SUDOKU GAME

A PROJECT REPORT

Submitted by

G. VAMSHI [Reg No: RA2112704010017]

Under the Guidance of

Dr. Elangovan

(Associate Professor, Department of Information Technology)

in partial fulfilment for the award of the degree

of

BACHELOR OF TECHNOLOGY

DEPARTMENT OF INFORMATION TECHNOLOGY

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

DEC 2022

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR-603203

BONAFIDE CERTIFICATE

Certified that this project report title “SCHOOL DATABASE MANAGEMENT SYSTEM” is the bonafide work of “G. VAMSHI [RA2112704010017]”, who carried out the project work under my supervision. Certified further, that to the best of knowledge the work report herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

<<Signature of the Supervisor>>

<<Signature>>

Signature of Internal Examiner

Signature of External Examine

ABSTRACT

In this report, we present the detailed development and implementation of simple Sudoku game. The Sudoku game consists of graphical user interface, solver and puzzle generator; implemented using java and java swings. The solver and generator is implemented using efficient algorithm. The solver finds the solution to the puzzles generated by the generator as well as to the puzzles entered by the user. Generator creates various number of different Sudoku puzzles. This project gives an insight in to the different aspects of java programming.

Sudoku puzzle is one of the most popular games that are helpful to improve intellectual development in the world. And it also engages many scholars focus on solving algorithms and grading methods based on computers. In this paper, Sudoku puzzle is studied based on customized information entropy. An algorithm is designed to solve Sudoku puzzles based on the customized conception of information entropy and corresponding prototype is implemented in C++ by object-oriented programming method. The definitions of inverse information entropy and information amount for inverse information entropy are introduced and directly used instead of information entropy in order to simplify the solving procedure. Experimental results show that the algorithm has better time efficiency than available methods including generic algorithms and rule based algorithms and it can solve not only unique-solution puzzles (including extremely difficult puzzles) but also multiple-solution puzzles. Furthermore, it is a feasible choice to grade difficulty of Sudoku puzzles based on the customized information entropy.

ACKNOWLEDGEMENTS

TABLE OF CONTENT

LIST OF FIGURES

LIST OF ABREVATIONS

JDK	Java Development Kit.
JVM	Java Virtual Machine.
JRE	Java Runtime Environment.
API	Application Programming Interface.
NLP	Natural Language Processing.
GMF	Graphical Modelling Framework.

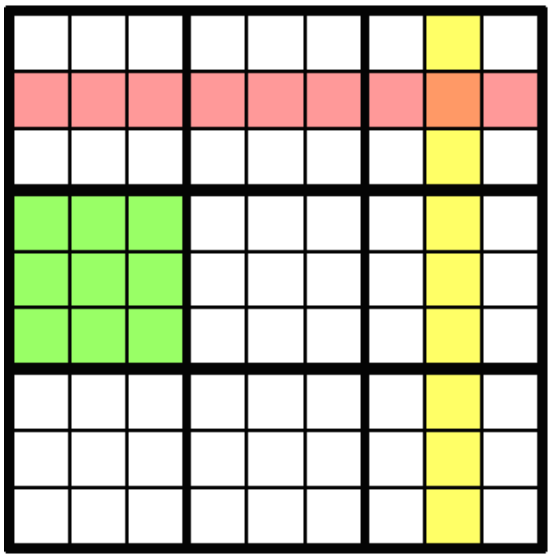
CHAPTER 1

INTRODUCTION

1.1 GENERAL

Sudoku is the Japanese abbreviation of a phrase meaning the digits must remain single, also known as Number Place, where Su means number, doku which translates as single or bachelor. Sudoku is not a mathematical or arithmetical puzzle. It works just as well if the numbers are substituted with letters or some other symbols, but numbers work best. The aim of the puzzle is to enter a numerical digit from 1 through 9 in each cell of a 9×9 grid made up of 3×3 subsquares or subgrids, starting with various digits given in some cells; each row, column, and subsquares region must contain each of the numbers 1 to 9 exactly once.

Throughout this document we refer to the whole puzzle as the grid/game board, a 3x3 subgrid as a block and the individual grids that contains the number as a cell.



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

Figure : Sample Sudoku game

1.2 HISTORY

The first Sudoku puzzle was created in 1979. In New York City the Sudoku puzzle appear first, which was published by the specialist puzzle publisher Dell Magazines in their magazine Dell Pencil Puzzles and word Games. First it was printed under the name “number place”. Howard Garns, a retired architect and freelance puzzle constructor designed this first puzzle. This mathematical construction is inspired by the Latin square, invention of Leonhard Euler. Later the puzzle was introduced in Japan by Nikoli during 1984 as "Suji wa dokushin ni kagiru", which can be translated as "the numbers must be single" or "the numbers must occur only once", later it abbreviated as Sudoku. The Sudoku puzzle can use symbols or colors instead of numerals.Sudoku is still a trademark owned by Nikoli. The the game's popularity really took off in 2005; it can now be found in many newspapers and magazines around the world.

1.3 RULES

Solving a Sudoku puzzle can be rather tricky, but the rules of the game are quite simple. Solving a sudoku puzzle does **not** require knowledge of mathematics; simple logic suffices. The objective of sudoku is to enter a digit from 1 through 9 in each cell, in such a way that:

I. Each horizontal **row** contains each digit exactly **once**

II. Each vertical **column** contains each digit exactly **once**

III. Each sub grid or **region** contains each digit exactly **once**

1.4 PURPOSE

The purpose of this report is that it provides technical documentation regarding a game named Simple Sudoku. Simple Sudoku is a simple game written using **Java**. This technical report identifies all features and lists out the functionalities and working environment in which the application (game) can be executed.

This report lists out the overall description of the game with essential features.

- **Improves concentration:**

It is impossible to solve a Sudoku puzzle without concentration. Since this game requires logical thinking, an interruption can break the chain of t thought and force the player to restart their analysis. The frustration of having to constantly go back to square one in order to progress will eventually train the brain to block any source of distraction.

The more puzzles you play, the more absorbed in your task you will be each time, improving your concentration skills step by step.

This better-developed skill will not only be felt when playing Sudoku but will also transpire to other activities in your life, be it at work, studying or performing a task that requires your full attention.

- **Helps to reduce anxiety and stress:**

The two big bad words of today's society can be tamed by a simple numbers puzzle. One of the benefits of Sudoku is that it requires the player

to concentrate on the grid and use logical thinking to find the solution for each cell. While doing this, the brain becomes fully focus on the task at hand rather than the source of stress and anxiety.

This break can be just enough for the player to regain their sense of balance and become calmer. Once the puzzle is over, they might even find that the task or the problem that generated so much anxiety is not as daunting as initially seemed.

- **Promotes a healthy mindset:**

  When the brain is not stimulated, it tends to dwell on negative thoughts and infuse the person with an overall sense of unhappiness.

  Just as exercising can boost your vitality, so does playing challenging thinking games like Sudoku. A fitter and happier brain is the first step to regard the world and your life with a healthier and more positive mindset.

