LOVELY PROFESSIONAL UNIVERSITY

Sudoku Solver Visualizer

Suduko Solver Visualiser

Hars Srivastava

2024

PHAGWARA - 144411

PROJECT REPORT

TITLE - Sudoku Solver Visualizer

Name - Harsh Srivastava

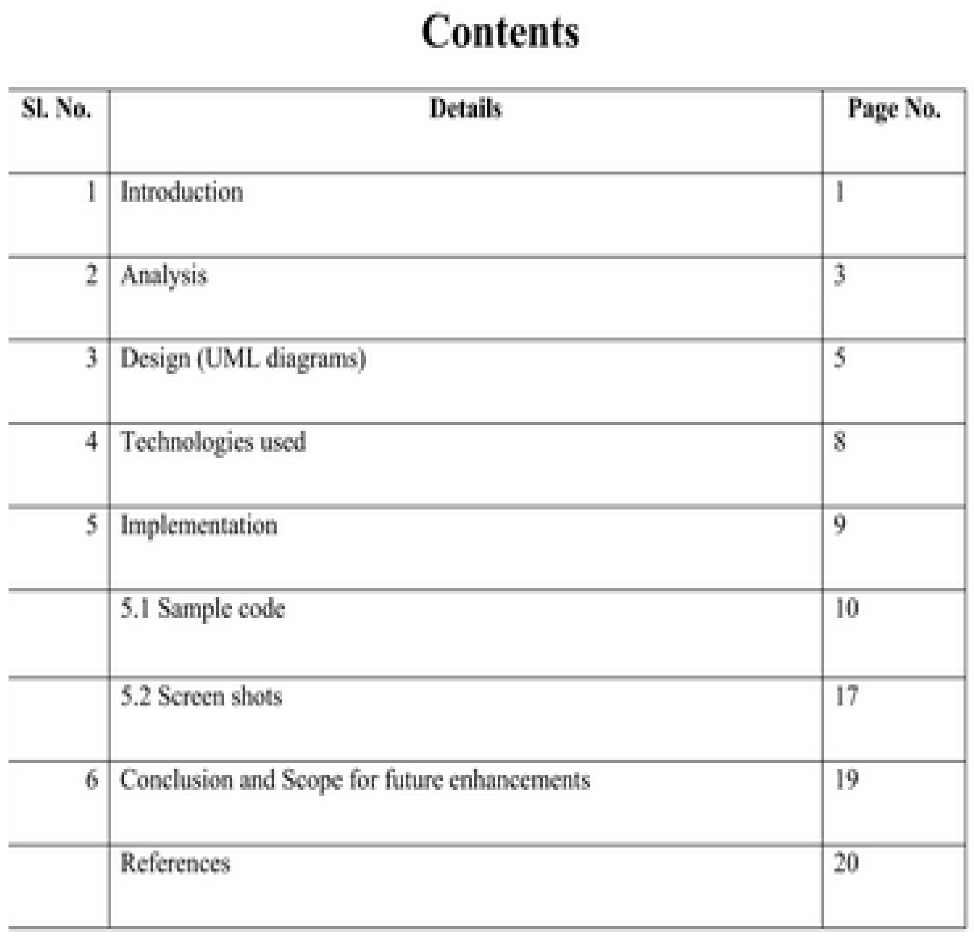
Virtual ID. - 12215211

ACKNOWLEDGMENT

I would like to express my immense gratitude and sincere thanks Rahul Singh in Information Technology for his guidance, valuable suggestions and encouragement in completing the Project work within the stipulated time.

ABSTRACT

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 mathematicalskills caused people to have enormous interest in accepting challenges to solvethe puzzle. Therefore, developers have tried to find algorithms in order togenerate the variety of puzzles for human players so that they could be evensolved by computer programming. In this essay, we have presented an algorithmcalled pencil-and-paper using human strategies. The purpose is to implement amore efficient algorithm and then compare it with another Sudoku solver namedas back tracking algorithm. This algorithm is a general algorithm that can beemployed in to any problems. The results have proved that the pencil-and-paperalgorithm solves the puzzle faster and more effective than the back tracking algorithm.



