {
      int row, column;
   };

   // For storing the coordinates of the cells in the same row,
   // same column and same quadrant as the given cell.
   // The coordinate of the given cell is not stored
   // so the array size is 8 and not 9
   Coordinate CellsInRow[8];
   Coordinate CellsInColumn[8];
   Coordinate CellsInQuadrant[8];

   bool NoSolution(Coordinate cell);
   void GetRowCells(Coordinate cell);
   void GetColumnCells(Coordinate cell);
   void GetQuadrantCells(Coordinate cell);

   bool SearchInSolution(int number);
   void FindPossibleSolutions(Coordinate cell);
   void FindAllPossibleSolutions();
   void FindAllNakedSingles(Coordinate cell);

   bool SearchInPossibleSolution(int number);
   void FindHiddenSingles(Coordinate cell);
   void FindAllHiddenSingles();

public:
   Sudoku();
   void CreatePuzzle();
   void PrintBoard();
   void PrintSolution(Coordinate coord);
   void testing();
   void SolvePuzzle();

};

#endif
```

Sudoku.cpp

```
#include "Sudoku.h"

// Constructor
// initialize all to 0
Sudoku::Sudoku() {
    for (int r = 0; r < 9; r++) {
        for (int c = 0; c < 9; c++) {
            board[r][c]. solution = 0;  // no solution yet
            for (int i = 0; i < 10; i++) {
                board[r][c].possible_solutions[i] = false; // no possible solution
            }
        }
    }
}


// initialize board with a puzzle
void Sudoku::CreatePuzzle() {
    board[0][2].solution = 9;
    board[0][3].solutions = 4;
   // the rest of the lines for the puzzle   ...
}

// prints the board with the solutions
void Sudoku::PrintBoard() {
    ...
}


// returns true if the given cell has no solution yet, i.e., it is empty,
// false otherwise
bool Sudoku::NoSolution(Coordinate cell) {
    ...
}

// print the solution for the given cell if there is one
// otherwise print all the possible solutions for the cell
void Sudoku::PrintSolution(Coordinate cell) {
    ...
}

// Initializes CellsInRow by filling in the coordinates
// for all the cells in the same row as the given cell.
// The coordinate of the given cell is not stored.
void Sudoku::GetRowCells(Coordinate cell) {
    ...
}

// Initializes CellsInColumn by filling in the coordinates
// for all the cells in the same column as the given cell.
// The coordinate of the given cell is not stored.
void Sudoku::GetColumnCells(Coordinate cell) {
    ...
}

// Initializes CellsInQuadrant by filling in the coordinates
// for all the cells in the same quadrant as the given cell.
// The coordinate of the given cell is not stored.
void Sudoku::GetQuadrantCells(Coordinate cell) {
    ...
}
```

```
// This is a helper function for FindPossibleSolutions
// For all the cells in the CellsInRow, CellsInColumn and CellsInQuadrant arrays
// that already have a solution
// search the solution in those cells for the given number
// (i.e. see if the solution in those cells is equal to the given number)
// returns true if the given number is found, false otherwise
bool Sudoku::SearchInSolution(int number) {
    // for all the cells in CellsInRow, CellsInColumn and CellsInQuadrant
    for (int i = 0; i < 8; i++) {
        if (...
}

// Find all the possible solutions for a given cell.
// The function updates the possible_solutions array for the given cell.
// The row,column coordinates specify the given cell.
void Sudoku::FindPossibleSolutions(Coordinate cell) {
    // get coordinates for all the cells in row, column and quadrant
    GetRowCells(coord);
    GetColumnCells(coord);
    GetQuadrantCells(coord);

    if (SearchInSolution(i)) { // if i is found in row, column or quadrant then
    ...                        // i is NOT a solution, otherwise it IS a solution
}

void Sudoku::testing() {
    CreatePuzzle();
    cout << "Initial puzzle" << endl;
    PrintBoard();
    Coordinate cell;
    cell.row = 6;
    cell.column = 2;
    FindPossibleSolutions(cell);
    PrintSolution(cell); // should print out 4,5,7

    cell.row = 8;
    cell.column = 2;
    FindPossibleSolutions(cell);
    PrintSolution(cell); // should print out 4

    cell.row = 6;
    cell.column = 7;
    FindPossibleSolutions(cell);
    PrintSolution(cell); // should print out 1,2,5,7,8,9
}
```

main.cpp

