Sudoku Solver

Spandana Patro

1702050103

Computer Science and Engineering

Veer Surendra Sai University of Technology, Burla

2020-2021

Sudoku Solver

A minor project submitted in partial fulfillment of the requirements for the

Degree of BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

By

SPANDANA PATRO

Registration No.: 1702050103

Under the Guidance of

Dr. Manas Ranjan Kabat

ASSOCIATE PROFESSOR & HEAD

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VEER SURENDRA SAI UNIVERSITY OF TECHNOLOGY

BURLA

VEER SURENDRA SAI UNIVERSITY OF TECHNOLOGY, BURLA, ODISHA

Declaration

I declare that this written submission represents my ideas in my own words and where other ideas or words have been included. I have adequately cited and referenced the original sources. I also declare that I adhered to all principles of academic and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will cause disciplinary action by the university and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

DATE:

Spandana Patro

Reg. No. – 1702050103

VEER SURENDRA SAI UNIVERSITY OF TECHNOLOGY, BURLA, ODISHA

Certificate

This is to certify that the major project report on "Sudoku Solver" submitted by SPANDANA PATRO, Registration No.: 1702050103, 7th Semester, Bachelor of Technology is approved for the degree of Bachelor of Technology in Computer Science and Engineering, is a record of an original research carried out by him under my supervision and guidance.

Dr. Manas Ranjan Kabat
Head of the Department

Dr. Manas Ranjan Kabat
Supervisor

Approval Sheet

This minor project entitled "**Sudoku Solver**" by Spandana Patro is approved for the degree of Bachelor of Technology in "Computer Science and Engineering", department of Computer Science and Engineering.

DATE: Supervisor

PLACE: Burla

ABSTRACT

Sudoku, originally called Number Place is a logic-based, number-placement puzzle. In classic Sudoku, 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 (also called "boxes", "blocks", or "regions") contain all of the digits from 1 to 9. The puzzle setter provides a partially completed grid, which for a well-posed puzzle has a single solution. In the last decade, solving the Sudoku puzzle has become every one's passion. The simplicity of puzzle's structure and the low requirement of mathematical skills caused people to have enormous interest in accepting challenges to solve the puzzle. Therefore, developers have tried to find algorithms in order to generate the variety of puzzles for human players so that they could be even solved by computer programming.

In this project I used backtracking method to solve the Sudoku puzzle and used html, css and javascript language to build this game. It aims at solving the puzzle of different modes (easy, medium and hard) in less time complexity.

CONTENTS

LIST OF FIGURES

1. INTRODUCTION

Currently, Sudoku puzzles are becoming increasingly popular among the people all over the world. The game has become popular now in a large number of countries and many developers have tried to generate even more complicated and more interesting puzzles. Today, the game appears in almost every newspaper, in books and in many websites.

In this project I present a Sudoku Solver named game using recursion and backtracking method by using html, css and javascript. The Brute force algorithm is then analyzed to compare with this algorithm in order to evaluate the efficiency of the proposed algorithm. The brute force is a general algorithm than can be applied to any possible problem. Backtracking is a technique based on algorithm to solve problem. It uses recursive calling to find the solution by building a solution step by step increasing values with time. It removes the solutions that don't give rise to the solution of the problem based on the constraints given to solve the problem.

1.1 More about Sudoku

Completed Sudoku puzzles area type of Latin square, with an additional constraint on the contents of individual regions. The modern puzzle was invented by an American architect, Howard Garns, in 1979 and published by Dell Magazines under the name "Number Place".It became popular in Japan in 1986, after it was published by Nikoli and given the name Sudoku, meaning single number.(Brian Hayes, 2006) It became an international hit in 2005. Solving Sudoku has been a challenging problem in the last decade. The purpose has been to develop more effective algorithm in order to reduce the computing time and utilize lower memory space. This project develops a online game for solving Sudoku puzzle by using a method, called backtracking method. Our ambition is to implement this algorithm by using html, css, java script. There are currently different variants of Sudoku such as 4X4 grids, 9X9 grids and 16X16 grids. This work is focused on classic and regular Sudoku of 9X9 board, and then a comparison is performed between the brute force method and Brute force algorithm. Hopefully, by doing this work we might be able to answer the following questions: How does the backtracking algorithm differ from the Brute force algorithm? Which one of them is more effective?