- **Helps kids develop their problem-solving skills:**

  Sudoku is a puzzle with simple, easy to understand rules that any kid can try. The need to engage in logical thinking to fill the grid correctly plus the process of trial and error they must apply will naturally and unconsciously help to develop their problem-solving skills.

  These benefits of Sudoku can also help them in other areas and even improve their school performance.

- **Improves thinking skills:**

  As far as the benefits of Sudoku go, improved thinking skill is likely one of the first players experience.

  In the initial stages, solving a puzzle can be a chaotic process and you are likely to jump from the analysis of rows and columns to groups randomly. However, the brain will instinctively begin to find patterns of solutions. As the game progresses, you will come to understand which

elements and which patterns are more likely to result in a quicker and easier solution.

Slowly, you will begin to apply this improved skill in your daily life too, and you will be able to identify more efficiently the best way to attain the outcome you desire.

1.5 SCOPE OF PROJECT

The objectives of the proposed Project are to increase the Thinking Capability.

The Game having all the records which u perform in playing you can Select Easy, hard level according to your choice. You can make your own Sudoku and at any Step you can go back to One Step as well as you can see the Solution of it.

To provide information on the various levels of functionality of the Simple Sudoku application. Graphical representation of the major components of the application. External interface features use on the application.

Helps in understanding the application (game). Noticing key intricacies in the game.

It is manually a very difficult job to perform and its need a lot of recalling, reminding and mathematical calculation. The game of "Sudoku" helps to increase mental thinking, vision etc.

1.6 RESEARCH OBJECTIVES

Undergraduate research project work is very demanding of students, of tutors and of resources and many students and the transition from traditional practical work difficult. In particular, they have unrealistic expectations of what can be achieved. In order to prepare students for their project work, some second-year courses include mini-projects. This paper reports on a case study of one such mini-project: it was

effective in preparing students for their project work but most students were unaware of this and as a result, many felt demoralized by their experience. A number of factors which might improve the effectiveness of mini-projects and reduce the students' negative feelings were identified including: making the aims and objectives unambiguous, achievable and explicit; recognizing the nature and difficulty of the demands which are being made of students; and providing sufficient time, support and guidance for students.

1.7 PROJECT LIMITATION

The Problem Faced by this System “Sudoku Game” is this that it only run-in desktop.

If you want to access this game then you must have java 5.0 installed in your desktop. The Limitation of this project that you not access it in your Mobile device or in android device.

CHAPTER 2

STUDY OF EXISTING SYSTEM

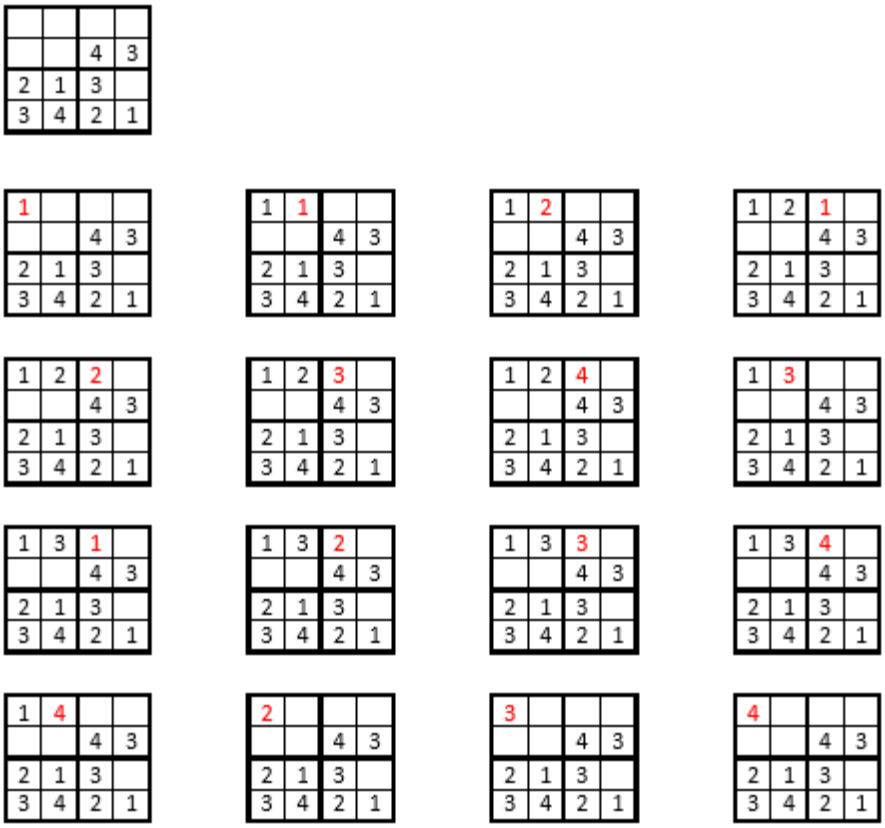
2.1 CASE STUDY

Brute Force Solving Method:

A simple way to solve a Sudoku puzzle is to simply try filling each blank square with the numbers 1 to 9 until a valid solution is found. I ended up devising three different implementations of this method, as shown below.

Method 1:

The first and most naïve solver starts at the top-left square and moves from left-to-right, top-to-bottom, filling each blank square with a number 1 to 9 until the grid is invalid (that is, a number is duplicated in a row, column or 3×3 region) or until the grid is filled and valid (that is, solved). If the grid is invalid, the solver will backtrack until it is valid and continue forward again.