Contents

1.INTRODUCTION

1.1. Introduction

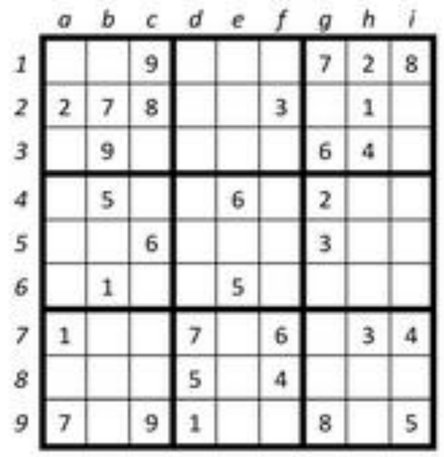
Currently, Sudoku puzzles are becoming increasingly popular among the people all over the world. The game has become popular now in a large number of countries and many developers have tried to generate even more complicated and more interesting puzzles. Today, the game appears in almost every newspaper, in books and in many websites. In this essay we present a Sudoku Solver named as pencil-and-paper algorithm using simple rules to solve the puzzles. The pencil-and-paper algorithm is formulated based on human techniques. This means that the algorithm is implemented based on human perceptions. Therefore the name of the solver is pencil-and-paper algorithm. The Brute force algorithm is then used 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. This algorithm generates any possible solutions until the right answer is found. The following subsections describe the problem statement, the purpose of this project, and the abbreviations and the definitions.

1.2 Problem Statement

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 essay develops an algorithm for solving Sudoku puzzle by using amethod, called pencil-and-paper algorithm. This algorithm resembles human methods, i.e. it describes how a person tries to solve the puzzle by using certain techniques. Our ambition is to implement the pencil-and-paper algorithm by using these techniques. 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 paper-and-pencil method and Brute force algorithm. Hopefully, by doing this work we might be able to answer the following questions: How does the pencil-and-paper algorithm differ from the Brute force algorithm? Which one of them is more effective? Is it possible to make these algorithms more efficient?

1.3 Purpose

The aim of the essay is to investigate the ba. Later these two methods are compared analytically. It is expected here to find an efficient method to solve the Sudoku puzzles. In this essay we have tried to implement the pencil-and-paper algorithm that simulate how human being would solve the puzzle by using some simple strategies that can be employed to solve the majority of Sudoku.



	a	b	c	d	e	f	g	h	i
1			9				7	2	8
2	2	7	8			3		1	
3		9					6	4	
4		5			6		2		
5			6				3		
6		1			5				
7	1			7		6		3	4
8				5		4			
9	7		9	1			8		5

Fig.1 *An example of a Sudoku puzzle.*

1.4. Abbreviations and Definitions

n this essay we have tried to use the same terminology, which is commonly used in other journals and research papers. In the following paragraph, there is a brief description of some the abbreviations and definitions that are used in the text. Sudoku: is a logic-based, combinatorial number placement puzzle . The word "Sudoku" is short for Su-ji wa dokushin ni kagiru (in Japanese), which means "the numbers must be single", see Fig.1.

Box (Region, Block): a region is a 3x3 box like the one shown in figure 1. There are 9 regions in a traditional Sudoku puzzle.

Grid (board): the Sudoku board consists of a form of matrix or windows. (Square): is used to define the minimum unit of the Sudoku board. Candidates: the number of possible values that can be placed into an empty square.

Clues: the given numbers in the grid at the beginning.

Grid (board): the Sudoku board consists of a form of matrix or windows.

2. ANALYSIS

This section starts with an explanation about Sudoku. Then a research is carried out on the previous works about this subject. Later, we discuss further about evaluated algorithms and finally a description of how the work is being carried out is presented.

2.1 Short about Sudoku

Sudoku is a logic-based puzzle that is played by numbers from 1 to 9. The Puzzle first appeared in newspapers in November 1892 in France and then Howard Garns an American architect presented it in its modern form There are already many journals, papers and essays that researched about Sudoku Solvers and most of them present different type of algorithms. Sudoku's popularity is based on several reasons. First of all it is fun and fascinating, and very easy to learn because of its simple rules. There are currently many different type of Sudoku puzzles, classic Sudoku that contains a 9X9 grid with given clues in various places, mini Sudoku that consists of a grid with 4X4 or 6X6 sizes. The other type of Sudoku is Mega Sudoku that contains a grid with 12X12 or 16X16 sizes In this text, the focus is mostly on the classic Sudoku, i.e. 9X9 grid. Furthermore, Sudoku has become so popular, compared to other games, all over the world because its rules are easy to understand and it can improve our brain and also it is fun. The structure of the puzzle is very simple, especially the classic puzzle. This essay is mainly focused on classic puzzle of a 9X9 grid. There already exist a number of digits in the board that make the puzzle solvable. It means that some numbers are already placed in the Sudoku board before starting playing. The board consists of 81 cells, which is divided into nine 3X3 sub boards and each 3X3 sub board is called "box" or "region". The main concept of the game is to place numbers from 1 to 9 on a 9X9 board so that every row, column and box contains any numbers but once. This means that no number is repeated more than once. An example of this game is illustrated in Fig.1. Generally, the puzzle has a unique solution. There are certain techniques to solve the puzzle by hand and these rules can be implemented into a computer program. These techniques are presented in more details in 2.3.