```
#include <iostream>
using namespace std;
#include "Sudoku.h""

int main() {
   Sudoku s;
   s.CreatePuzzle();
   s.PrintBoard();
   s.SolvePuzzle();
}
```

6. *GetQuadrantCells*

Given a cell coordinate, how to find the cells in the quadrant to initialize the array CellsInQuadrant?

0,0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8
1,0	1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8
2,0	2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8
3,0	3,1	3,2	3,3	3,4	3,5	3,6	3,7	3,8
4,0	4,1	4,2	4,3	4,4	4,5	4,6	4,7	4,8
5,0	5,1	5,2	5,3	5,4	5,5	5,6	5,7	5,8
6,0	6,1	6,2	6,3	6,4	6,5	6,6	6,7	6,8
7,0	7,1	7,2	7,3	7,4	7,5	7,6	7,7	7,8
8,0	8,1	8,2	8,3	8,4	8,5	8,6	8,7	8,8

Notice that for both the row and column coordinates in each quadrant they are in the range 0-2, 3-5, or 6-8. Here are the steps:

1. Given either the row or column number, determine which range (0-2, 3-5, or 6-8) it is in.
2. Once you know which range it is in then you can determine the starting number for that range.
3. Knowing the starting number for the range, you can now get the remaining two numbers.

   For example, given a 7, it is in the range 6-8, so the starting number is 6. Therefore the three numbers are 6, 7 and 8.

   A very efficient and simple way to find the starting number for the range is just to do an integer division and multiplication, i.e. number/3*3.

For your project you DO NOT need to read any further.

However, you'll get BONUS POINTS if you attempt what is discussed next.

7. Here's a very difficult puzzle to solve

Note, for your final project your program does not need to be able to solve this puzzle.

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

Start with the three steps for solving the easy puzzle.