4	1		
		4	3
2	1	3	
3	4	2	1

4	2		
		4	3
2	1	3	
3	4	2	1

4	2	1	
		4	3
2	1	3	
3	4	2	1

4	2	1	1
		4	3
2	1	3	
3	4	2	1

4	2	1	2
		4	3
2	1	3	
3	4	2	1

4	2	1	3
		4	3
2	1	3	
3	4	2	1

4	2	1	4
		4	3
2	1	3	
3	4	2	1

4	2	2	
		4	3
2	1	3	
3	4	2	1

4	2	3	
		4	3
2	1	3	
3	4	2	1

4	2	4	
		4	3
2	1	3	
3	4	2	1

4	3		
		4	3
2	1	3	
3	4	2	1

4	3	1	
		4	3
2	1	3	
3	4	2	1

4	3	1	1
		4	3
2	1	3	
3	4	2	1

4	3	1	2
		4	3
2	1	3	
3	4	2	1

4	3	1	2
1		4	3
2	1	3	
3	4	2	1

4	3	1	2
1	1	4	3
2	1	3	
3	4	2	1

4	3	1	2
1	2	4	3
2	1	3	
3	4	2	1

4	3	1	2
1	2	4	3
2	1	3	1
3	4	2	1

4	3	1	2
1	2	4	3
2	1	3	2
3	4	2	1

4	3	1	2
1	2	4	3
2	1	3	3
3	4	2	1

4	3	1	2
1	2	4	3
2	1	3	4

4	3	1	2
1	2	4	3
2	1	3	4

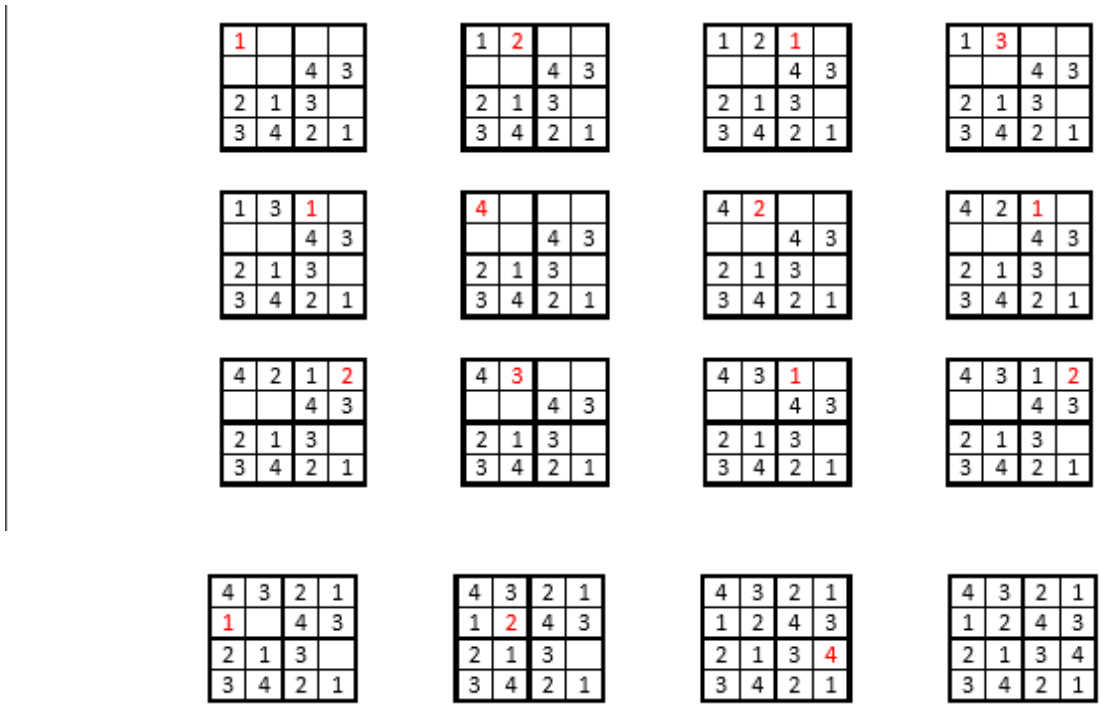
Method 2:

The second implementation introduces the concept of 'domains'. Each square in a grid has a domain of up to 9 values (1 to 9) that is reduced according to numbers already present in the intersecting row, column and region.

In this grid, we can reduce the domain of the top-left square to {1,4} since 2 and 3 already appear in the first column.

Likewise, we can restrict the domain of the top-right square to {2}, since 3 and 1 occur in the rightmost column and 4 appears in the top-right region.

This solver acts in the same way as the previous implementation except that it restricts the domains of the grid before it starts, only using numbers present in the initial domains. While the domain restriction requires computation, it should result in significantly less backtracking to find the solution of a grid. Illustrating the process:



2.2 PROPOSED SYSTEM

Minimum Requirement:

CPU Speed:	1.4 GHz
RAM:	512 MB
Hard Disk:	80 GB
OS:	Windows 10/9/8/2000/XP
Mouse:	
Keyboard:	
Sound Card:	NO
Language:	Java 5.0

CHAPTER 3

PROPOSED METHODOLOGY

3.1 Requirement for User:

The basic requirement was to create a simple Sudoku game, where user can able to get different puzzles, able to check and get solution for those puzzles, able to get solution for their own puzzles. Considering this requirement, a simple 9x9 grid layout has been made and discussed with the users to find out what exactly they expect the user interface to be. Some of the questions which are raised during the discussion are.

I. Layout

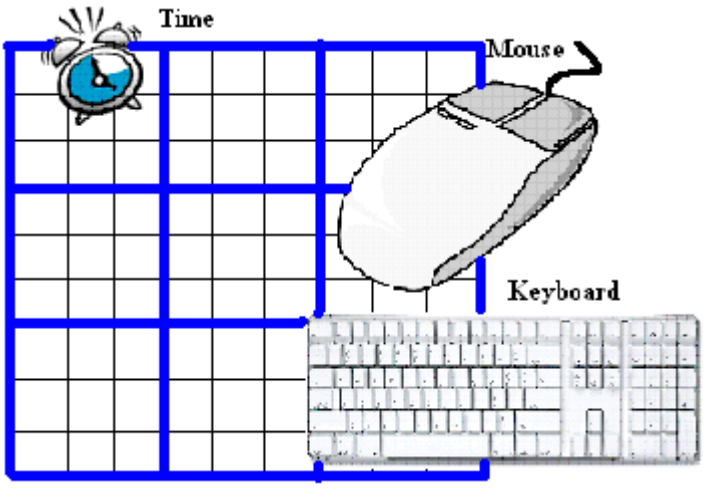
II. Interaction either from mouse or keyboard

III. Type of menus

IV. Resizable window or not

V. Kind of help/support

VI. Timer



3.2 Logic Behind Sudoku:

The Generator is for creating qualified Sudoku arrays. Combining with the Solver, we could start to make our own Sudoku puzzles. The main technique in this algorithm is using permutations, based on one qualified pattern to enumerate qualified Sudoku arrays. Here, we only use Band permutations (3! times variations), Row permutations within a band (3! ^{3} times variations), Stack permutations (3! times variations), and Column permutations within a stack (3! ^{3} times variations) to get the arrays. Finally, put the qualified Sudoku array into a text file. (Ex: 3! =3*2*1= 6 ---123,132,213,231,312,321).

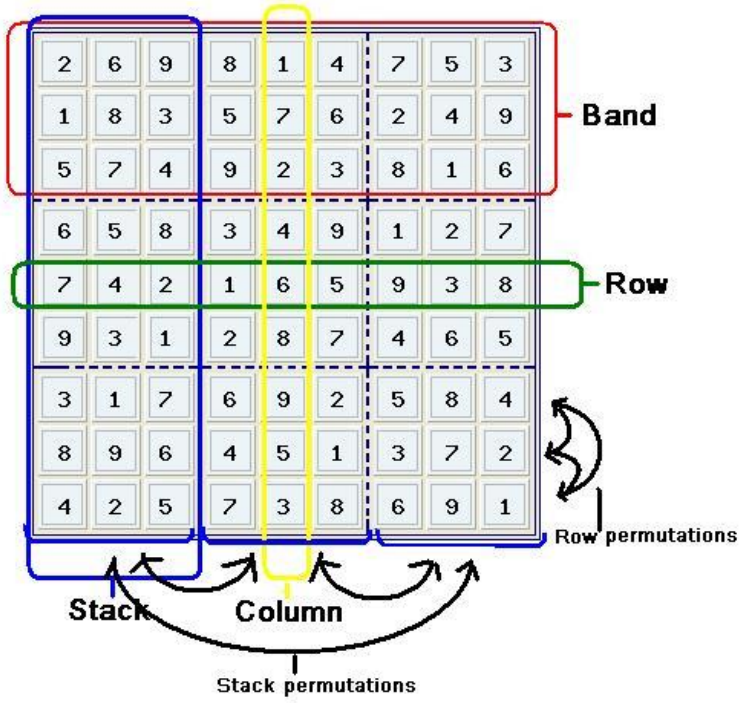


Figure: Different types of permutations

3.3 Sudoku Database:

This part of the system comprises the collection of puzzles obtained from different sources in a text file. The system also allows the user to store and retrieve their unfinished puzzles.