2.2 Previous Research

We have noticed that there is a large volume of published studies describing Sudoku problems. Furthermore, several research have been made to solve Sudoku problems in a more efficient way [4]. It has conclusively been shown that solving the puzzle, by using different algorithms, is definitely possible but most developers seek for optimizations techniques such as genetic algorithms, simulated annealing, etc.

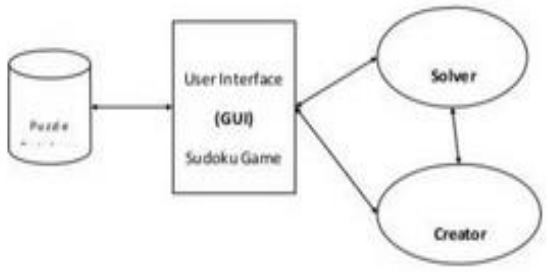
Different authors have made relative works already. Nelishia Pillay gives a solution for solving Sudoku by combining human intuition and optimization. This author has investigated the use of genetic programming to improve a space of programs combined of the heuristics moves. However, we seek a solution to solve Sudoku puzzle based on human strategies, which uses techniques such as: naked single method, hidden single method etc. J.F. Crook have also discussed about solving Sudoku and presented an algorithm on how to solve the puzzles of differing difficulty with pencil-and-paper algorithm. This method has not been implemented and therefore it is hard to discuss how the algorithm performs . Tom Davis has done a research about "The Mathematics of Sudoku". Tom has described all techniques that people usually use to solve the puzzles but his major attempt is to describe these techniques from mathematical perspective. However, all the strategies he mentions are not required to solve the puzzle. For instance the easy puzzles can be solved using only one or two strategies.

2.3 Algorithm: Back Tracking

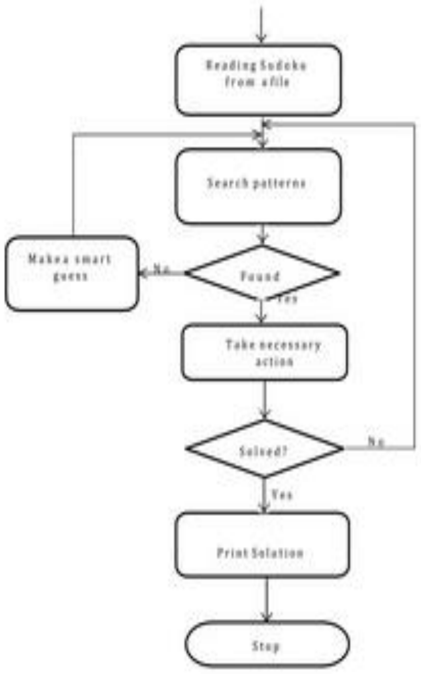
The unique missing method and the naked single method are able to solve all puzzles with easy and medium level of difficulties. In order to solve puzzles with even more difficult levels such as hard and evil the backtracking method has been used to complete the algorithm. A human player solves the puzzle by using simple techniques. If the puzzle is not solvable by using the techniques the player then tries to fill the rest of the empty squares by guessing. The backtracking method, which is similar to the human strategy (guessing), is used as a help method to the pencil-and-paper algorithm. In other words, if the puzzle cannot be filled when using the unique missing method and the naked single method, the backtracking method will take the puzzle and fill the rest of empty squares. Generally, the backtracking method find empty square and assign the lowest valid number in the square once the content of other squares in the same row, column and box are considered. However, if none of the numbers from 1 to 9 are valid in a certain square, the algorithm backtracks to the previous square, which was filled recently. The above-mentioned methods are an appropriate combination to solve any Sudoku puzzles. The naked single method can find quickly single candidates to the empty squares that needed only one single value. Since the puzzle comes to its end solution the unique missing method can be used to fill rest of the puzzles. Finally, if either method fills the board the algorithm calls the backtracking method to fill the rest of the board.