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

Fig 1 An Example of a Sudoku puzzle

1.2 Purpose

The aim of the project is to investigate the brute force algorithm and the backtracking algorithm. Later these two methods are compared by analyzing their time complexity. It is expected here to find an efficient method to solve the Sudoku puzzles. In this online game we have tried to implement the backtracking algorithm that would solve the puzzle by using some simple strategies that can be employed to solve the majority of Sudoku.

2. BACKGROUND STUDY

This section starts with the details about two algorithms brute force and backtracking algorithm. Later, we discuss further about evaluated algorithms and finally a description of how the code is being carried out is presented.

2.1 Brute force Technique

Approach: The naive approach is to generate all possible configurations of numbers from 1 to 9 to fill the empty cells. Try every configuration one by one until the correct configuration is found, i.e. for every unassigned position fill the position with a number from 1 to 9. After filling all the unassigned position check if the matrix is safe or not. If safe print else recurs for other cases.

Algorithm:

1. Create a function that checks if the given matrix is valid Sudoku or not. Keep Hash map for the row, column and boxes. If any number has a frequency greater than 1 in the hash Map return false else return true;
2. Create a recursive function that takes a grid and the current row and column index.
3. Check some base cases. If the index is at the end of the matrix, i.e. i=N-1 and j=N then check if the grid is safe or not, if safe print the grid and return true else return false. The other base case is when the value of column is N, i.e j = N, then move to next row, i.e. i++ and j = 0.
4. if the current index is not assigned then fill the element from 1 to 9 and recur for all 9 cases with the index of next element, i.e. i, j+1. if the recursive call returns true then break the loop and return true.
5. if the current index is assigned then call the recursive function with index of next element, i.e. i, j+1

2.2 Backtracking Method

Approach:

Sudoku can be solved by one by one assigning numbers to empty cells. Before assigning a number, check whether it is safe to assign. Check that the same number is not present in the current row, current column and current 3X3 sub grid. After checking for safety, assign the number, and recursively check whether this assignment leads to a solution or not. If the assignment doesn't lead to a solution, then try the next number for the current empty cell. And if none of the number (1 to 9) leads to a solution, return false and print no solution exists.

Algorithm:

1. Create a function that checks after assigning the current index the grid becomes unsafe or not. Keep Hash map for a row, column and boxes. If any number has a frequency greater than 1 in the hash Map return false else return true; hash Map can be avoided by using loops.
2. Create a recursive function that takes a grid.
3. Check for any unassigned location. If present then assign a number from 1 to 9, check if assigning the number to current index makes the grid unsafe or not, if safe then recursively call the function for all safe cases from 0 to 9. if any recursive call returns true, end the loop and return true. If no recursive call returns true then return false.
4. If there is no unassigned location then return true.