The block diagram of Sudoku game is shown in the figure.

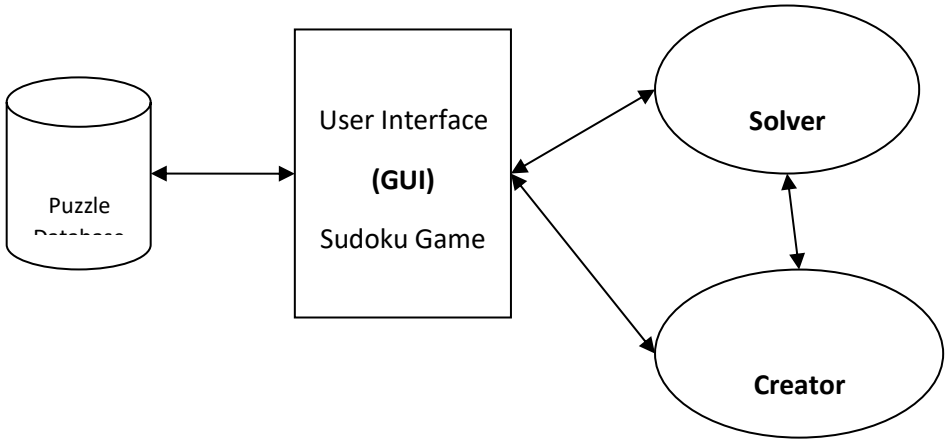


Figure: Block diagram of Sudoku game

CHAPTER 4

IMPLEMENTATION

4.1 TECHNIQUE

Language of Implementation

For this project I chose to use Java as the programming language. This was simply due to it being the language I had been using most recently. A Constraint Programming language like Prolog would probably have been better suited to this task since such languages include the ability to do some of the solving work automatically. However, I have no experience in using this programming paradigm.

Measurement Limitations

Test Grid 1 (28 hints, from The Age)

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

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

This grid can be solved using only the Singleton Domains/Unique Domains logic techniques.

Test Grid 2 (28 hints, from The Age)

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

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

This grid needs all 5 implemented logic techniques to be solved.

Test Grid 3 (26 hints, from The Australian)

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

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

This grid has a relatively low number of hints.

Generator Implementation

Remarks

The generator demonstrates that the number of hints given in a puzzle is not necessarily a good measure of how difficult a puzzle is for a human player is can be -the example Easy grid below has only 24 hints and yet requires no techniques other than the basic unique/singleton domain methods, while the example Hard grid has 26 hints and requires the other techniques to be used several times.

Generating a Grid

Generating the Filled Grid Creating a filled grid turned out to be an easy task simply running the brute force solver on an empty (0 squares filled) grid will produce a valid solved grid. However, in the original implementation the solver would produce the same grid every time (since it runs from left-to-right, top-to-bottom always). By instead taking a random path through the grid a random solved Sudoku grid may be produced.

First Square Removal Attempt

My first attempt to progressively remove squares from the filled grid did not fully succeed. My method was to remove a square, reset the domains of the row, column and 3×3 region intersecting that square and then check if the value of that square was still deducible by logic. This worked for hint-counts of about 35 and upwards, but never produced any grids with fewer hints. Furthermore, the grids produced were always ‘easy’ that is to say, they never required the more advanced logic techniques (pointing pairs, etc.) to be solved.

Second Square Removal Attempt

My second approach was to simply remove numbers and check with the brute force solver whether the grid still had a unique solution – if it did, another

number would be removed, if not, a different number would be removed. The process would repeat until the desired number of filled squares was reached.

This method worked reasonably well it produced grids with a relatively small number of hints (down to about 25). Unfortunately, the time taken to produce a grid was extremely variable anywhere from a few seconds to many minutes. The variation is caused by the random path that the decision as to which square to remove is entirely random. Thus, the generator might 'get lucky' and happen to successively pick hints that maintain a unique solution, quickly finding an empty puzzle, or the generator might do the reverse and consistently pick squares which cannot be emptied without increasing the solution count. Furthermore, the brute force solver was being run again and again, increasing the time taken to find a grid.

Third Square Removal Attempt

This time, instead of using the brute force solver to check whether the generated grid had a unique solution, the logic solver was used to check if the grid was solvable. Since the logic solver is nearly always faster than the brute force solver, this should decrease the time taken to find a grid. However, the variability remains of the generator still removes squares at random.

Different Grid Varieties

Difficulty Level

The generator provides 2 different difficulty levels: easy and hard. Easy grids are solvable by using only the unique and singleton domain techniques (see the Logic Solver section of this report), while hard grids require at least one use of the other techniques.

Symmetry

The Sudoku puzzles published in The Age and The Australian have their hints distributed in a symmetric pattern across the board for

aesthetic reasons. The generator provides the option of generating such grids. However, the number of possible empty grids that can be made from a filled grid is smaller if symmetry is required, so using the symmetrical option can slow down generation.

Easy Symmetric Grid (27 hints)

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

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

Hard Symmetric Grid (30 hints)

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

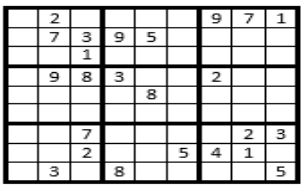
3	9	7	5	8	1	4	6	2
5	8	2	4	9	6	1	7	3
4	1	6	3	7	2	9	5	8
1	7	8	2	6	9	5	3	4
2	5	9	8	4	3	7	1	6
6	3	4	7	1	5	2	8	9
7	4	1	9	3	8	6	2	5
9	2	3	6	5	7	8	4	1
8	6	5	1	2	4	3	9	7

Measurements

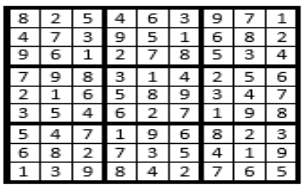
As mentioned above, even in its final implementation the generator is extremely variable as to how long it takes to generate a grid with a given number of hints as it clears squares in a random order. As such a simple average is not an appropriate measure. Instead, I ran the generator 10 times for various grid types and recorded the times.