3.DESIGN

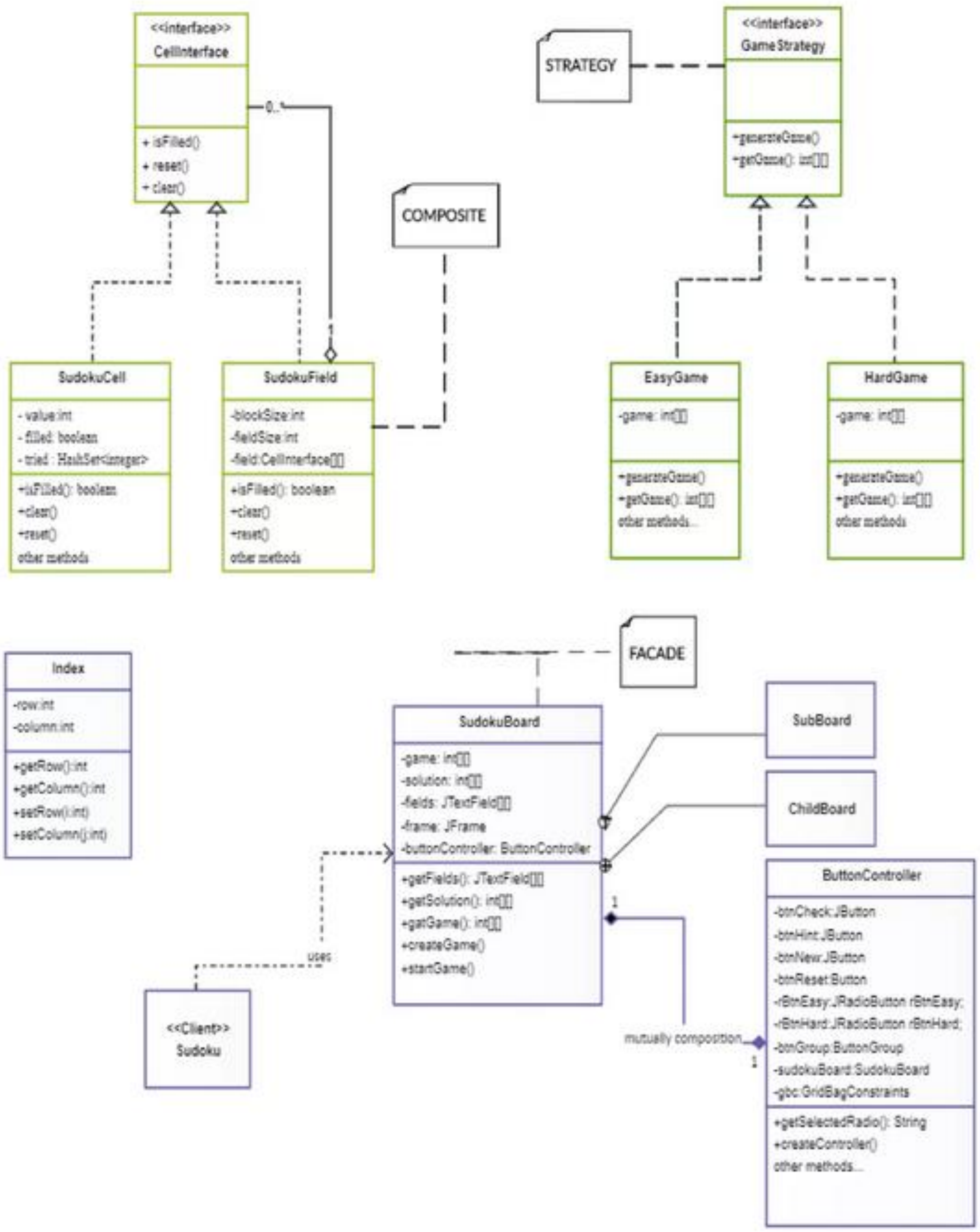
3.1 Block Diagram



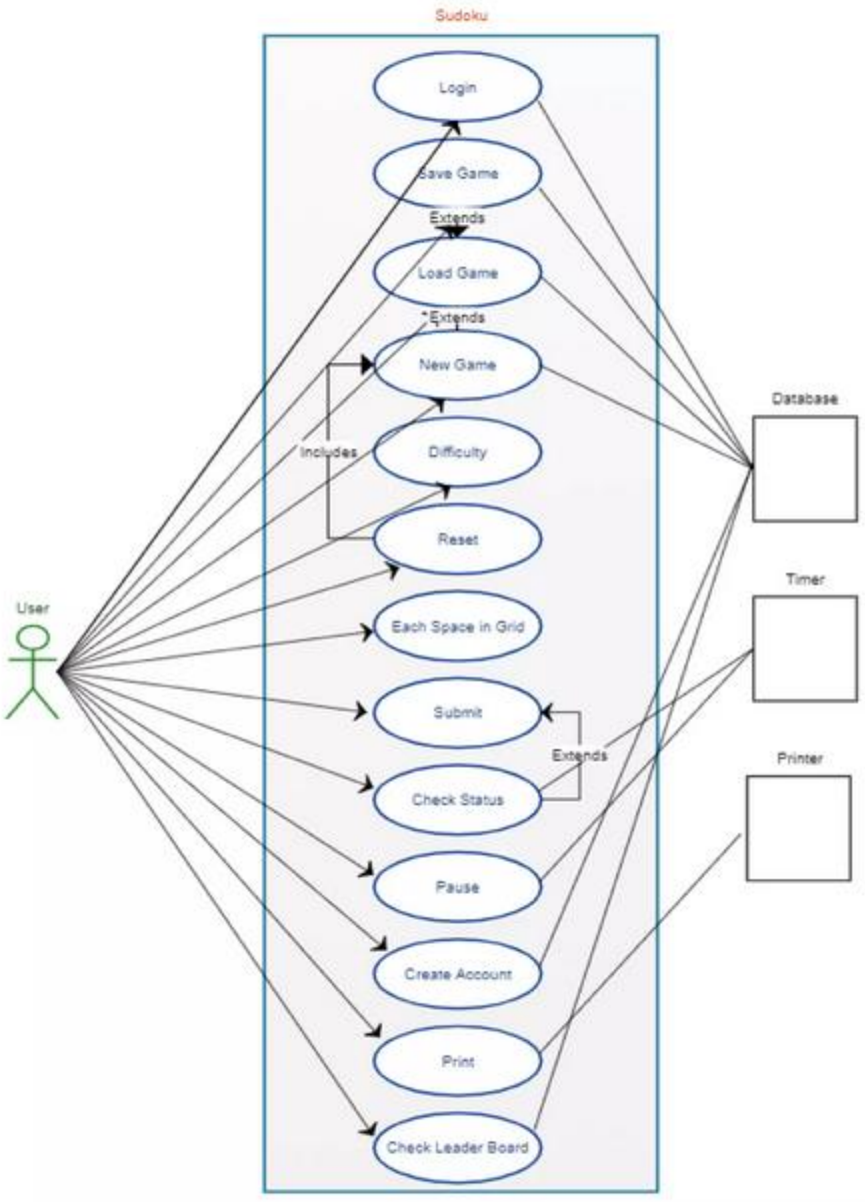
3.2 Flow Chart



3.3 Class Diagram



3.4 Use Case Diagram



5. IMPLEMENTATION

In some recursive functions, such as binary search or reversing a file, each recursive call makes just one recursive call. The "tree" of calls forms a linear line from the initial call down to the base case. In such cases, the performance of the overall algorithm is dependent on how deep the function stack gets, which is determined by how quickly we progress to the base case. For reverse file, the stack depth is equal to the size of the input file, since we move one closer to the empty file base case at each level. For binary search, it more quickly bottoms out by dividing the remaining input in half at each level of the recursion. Both ofthese can be done relatively efficiently.

Now consider a recursive function such as subsets or permutation that makes not just one recursive call, but several. The tree of function calls has multiple branches at each level, which in turn have further branches, and so on down to the base case. Because of the multiplicative factors being carried down the tree, the number of calls can grow dramatically as the recursion goes deeper. Thus, these exhaustive recursion algorithms have the potential to be very expensive. Often the different recursive calls made at each level represent a decision point, where we have choices such as what letter to choose next or what turn to make when reading a map. Might there be situations where we can save some time by focusing on the most promising options, without committing to exploring them all?