3. CODE AND OUTPUT

3.1 HTML Code

```
<!DOCTYPE html>
<html lang="en">

<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <link rel="stylesheet" href="style.css">
   <script defer src="script.js"></script>
</head>

<body>
   <nav class="navbar navbar-light " style="font-size: 25px; font-family: sans-serif;
background-color: cyan;">
      <h2 style="font-family: cursive; animation-fill-mode: forwards;">Sudoku Solver
Project</h2>

   </nav>
   <br><br>
   <div id="container">

      <div class="lsb tsb" id="0">

      </div>

      <div class="tsb ldb" id="1">

      </div>

      <div class="tsb ldb" id="2">

      </div>

      <div class="tsb lsb" id="3">

      </div>
```

```
<div class="tsb ldb" id="4">

</div>

<div class="tsb ldb" id="5">

</div>
<div class="tsb lsb" id="6">

</div>

<div class="tsb ldb" id="7">

</div>
<div class="tsb rsb ldb" id="8">

</div>

<div class="lsb tdb" id="9">

</div>

<div class="ldb tdb" id="10">

</div>

<div class="ldb tdb" id="11">

</div>

<div class="lsb tdb" id="12">

</div>
<div class="ldb tdb" id="13">

</div>

<div class="ldb tdb" id="14">

</div>
<div class="lsb tdb" id="15">
```

```
</div>

<div class="ldb tdb" id="16">

</div>
<div class="rsb ldb tdb" id="17">

</div>

<div class="lsb bsb tdb" id="18">

</div>

<div class="bsb ldb tdb" id="19">

</div>

<div class="bsb ldb tdb" id="20">

</div>

<div class="bsb lsb tdb" id="21">

</div>
<div class="bsb ldb tdb" id="22">

</div>

<div class="bsb ldb tdb" id="23">

</div>
<div class="bsb lsb tdb" id="24">

</div>

<div class="bsb ldb tdb" id="25">

</div>
<div class="bsb rsb ldb tdb" id="26">
```



```
 </div>
```

```
<div class="lsb" id="27">

</div>

<div class="ldb" id="28">

</div>

<div class="ldb" id="29">

</div>

<div class="lsb" id="30">

</div>
<div class="ldb" id="31">

</div>

<div class="ldb" id="32">

</div>
<div class="lsb" id="33">

</div>

<div class="ldb" id="34">

</div>
<div class="rsb ldb" id="35">

</div>

<div class="lsb tdb" id="36">

</div>

<div class="ldb tdb" id="37">

</div>
```

```
<div class="ldb tdb" id="38">
</div>

<div class="lsb tdb" id="39">

</div>
<div class="ldb tdb" id="40">

</div>

<div class="ldb tdb" id="41">

</div>
<div class="lsb tdb" id="42">

</div>

<div class="ldb tdb" id="43">

</div>
<div class="rsb ldb tdb" id="44">

</div>

<div class="lsb bsb tdb" id="45">

</div>

<div class="bsb ldb tdb" id="46">

</div>

<div class="bsb ldb tdb" id="47">
</div>

<div class="bsb lsb tdb" id="48">

</div>
<div class="bsb ldb tdb" id="49">

</div>
```

```
<div class="bsb ldb tdb" id="50">

</div>
<div class="bsb lsb tdb" id="51">

</div>

<div class="bsb ldb tdb" id="52">

</div>
<div class="bsb rsb ldb tdb" id="53">

</div>

<div class="lsb" id="54">

</div>

<div class="ldb" id="55">

</div>

<div class="ldb" id="56">

</div>

<div class="lsb" id="57">

</div>
<div class="ldb" id="58">

</div class="ldb">

<div class="ldb" id="59">

</div>
<div class="lsb" id="60">
</div>

<div class="ldb" id="61">
```

```
</div>
<div class="rsb ldb" id="62">

</div>

<div class="lsb tdb" id="63">

</div>

<div class="ldb tdb" id="64">

</div>

<div class="ldb tdb" id="65">

</div>

<div class="lsb tdb" id="66">

</div>
<div class="ldb tdb" id="67">

</div>

<div class="ldb tdb" id="68">

</div>
<div class="lsb tdb" id="69">

</div>

<div class="ldb tdb" id="70">
</div>
<div class="rsb ldb tdb" id="71">

</div>

<div class="lsb bsb tdb" id="72">

</div>
```

```
      <div class="bsb ldb tdb" id="73">

      </div>

      <div class="bsb ldb tdb" id="74">

      </div>

      <div class="bsb lsb tdb" id="75">

      </div>
      <div class="bsb ldb tdb" id="76">

      </div>

      <div class="bsb tdb ldb" id="77">

      </div>
      <div class="bsb lsb tdb" id="78">

      </div>

      <div class="bsb ldb tdb" id="79">

      </div>
      <div class="bsb rsb ldb tdb" id="80">

      </div>


   </div>
   <br>
   <div id="buttons">
      <button type="button" class="btn btn-danger" id="generate-sudoku">Get New
         Puzzle</button>    
      <button type="button" class="btn btn-success" id="solve">Solve</button>
   </div>
</body>

</html>
```