Example Generated Grids

Easy Grid (24 hints)

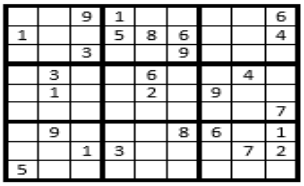


	2					9	7	1
	7	3	9	5				
		1						
	9	8	3			2		
				8				
		7					2	3
		2			5	4	1	
	3		8					5

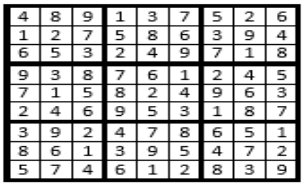


8	2	5	4	6	3	9	7	1
4	7	3	9	5	1	6	8	2
9	6	1	2	7	8	5	3	4
7	9	8	3	1	4	2	5	6
2	1	6	5	8	9	3	4	7
3	5	4	6	2	7	1	9	8
5	4	7	1	9	6	8	2	3
6	8	2	7	3	5	4	1	9
1	3	9	8	4	2	7	6	5

Hard Grid (26 hints)



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



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

Easy Grid (24 hints)

Attempt	Time Taken
1	168ms
2	139ms
3	1445ms
4	142ms
5	146ms
6	127ms
7	2260ms
8	23741ms
9	107ms
10	114ms

Although the numbers generally hovered around the 140ms mark, attempts 3, 7 and particularly 8 demonstrate the extreme variability of the generator.

Hard Grid (28 hints)

Attempt	Time Taken
1	6878ms
2	11422ms
3	25651ms
4	16665ms
5	10757ms
6	17517ms
7	26599ms
8	12683ms
9	24456ms
10	43491ms

The time taken to generate a hard grid is on average very high compared to generating an easy grid, but the time is still quite variable.