In some contexts, we have no choice but to exhaustively examine all possibilities, such as when trying to find some globally optimal result, But what if we are interested in finding any solution, whichever one that works out first? At each decision point, we can choose one of the available options, and sally forth, hoping it works out. If we eventually reach our goal from here, we're happy and have no need to consider the paths not taken. However, if this choice didn't work out and eventually leads to nothing but dead ends; when we backtrack to this decision point, we try one of the other alternatives.

What's interesting about backtracking is that we back up only as far as needed to reach a previous decision point with an as-yet-unexplored alternative. In general, that will be at the most recent decision point. Eventually, more and more of these decision points will have been fully explored, and we will have to backtrack further and further. If we backtrack all the way to our initial state and have explored all alternatives from there, we can conclude the particular problem is unsolvable. In such a case, we will have done all the work of the exhaustive recursion and known that there is no viable solution possible.

As was the case with recursion, simply discussing the idea doesn't usually make the concepts transparent, it is therefore worthwhile to look at many examples until you begin to see how backtracking can be used to solve problems. This handout contains code for several recursive backtracking examples. The code is short but dense and is somewhat sparsely commented, you should make sure to keep up with the discussion in lecture. The fabulous maze backtracking example is fully covered in the reader as an additional example to study.

5.1 Sample Code

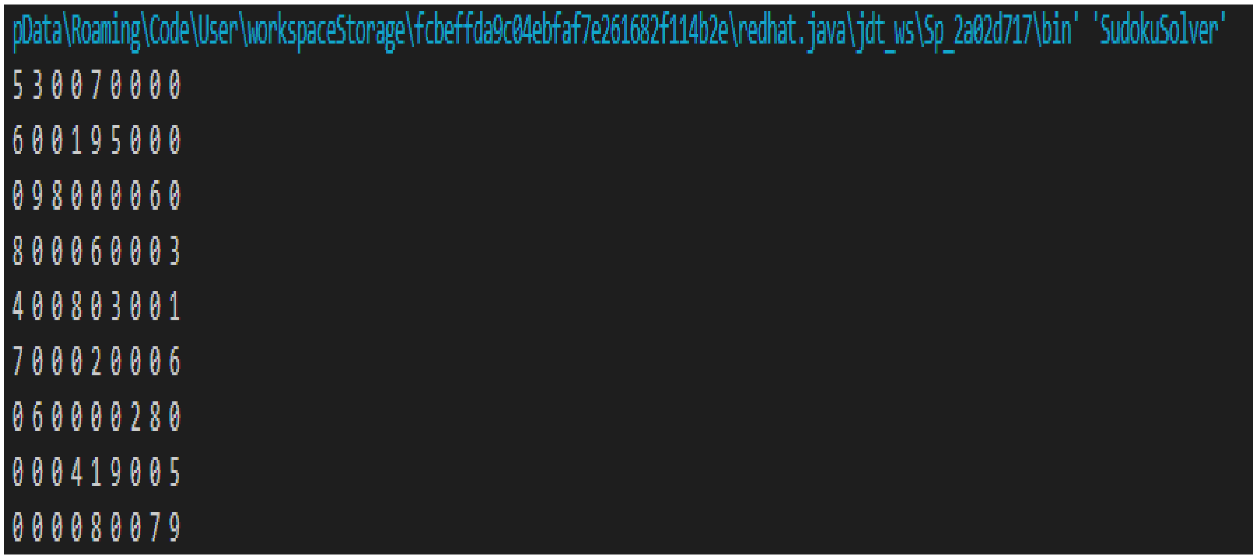
```
import javax.swing.*;

import java.awt.*;

import javax.swing.border.Border;

import javax.swing.border.LineBorder;


public class SudokuSolver {


    final static int N = 9;  // Size of the Sudoku grid

    static JLabel[][] jLabel = new JLabel[N][N];


    static int board[][] = {

        {5, 3, 0, 0, 7, 0, 0, 0, 0},

        {6, 0, 0, 1, 9, 5, 0, 0, 0},

        {0, 9, 8, 0, 0, 0, 0, 6, 0},

        {8, 0, 0, 0, 6, 0, 0, 0, 3},

        {4, 0, 0, 8, 0, 3, 0, 0, 1},

        {7, 0, 0, 0, 2, 0, 0, 0, 6},

        {0, 6, 0, 0, 0, 0, 2, 8, 0},

        {0, 0, 0, 4, 1, 9, 0, 0, 5},
        {0, 0, 0, 0, 8, 0, 0, 7, 9}

    };


    static int board2[][] = {

        {0, 0, 0, 6, 0, 0, 4, 0, 0},

        {7, 0, 0, 0, 0, 3, 6, 0, 0},

        {0, 0, 0, 0, 9, 1, 0, 8, 0},

        {0, 0, 0, 0, 0, 0, 0, 0, 0},

        {0, 5, 0, 1, 8, 0, 0, 0, 3},

        {0, 0, 0, 3, 0, 6, 0, 4, 5},

        {0, 4, 0, 2, 0, 0, 0, 6, 0},

        {9, 0, 3, 0, 0, 0, 0, 0, 0},

        {0, 2, 0, 0, 0, 0, 1, 0, 0}

    };


    static void printSolution() {

        for (int i = 0; i < N; ++i) {

            for (int j = 0; j < N; ++j) {

                System.out.printf("%d ", board[i][j]);

            }

            System.out.printf("\n");

        }
```

```
}

static boolean isSafe(int row, int col, int num) {
    try {
        Thread.sleep(1);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    // Check the row
    for (int x = 0; x < N; x++) {
        if (board[row][x] == num) {
            return false;
        }
    }

    // Check the column
    for (int x = 0; x < N; x++) {
        if (board[x][col] == num) {
            return false;
        }
    }
```