3.2 CSS Code

```
#container{
   height: auto;
   width: 540px;
   background-color:#a6e4ef;
   display: flex;
   flex-wrap: wrap;
   justify-content: space-evenly;
   align-content: space-evenly ;
   margin: 0 auto;
}
body{
   background-color: #a6e4ef;
}
body h2{
   border-left: 20px;
   border-left-color: red;
}
#generate-sudoku{
   margin:auto;
   display:block;;
}
#solve{
   margin:auto;
   display:block;;
}
#container div{
   background-color: #72a2c5;
   height: 60px;
   width: 60px;
   box-sizing: border-box;
   font-family: cursive;
   text-align: center;
   vertical-align: middle;
   line-height: 60px;
   font-size: 30px;
```

```
   color: green;
   }
#container div:hover{
   background-color: lightskyblue;
}
.lsb{
   border-left: black 2px solid;
}

.bsb{
   border-bottom: black 2px solid;
}

.rsb{
   border-right: black 2px solid;
}
.tsb{
   border-top: black 2px solid;
}
.ldb{
   border-left: black 0.6px dashed;
}
.bdb{
   border-bottom: black 0.6px dashed;
}
.rdb{
   border-right: black 0.6px dashed;
}

.tdb{
   border-top: black 0.6px dashed;
}
```

3.3 JavaScript Code

```
var arr = [[], [], [], [], [], [], [], [], []]
var temp = [[], [], [], [], [], [], [], [], []]
for (var i = 0; i < 9; i++) {
    for (var j = 0; j < 9; j++) {
        arr[i][j] = document.getElementById(i * 9 + j);
    }
}
function initializeTemp(temp) {
    for (var i = 0; i < 9; i++) {
        for (var j = 0; j < 9; j++) {
            temp[i][j] = false;
        }
    }
}
function setTemp(board, temp) {
    for (var i = 0; i < 9; i++) {
        for (var j = 0; j < 9; j++) {
            if (board[i][j] != 0) {
                temp[i][j] = true;
            }
        }
    }
}
function setColor(temp) {

    for (var i = 0; i < 9; i++) {
        for (var j = 0; j < 9; j++) {
            if (temp[i][j] == true) {
                arr[i][j].style.color = "#DC3545";
            }

        }
    }
}
function resetColor() {
    for (var i = 0; i < 9; i++) {
        for (var j = 0; j < 9; j++) {
  arr[i][j].style.color = "green"
        }
    }
```

```
}
var board = [[], [], [], [], [], [], [], [], []]
let button = document.getElementById('generate-sudoku')
let solve = document.getElementById('solve')
console.log(arr)
function changeBoard(board) {
    for (var i = 0; i < 9; i++) {
        for (var j = 0; j < 9; j++) {
            if (board[i][j] != 0) {
                arr[i][j].innerText = board[i][j]
            }
            else
                arr[i][j].innerText = ''
        }
    }
}
button.onclick = function () {
    var xhrRequest = new XMLHttpRequest()
    xhrRequest.onload = function () {
        var response = JSON.parse(xhrRequest.response)
        console.log(response)
        initializeTemp(temp)
        resetColor()
        board = response.board
        setTemp(board, temp)
        setColor(temp)
        changeBoard(board)
    }
    xhrRequest.open('get', 'https://sugoku.herokuapp.com/board?difficulty=easy')
        xhrRequest.send()
}
function isSafe(board,r,c,no){
    for(var i=0;i<9;i++){
        if(board[i][c]==no || board[r][i]==no){
            return false;
        }
    }
    var sx = r - r%3;
    var sy = c - c%3
    for(var x=sx;x<sx+3;x++){
        for(var y=sy;y<sy+3;y++){
```

```
            if(board[x][y]==no){
                return false;
            }
        }
    }
    return true;
}
function solveSudokuHelper(board,r,c){
    if(r==9){
        changeBoard(board);
        return true;
    }
    //other cases
    if(c==9){
        return solveSudokuHelper(board,r+1,0);
    }
        if(board[r][c]!=0){
        return solveSudokuHelper(board,r,c+1);
    }
    for(var i=1;i<=9;i++){

        if(isSafe(board,r,c,i)){
            board[r][c] = i;
            var success = solveSudokuHelper(board,r,c+1);
            if(success==true){
                return true;
            }
            board[r][c] = 0;
        }
    }
    return false;
}
function solveSudoku(board) {
    solveSudokuHelper(board,0,0);
}


solve.onclick = function () {
    solveSudoku(board)
}
```