4.2 DIAGRAMS

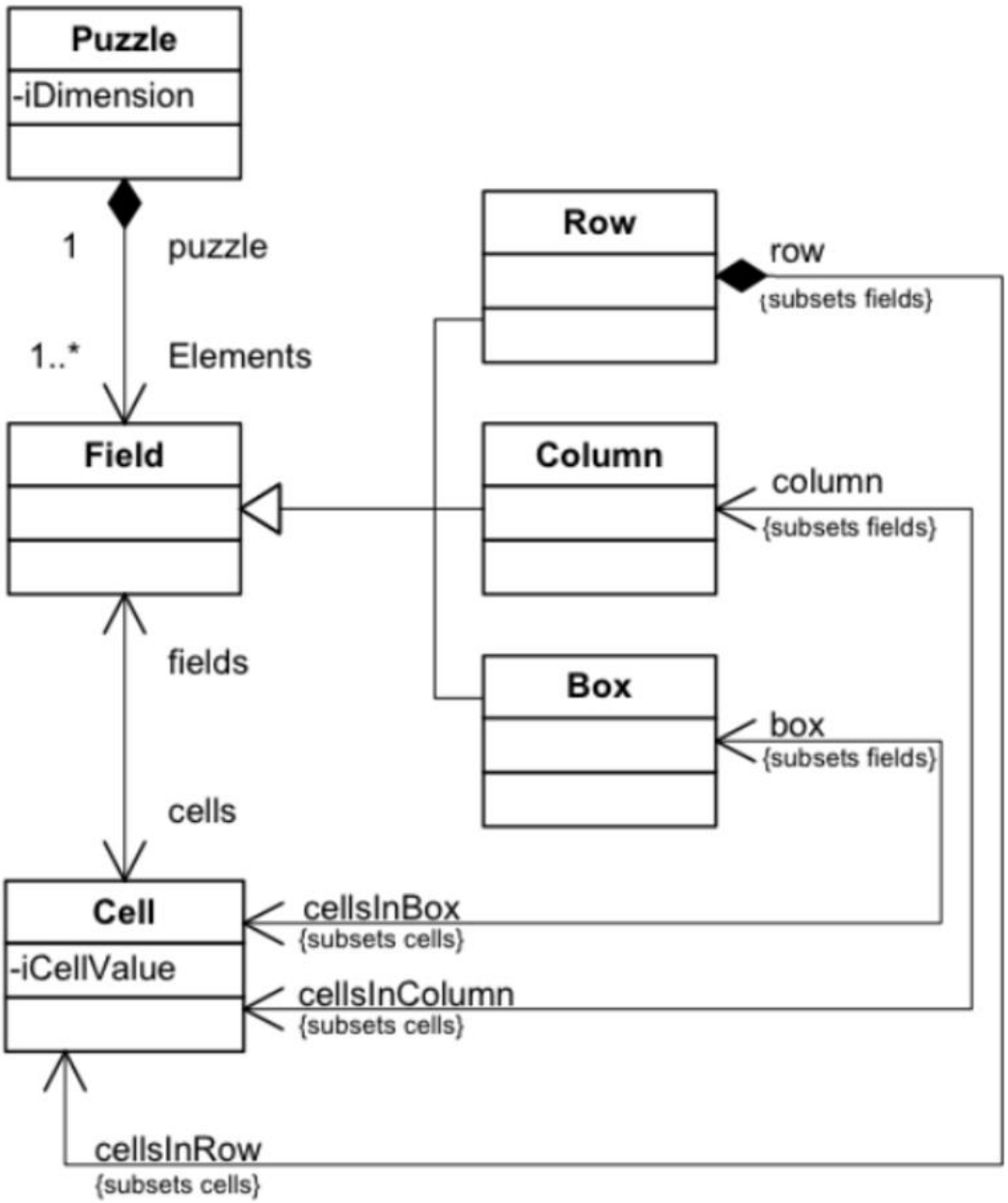


Fig 4.2.1 Structure meta-model for Sudoku

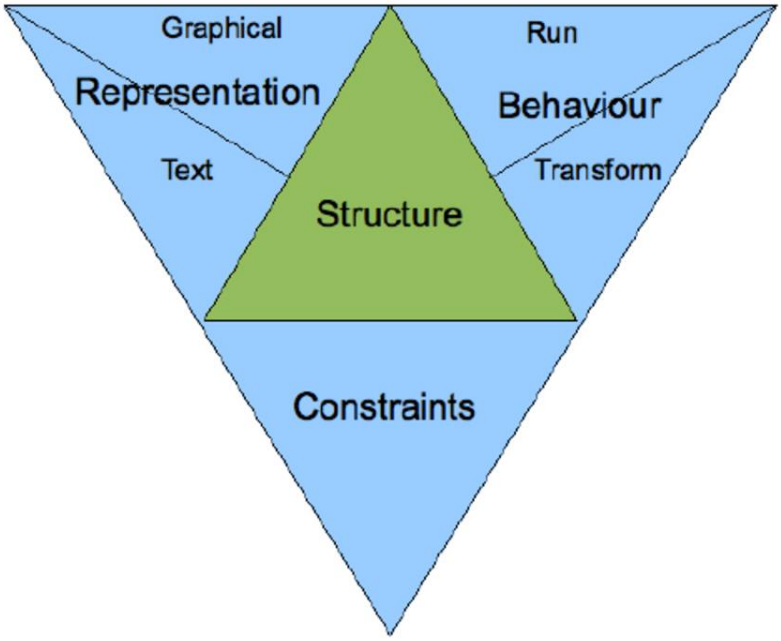


Fig 4.2.2 Aspects of a computer language description

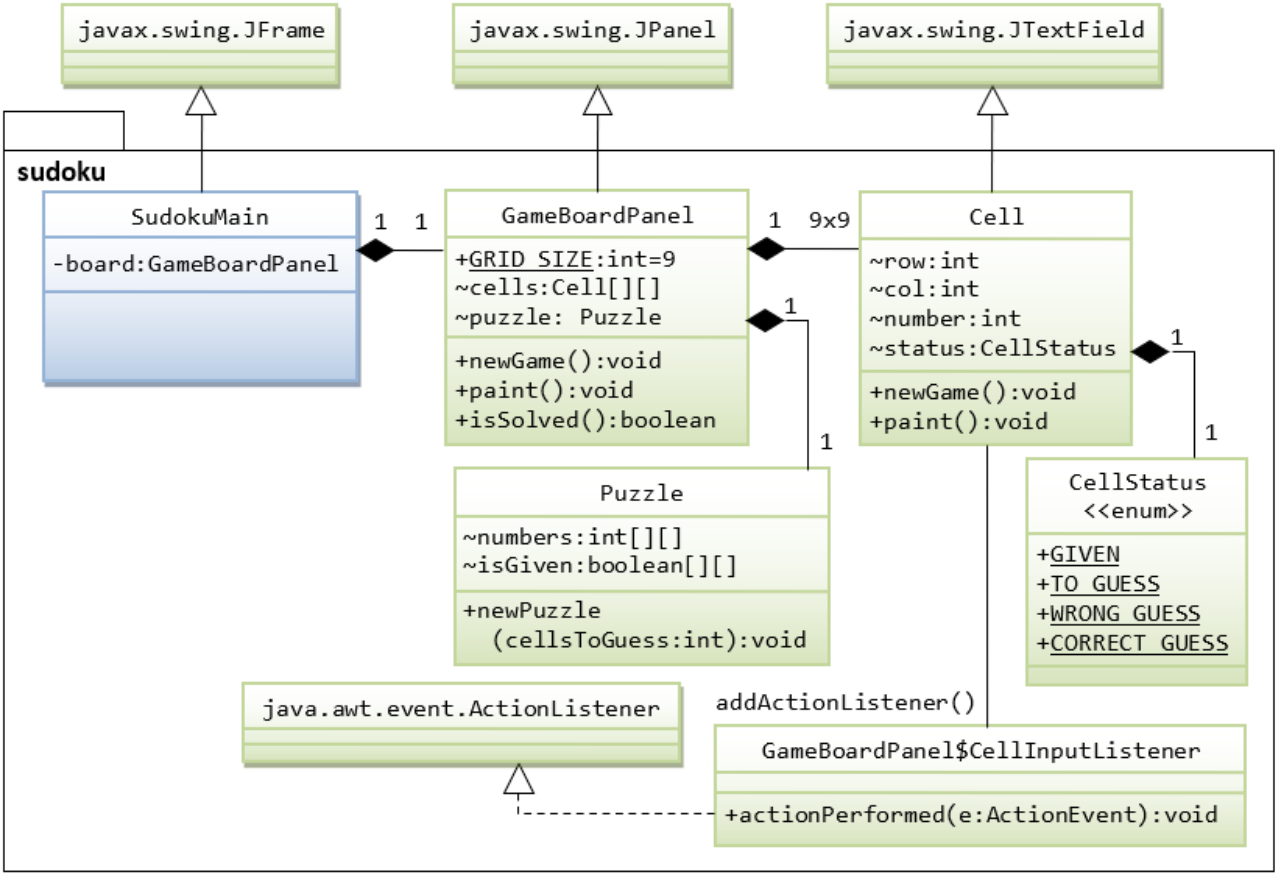


Fig 4.2.3 Class diagram

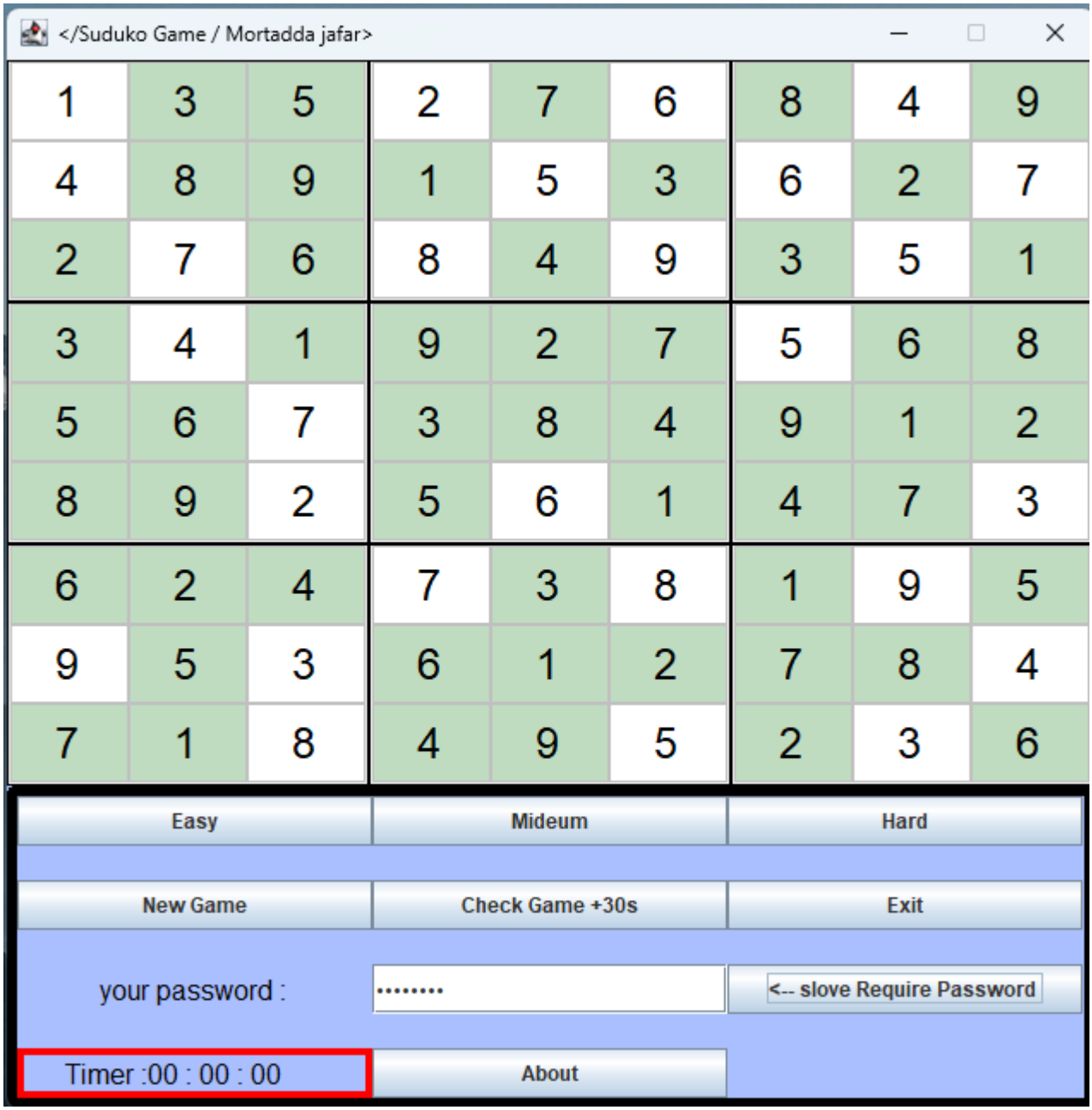


Fig 4.3.4 Eclipse: Graphical editor created in GMF shows the editor created with the GMF.