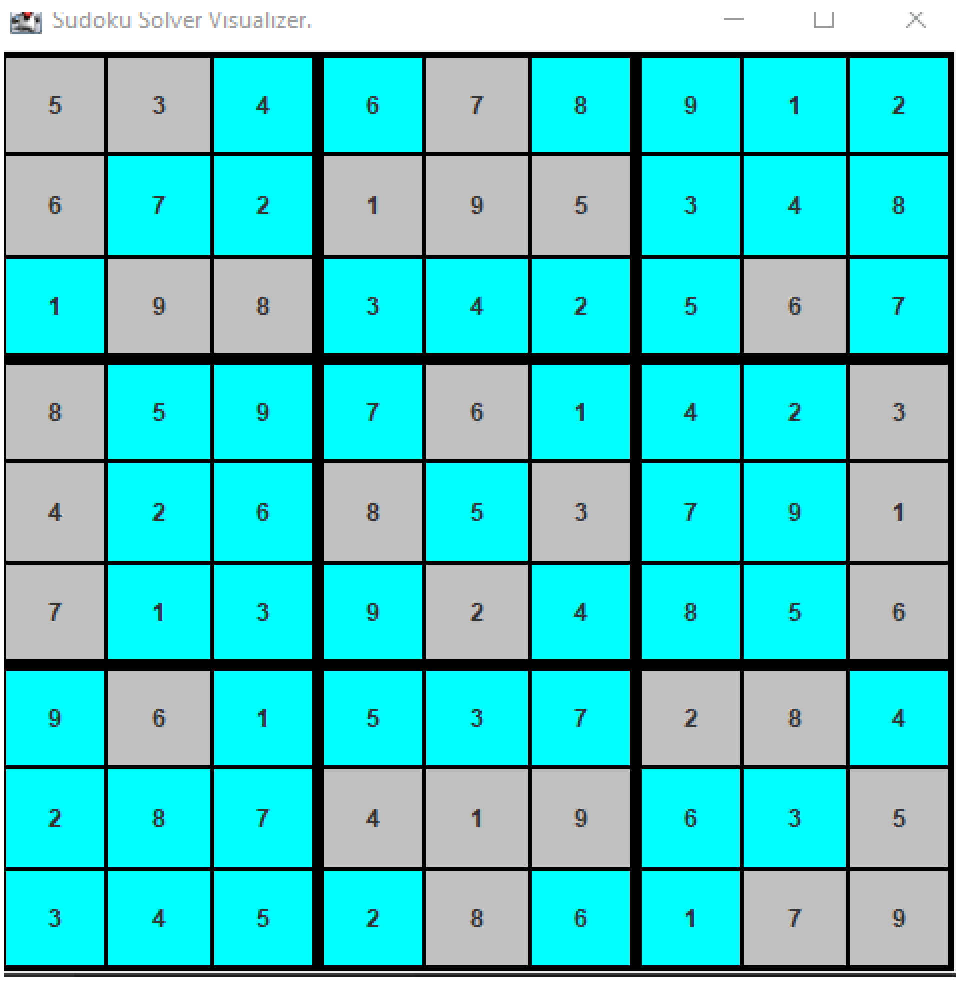
```
        // Check the 3x3 subgrid
        int startRow = row - row % 3, startCol = col - col % 3;
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                if (board[i + startRow][j + startCol] == num) {
                    return false;
                }
            }
        }

        return true;
    }

    static boolean findSolution(int row, int col) {
        if (row == N - 1 && col == N) {
            return true;
        }

        if (col == N) {
            row++;
            col = 0;
        }


        if (board[row][col] != 0) {

            return findSolution(row, col + 1);

        }


        for (int num = 1; num <= N; num++) {

            if (isSafe(row, col, num)) {

                board[row][col] = num;

                jLabel[row][col].setText(String.valueOf(num));

                jLabel[row][col].setBackground(Color.CYAN);


                if (findSolution(row, col + 1)) {

                    return true;

                }


                board[row][col] = 0;

                jLabel[row][col].setText("0");

                jLabel[row][col].setBackground(Color.RED);

            }

        }
```

```
        return false;
    }


    static void solveSudoku() {
        try {
            Thread.sleep(200);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }


        for (int i = 0; i < N; ++i) {
            for (int j = 0; j < N; ++j) {
                try {
                    Thread.sleep(10);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }


                jLabel[i][j].setText(board[i][j] == 0 ? "0" : String.valueOf(board[i][j]));
                jLabel[i][j].setBackground(Color.LIGHT_GRAY);
            }
        }
        if (!findSolution(0, 0)) {
            System.out.println("No Solution.\n");
        } else {
            printSolution();
        }
    }

    SudokuSolver() {
        JFrame jFrame = new JFrame("Sudoku Solver Visualizer.");
        jFrame.setLayout(new GridLayout(N, N));
        jFrame.setSize(500, 500);
        jFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        for (int i = 0; i < N; ++i) {
            for (int j = 0; j < N; ++j) {
                jLabel[i][j] = new JLabel("0");
                jLabel[i][j].setHorizontalAlignment(SwingConstants.CENTER);
                jLabel[i][j].setSize(50, 50);
                jLabel[i][j].setOpaque(true);

                int top = (i % 3 == 0) ? 3 : 1;
                int left = (j % 3 == 0) ? 3 : 1;

                int bottom = ((i + 1) % 3 == 0) ? 3 : 1;

                int right = ((j + 1) % 3 == 0) ? 3 : 1;


                jLabel[i][j].setBorder(BorderFactory.createMatteBorder(top, left, bottom, right,
Color.BLACK));


                jFrame.add(jLabel[i][j]);

            }

        }


        jFrame.setVisible(true);

    }


    public static void main(String args[]) {

        SwingUtilities.invokeLater(new Runnable() {

            @Override

            public void run() {

                new SudokuSolver();

            }

        });

        printSolution();

        System.out.println();
        solveSudoku();


        printSolution();
    }
}
```