3.4 Output

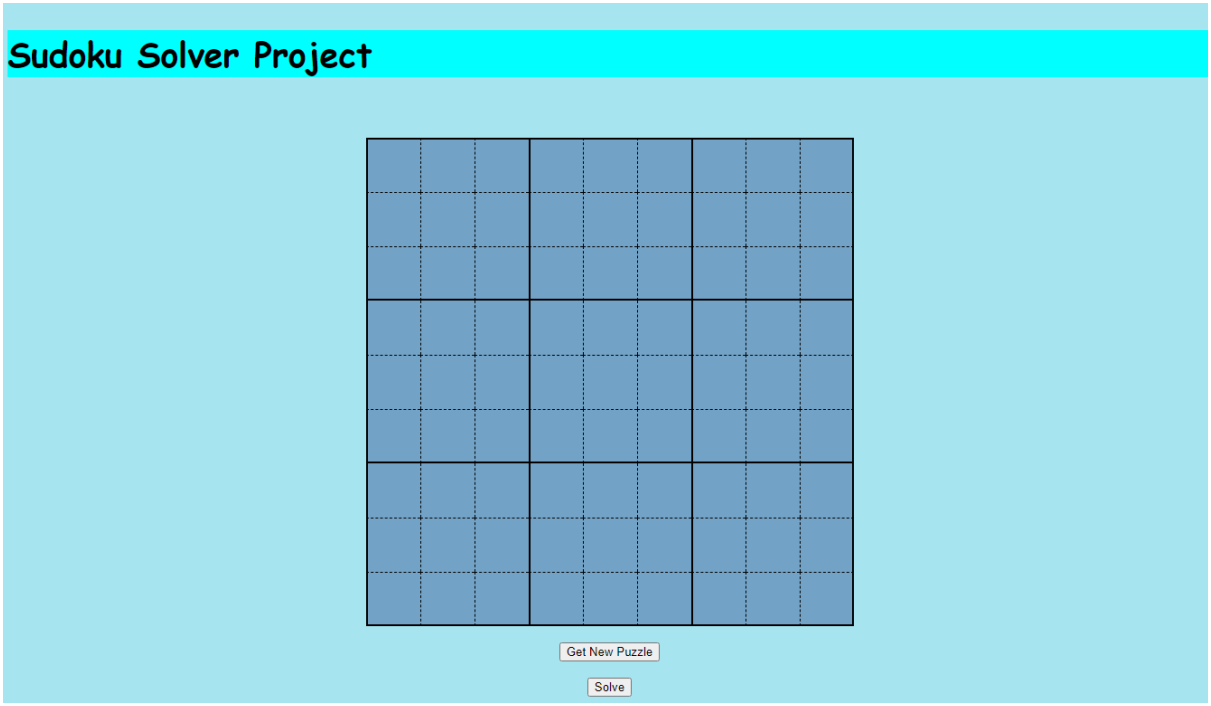


Fig 2 The web page of Sudoku Solver Game