4.3 PROGRAM CODE

Panel.java

```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Font;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.JTextField;
import javax.swing.Timer;

public class Panal extends javax.swing.JPanel {

    Sudoku game;
    private Timer timer;
    private JButton nbtn = new JButton("new game");
    private static JTextField[][] boxes;
```

```
    private JPasswordField pass = new JPasswordField("mortadda");
    private JLabel label = new JLabel("      Timer :00 : 00 : 00");
    private JLabel passsLabel = new JLabel("           your password :");
    private JPanel[][] paneles;
    private JPanel center, bPanel, levelPanel;
    private JButton nBtn, cBtn, eBtn, hardBtn, midBtn, easyBtn, slove, about;
    private int[][] temp = new int[9][9];
    private int[][] grid = new int[9][9];
    private int counter = 0;

    public JTextField newtextfield() {
        JTextField j = new JTextField("");
        j.setBorder(BorderFactory.createLineBorder(Color.lightGray));
        j.setFont(new Font(Font.DIALOG, Font.PLAIN, 25));
        j.setHorizontalAlignment(JTextField.CENTER);
        /*------------------mouse lisner----------------*/
        j.addMouseListener(new MouseAdapter() {

            @Override
            public void mouseEntered(MouseEvent e) {
                if (j.isEditable()) {
                    ((JTextField)
e.getSource()).setBorder(BorderFactory.createLineBorder(Color.decode("#f6ea80")
));
                    ((JTextField) e.getSource()).setBackground(Color.decode("#f6ea80"));
                }
            }

            @Override
```

```
        public void mouseExited(MouseEvent e) {
            if (j.isEditable()) {
                ((JTextField)
e.getSource()).setBorder(BorderFactory.createLineBorder(Color.lightGray));
                ((JTextField) e.getSource()).setBackground(Color.white);
            }
        }
    });
    /*-------------------------------------------*/

    j.addKeyListener(new KeyListener() {

        @Override
        public void keyTyped(KeyEvent e) {
        }

        @Override
        public void keyPressed(KeyEvent e) {
        }

        @Override
        public void keyReleased(KeyEvent e) {
            if (j.isEditable()) {
                ((JTextField) e.getSource()).setForeground(Color.decode("#0c4"));
            } else {
                ((JTextField) e.getSource()).setForeground(Color.black);
            }
        }
    });
```

```
    return j;
}

public Panal() {
    initComponents();
    /*-----------------------main panal  ----------------------------------*/
    center = new JPanel();  //main panel
    center.setLayout(new GridLayout(3, 3));     //grid for 3*3
    center.setBackground(Color.BLACK);
    setLayout(new BorderLayout());
    add(center);  //add main panel to frame

    boxes = new JTextField[9][9];
    paneles = new JPanel[3][3];
    passsLabel.setFont(new Font(Font.DIALOG, Font.PLAIN, 16));
    passsLabel.setForeground(Color.black);
    label.setForeground(Color.black);
    label.setBorder(BorderFactory.createLineBorder(Color.red, 4));
    label.setFont(new Font(Font.DIALOG, Font.PLAIN, 16));

    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            paneles[i][j] = new JPanel();
            paneles[i][j].setBorder(BorderFactory.createLineBorder(Color.black));
            paneles[i][j].setLayout(new GridLayout(3, 3));
            center.add(paneles[i][j]);
        }
    }
    /*-----------------------text fildes in boxes----------------------------------*/
```

```
        for (int n = 0; n < 9; n++) {
            for (int i = 0; i < 9; i++) {
                boxes[n][i] = newtextfield();
                int fm = (n + 1) / 3;
                if ((n + 1) % 3 > 0) {
                    fm++;
                }
                int cm = (i + 1) / 3;
                if ((i + 1) % 3 > 0) {
                    cm++;
                }
                paneles[fm - 1][cm - 1].add(boxes[n][i]);   //add box to panel
            }
        }
        /*-----------------------panal for buttons ------------------------------------*/

        bPanel = new JPanel();
        bPanel.setBackground(Color.decode("#AABFFF"));
        bPanel.setBorder(BorderFactory.createLineBorder(Color.black, 6, true));
        bPanel.setLayout(new GridLayout(4, 3, 0, 20));

        /*-----------------------panal for new game button ----------------------------------
-*/
        ActionListener action = new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent event) {
                label.setText(TimeFormat(counter));
                counter++;
```

```
        }
    };

    /*-----------------------panal for new game button ----------------------------------
-*/
    nBtn = new JButton("New Game");
    nbtn.setSize(20, 50);
    timer = new Timer(1000, action);
    nBtn.addActionListener(new ActionListener() {

        @Override
        public void actionPerformed(ActionEvent e) {
            counter = 0;
            timer.start();
            restgame();
            Sudoku.newGame();

        }
    });

    /*-----------------------panal for check game button --------------------------------
---*/
    cBtn = new JButton("Check Game +30s");

    cBtn.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            for (int i = 0; i < 9; i++) {
```

```
                for (int j = 0; j < 9; j++) {
                    if (!boxes[i][j].isEditable()) {
                        continue;
                    } else if (boxes[i][j].getText().equals(String.valueOf(grid[i][j]))) {
                        boxes[i][j].setBackground(Color.decode("#C0DCD9"));
                    } else if (boxes[i][j].getText().isEmpty()) {
                        boxes[i][j].setBackground(Color.WHITE);
                        continue;
                    } else {
                        boxes[i][j].setBackground(Color.red);
                    }
                }
            }
             counter += 30;
        }
    });
    /*-----------------------panal for new Exit button ----------------------------------
*/
    eBtn = new JButton("Exit");

    eBtn.addActionListener(new ActionListener() {

        @Override
        public void actionPerformed(ActionEvent e) {
            System.exit(0);
        }
    });
```

```
        /*-----------------------panal for new Hard button ----------------------------------
*/
        easyBtn = new JButton("Hard");

        easyBtn.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                restgame();
                counter = 0;
                timer.start();
                Sudoku.setlevel(4);
                Sudoku.newGame();
            }
        });
        /*-----------------------panal for new Hard button ----------------------------------
*/
        midBtn = new JButton("Mideum");

        midBtn.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                restgame();
                counter = 0;
                timer.start();
                Sudoku.setlevel(3);
                Sudoku.newGame();

            }
        });
```

```
        /*-----------------------panal for new Hard button ----------------------------------
*/
        hardBtn = new JButton("Easy");

        hardBtn.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                restgame();
                counter = 0;
                timer.start();
                Sudoku.setlevel(2);
                Sudoku.newGame();
            }
        });
        /*-----------------------panal for new Hard button ----------------------------------
*/
        slove = new JButton("<-- slove Require Password ");

        slove.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if (pass.getText().equals("mortadda")) {
                    timer.stop();
                    counter = 0;
                    label.setText(TimeFormat(counter));
                    for (int i = 0; i < 9; i++) {
                        for (int j = 0; j < 9; j++) {
                            boxes[i][j].setText(String.valueOf(grid[i][j]));
                        }
```

```
                }
            } else {
                JOptionPane.showMessageDialog(center, "your password incorrect");
            }
        }
    });
    /*-----------------------panal for new about button ----------------------------------
-*/
    about = new JButton("About");

    about.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            JOptionPane.showMessageDialog(center, "By vamshi gadde");
        }
    });
    /*-----------------------panal for new about button ----------------------------------
-*/
     pass.addMouseListener(new MouseAdapter() {

        @Override
        public void mouseClicked(MouseEvent e) {
              ((JPasswordField) e.getSource()).setText("");
        }
    });

    /*-----------------------add button panal and butons to frame and panel -----------
-------------------------*/
    bPanel.add(hardBtn);   //add new game button to
    bPanel.add(midBtn);
```

```
    bPanel.add(easyBtn);
    bPanel.add(nBtn);   //add new game button to
    bPanel.add(cBtn);
    bPanel.add(eBtn);
    bPanel.add(passsLabel);
    bPanel.add(pass);
    bPanel.add(slove);
    bPanel.add(label);
    bPanel.add(about);

    add(bPanel, "South");   //add button panel to frame

}

public void setarray(int[][] grid, int[][] temp) {
    for (int i = 0; i < 9; i++) {
        for (int j = 0; j < 9; j++) {
            this.temp[i][j] = temp[i][j];
            this.grid[i][j] = grid[i][j];
        }
    }
}

public void setTextLable() {
    for (int i = 0; i < 9; i++) {
        for (int j = 0; j < 9; j++) {
            if (this.temp[i][j] != 0) {
                boxes[i][j].setText(String.valueOf(this.temp[i][j]));
                boxes[i][j].setEditable(false);
```

```
                boxes[i][j].setBackground(Color.decode("#C0DCC0"));
            } else {
                boxes[i][j].setText("");
            }
        }
    }
}

public static void restgame() {
    for (int i = 0; i < 9; i++) {
        for (int j = 0; j < 9; j++) {
            boxes[i][j].setForeground(Color.black);
            boxes[i][j].setEditable(true);
            boxes[i][j].setBackground(Color.WHITE);
        }
    }
}

private String TimeFormat(int count) {

    int hours = count / 3600;
    int minutes = (count - hours * 3600) / 60;
    int seconds = count - minutes * 60;

    return String.format("      Timer :" + "%02d", hours) + " : " +
String.format("%02d", minutes) + " : " + String.format("%02d", seconds);
}
```

```
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-
BEGIN:initComponents
    private void initComponents() {


        setLayout(null);
    }// </editor-fold>//GEN-END:initComponents



    // Variables declaration - do not modify//GEN-BEGIN:variables
    // End of variables declaration//GEN-END:variables
}
```