5.1. Sample Output



```
pData\Roaming\Code\User\workspaceStorage\fcbeffda9c04ebfaf7e261682f114b2e\redhat.java\jdt_ws\Sp_2a02d717\bin' 'SudokuSolver'
5 3 0 0 7 0 0 0 0
6 0 0 1 9 5 0 0 0
0 9 8 0 0 0 0 6 0
8 0 0 0 6 0 0 0 3
4 0 0 8 0 3 0 0 1
7 0 0 0 2 0 0 0 6
0 6 0 0 0 0 2 8 0
0 0 0 4 1 9 0 0 5
0 0 0 0 8 0 0 7 9
```



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

6.CONCLUSION AND FUTURE ENHANCEMENTS

Conclusion

This study has shown that the pencil-and-paper algorithm is a feasible method to solve any doku puzzles. The algorithm is also an appropriate method to find a solution faster and more efficient compared to the back tracking 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 pencil-and-paper algorithm is better than the back tracking algorithm with respect to the computing time to solve any puzzle.

The back tracking 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.

Future Enhancements

Further research needs to be carried out in order to optimize the pencil-and-paper algorithm. A possible way could be implementing of other human strategies (x-wings, swordfish, etc.). Other alternatives might be to establish whether it is feasible to implement an algorithm based only on human strategies so that no other algorithm is involved in the pencil-and-paper algorithm and also make sure that these strategies can solve any puzzles with any level of difficulties