After clicking Get New Puzzle Button

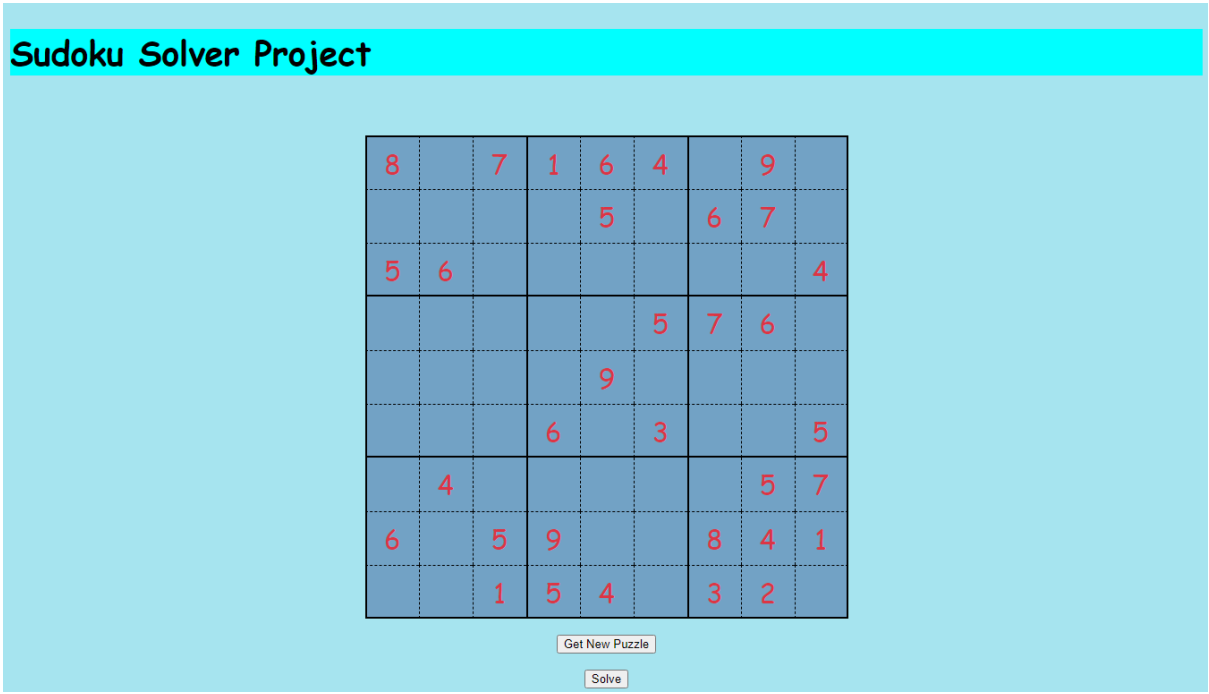