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

After doing the three steps to solve the easy puzzle you'll end up with this, and the loop exits with no more solutions to be found.

Step 4

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

Two more new things to look for in all the possible solutions:

1) Look for two cells with the only two numbers in either row, column, or quadrant. Cells (8,7) and (8,8) are the only cells in the quadrant that have the numbers 1 and 5. So 1 and 5 are the only possible solutions in these two cells. – **Naked Pair**

2) Look for three cells with the only three numbers in either row, column, or quadrant. Cells (8,0), (8,1), and (8,3) have only the three numbers 2, 4 and 6 as possible solutions. So these three numbers are the only possible solutions in these three cells and cannot be possibilities for any other cells in that row. – **Naked Triplet**

Step 5

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

Update the possible solutions in the corresponding row, column or quadrant.

Step 6

Repeat from step 1, i.e., the steps for the easy solution.

Here's the final solution

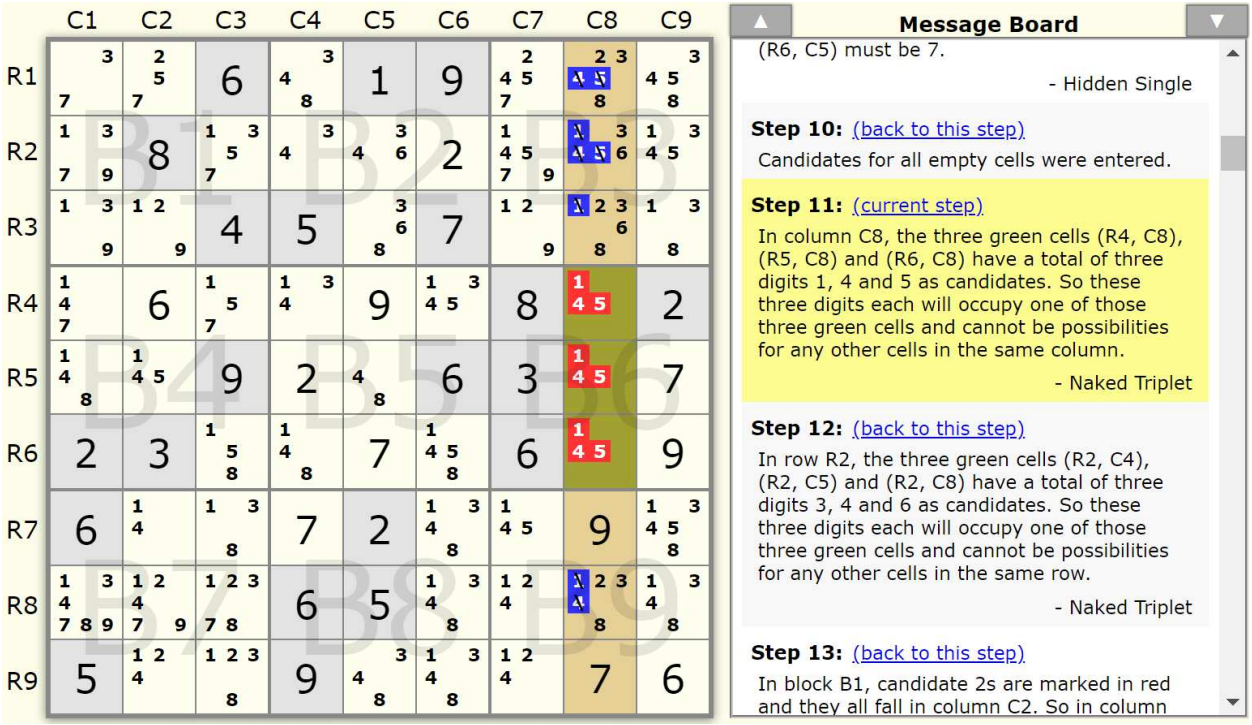
8	5	4	3	1	2	9	7	6
1	7	2	5	6	9	4	3	8
6	3	9	7	8	4	5	1	2
7	4	8	1	3	6	2	9	5
5	2	6	9	7	8	1	4	3
9	1	3	4	2	5	6	8	7
2	9	1	8	5	7	3	6	4
3	8	5	6	4	1	7	2	9
4	6	7	2	9	3	8	5	1

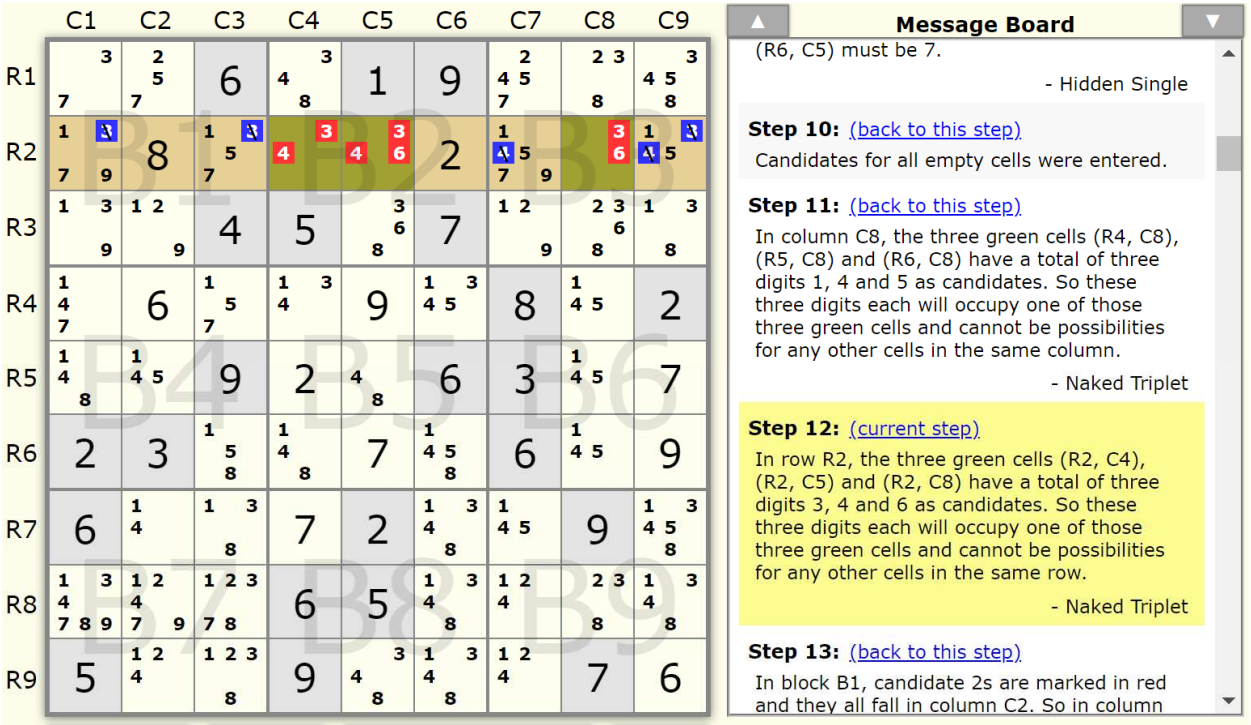
8. Here's an even harder puzzle to solve

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

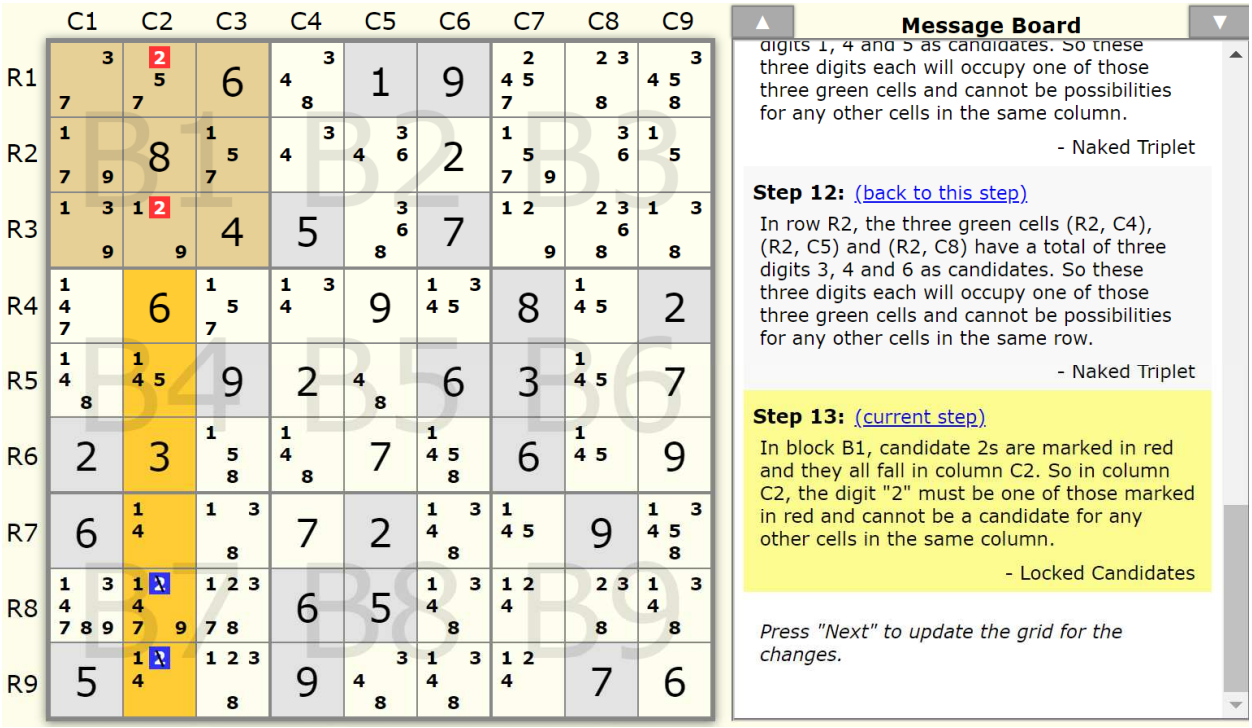
3	7	6	8	1	9	4	2	5
9	8	5	3	4	2	7	6	1
1	2	4	5	6	7	9	8	3
7	6	1	4	9	3	8	5	2
4	5	9	2	8	6	3	1	7
2	3	8	1	7	5	6	4	9
6	1	3	7	2	4	5	9	8
8	9	7	6	5	1	2	3	4
5	4	2	9	3	8	1	7	6

Naked Triplet

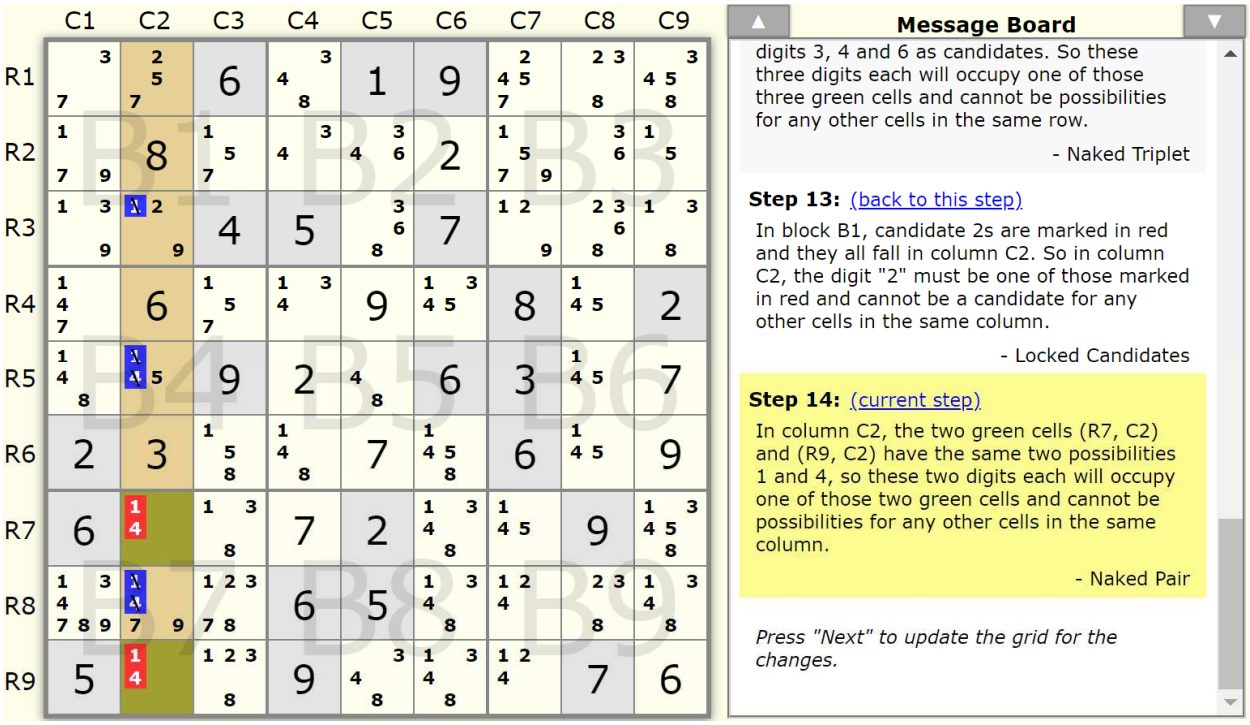




Locked candidates



Naked Pair



XY-Wing