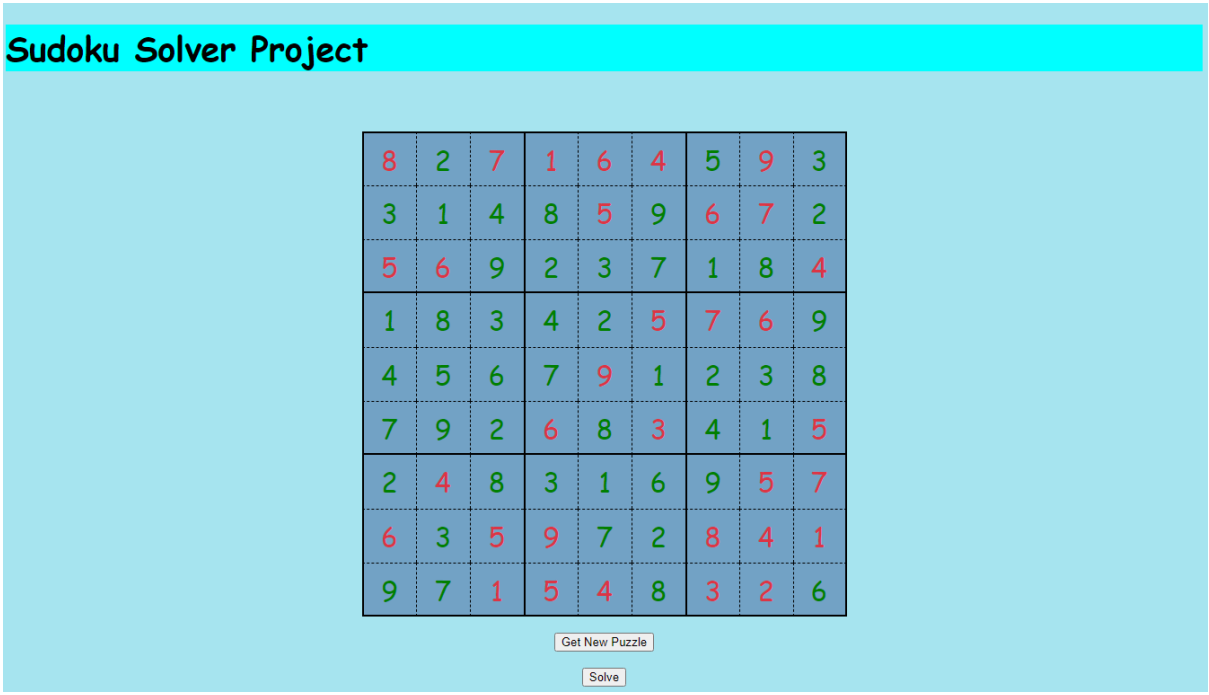
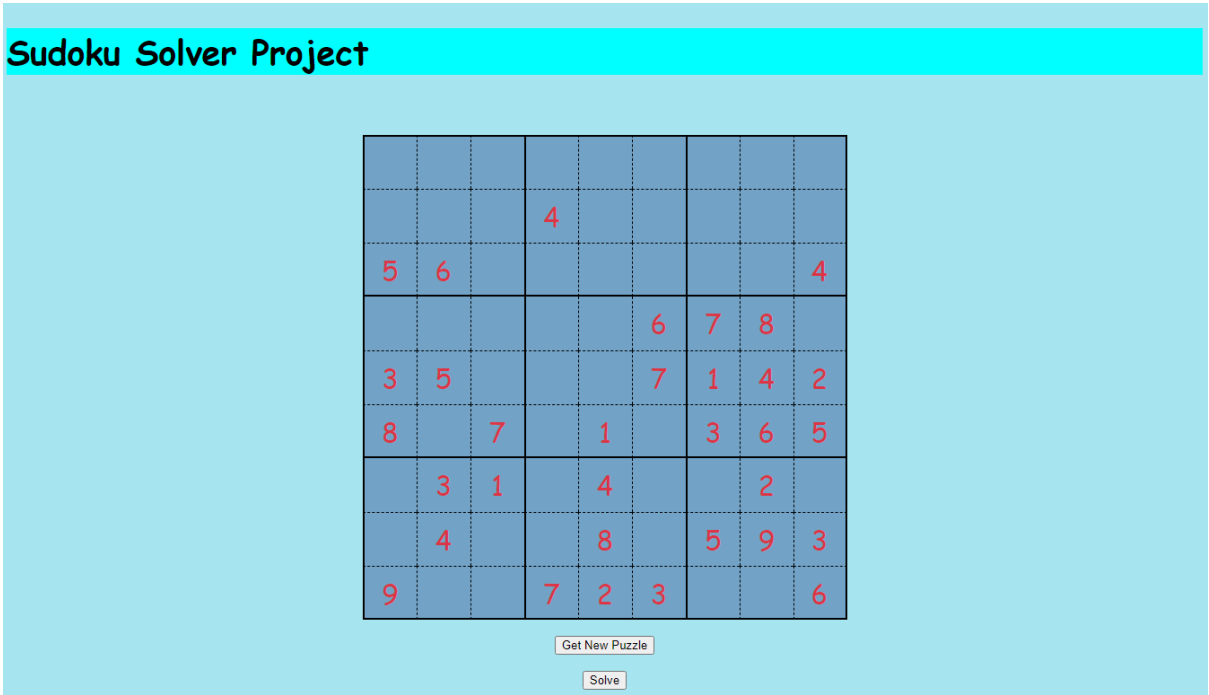


Fig 3 The Puzzle

After Clicking Solve Button



For New puzzle we can click get New Puzzle Again



4. ANALYSIS AND RESULT

Complexity Analysis:

Brute force technique

- **Time complexity:** O(9^(n*n)).
  For every unassigned index there are 9 possible options so the time complexity is O(9^(n*n)).
- **Space Complexity:** O(n*n).

Backtracking Method

- **Time complexity:** O(9^(n*n)).
  For every unassigned index, there are 9 possible options so the time complexity is O(9^(n*n)). The time complexity remains the same but there will be some early pruning so the time taken will be much less than the naive algorithm but the upper bound time complexity remains the same.
- **Space Complexity:** O(n*n).

5. Conclusion

This study has shown that the backtracking algorithm is a feasible method to solve any Sudoku puzzles. The algorithm is also an appropriate method to find a solution faster and more efficient compared to the brute force algorithm. The proposed algorithm is able to solve such puzzles with any level of difficulties in a short period of time (less than one second). The testing results have revealed that the performance of the backtracking algorithm is better than the brute force algorithm with respect to the computing time to solve any puzzle. The brute force algorithm seems to be a useful method to solve any Sudoku puzzles and it can guarantee to find at least one solution. However, this algorithm is not efficient because the level of difficulties is irrelevant to the algorithm. In other words, the algorithm does not adopt intelligent strategies to solve the puzzles. This algorithm checks all possible solutions to the puzzle until a valid solution is found which is a time consuming procedure resulting an inefficient solver. As it has already stated the main advantage of using the algorithm is the ability to solve any puzzles and a solution is certainly guaranteed. Further research needs to be carried out in order to optimize the backtracking algorithm.

REFERENCES

1) Wikipedia [cited 2013 February 21], Web site: http://en.wikipedia.org/wiki/Sudoku
2) Home Of Logic Puzzles [cited 2013 February 22], Web Page: http://www.conceptispuzzles.com/index.aspx?uri=puzzle/sudoku/classic
3) J.F. Crook, A pencil and paper algorithm for solving Sudoku Puzzles, [Cited 2013 February 24], Winthrop University, Webpage: http://www.ams.org/notices/200904/tx090400460p.pdf
4) A.S. Showdhury, S. Skher Solving Sudoku with Boolean Algebra [Cited 2013 February 24], International Journal of Computer Applications, Peer-reviewed Research, Webpage: http://research.ijcaonline.org/volume52/number21/pxc3879024.pdf
5) N. Pillay, Finding Solutions to Sudoku Puzzles Using Human Intuitive Heuristics, South African Research Articles, Webpage: http://sacj.cs.uct.ac.za/index.php/sacj/article/viewArticle/111
6) Ch. Xu, W. Xu, The model and Algorithm to Estimate the Difficulty Levels of Sudoku Puzzles, Journal of Mathematics Research, 2009 Webpage: http://journal.ccsenet.org/index.php/jmr/article/viewFile/3732/3336
7) T. Davis, The Mathematics of Sudoku (http://www.geometer.org/index.html), Research Article, Webpage: http://share.dschola.it/castigliano/ips/Documentazione%20Progetto/Materiale%20Didattico/M atematica/1E/sudoku.pdf
8) The figures are taken from webpage: http://www.conceptispuzzles.com/index.aspx?uri=puzzle/sudoku/techniques
9) T. Kovacs, Artificial Intelligence through Search: Solving Sudoku Puzzles, Journal Papers, Webpage: http://www.cs.bris.ac.uk/Publications/Papers/2000948.pdf
10) Sudoku solver using brute force, visited in Mars 2013, https://github.com/olav/JavaSudokuSolver
11) The puzzle generator, visited in Mars 2013, websudoku.com/