Sudoku.java

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Random;
import javax.swing.JFrame;

public class Sudoku {


    static JFrame frame;
    static Panal p;
    private  static int[][] grid;
    private static int[][] temp;
    private static Random ran = new Random();
    private static int level = 2;
```

```
public static void main(String[] args) {

    grid = new int[9][9];
    temp = new int[9][9];
    frame = new JFrame();
    frame.setResizable(false);
    frame.setLocation(320, 40);
    frame.setSize(650, 650);
    frame.setTitle("</Suduko Game / Mortadda jafar>");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    p = new Panal();
    frame.setContentPane(p);
    frame.setVisible(true);
}

public static void newGame() {
    int k = 0;
    ArrayList<Integer> randomnumber = getRandomNum();

    for (int i = 0; i < 9; i++) {
        for (int j = 0; j < 9; j++) {
            grid[i][j] = 0;
            if (((j + 2) % 2) == 0 && ((i + 2) % 2) == 0) {
                grid[i][j] = randomnumber.get(k);
                k++;
                if (k == 9) {
                    k = 0;
                }
            }
```

```
            }
        }

        if (search(grid)) {
            System.out.println("OK !!");
        }
        int rann = ran.nextInt(level);
        int c = 0;
        for (int i = 0; i < 9; i++) {
            for (int j = 0; j < 9; j++) {
                temp[i][j] = 0;
                if (c < rann) {
                    c++;
                    continue;
                } else {
                    rann = ran.nextInt(level);
                    c = 0;
                    temp[i][j] = grid[i][j];
                }
            }
        }

//        for (int i = 0; i < grid.length; i++) {
//            for (int j = 0; j < grid[i].length; j++) {
//                System.err.print(grid[i][j]+" ");
//            }
//            System.out.println("sssssssssssssssssssssss");
//        }
        p.setarray(grid, temp);
```

```
    p.setTextLable();
}

public static int[][] getFreeCellList(int[][] grid) {

    int numberOfFreeCells = 0;
    for (int i = 0; i < 9; i++) {
        for (int j = 0; j < 9; j++) {
            if (grid[i][j] == 0) {
                numberOfFreeCells++;
            }
        }
    }

    int[][] freeCellList = new int[numberOfFreeCells][2];
    int count = 0;
    for (int i = 0; i < 9; i++) {
        for (int j = 0; j < 9; j++) {
            if (grid[i][j] == 0) {
                freeCellList[count][0] = i;
                freeCellList[count][1] = j;
                count++;
            }
        }
    }

    return freeCellList;
}
```

```
public static boolean search(int[][] grid) {
    int[][] freeCellList = getFreeCellList(grid);
    int k = 0;
    boolean found = false;

    while (!found) {
        //get free element one by one
        int i = freeCellList[k][0];
        int j = freeCellList[k][1];
        // if element equal 0 give 1 to first test
        if (grid[i][j] == 0) {
            grid[i][j] = 1;
        }
        // now check 1 if is avaible
        if (isAvaible(i, j, grid)) {
            //if free is equal k ==> board sloved
            if (k + 1 == freeCellList.length) {
                found = true;
            } else {
                k++;
            }
        }
        //increase element  by 1
        else if (grid[i][j] < 9) {
            grid[i][j] = grid[i][j] + 1;
        }
        //now if element value eqaule 9 backtrack to later element
        else {
            while (grid[i][j] == 9) {
```

```
                grid[i][j] = 0;
                if (k == 0) {
                    return false;
                }
                k--; //backtrack to later element
                i = freeCellList[k][0];
                j = freeCellList[k][1];
            }
            grid[i][j] = grid[i][j] + 1;
        }
    }

    return true;
}

public static boolean isAvaible(int i, int j, int[][] grid) {

    // Check   row
    for (int column = 0; column < 9; column++) {
        if (column != j && grid[i][column] == grid[i][j]) {
            return false;
        }
    }

    // Check  column
    for (int row = 0; row < 9; row++) {
        if (row != i && grid[row][j] == grid[i][j]) {
            return false;
        }
```

```
        }

        // Check box
        for (int row = (i / 3) * 3; row < (i / 3) * 3 + 3; row++) {//      i=5 ,j=2   || row =3 col=0   ||i=3  j=0
            for (int col = (j / 3) * 3; col < (j / 3) * 3 + 3; col++) {
                if (row != i && col != j && grid[row][col] == grid[i][j]) {
                    return false;
                }
            }
        }

        return true; //else return true
    }

    public static ArrayList<Integer> getRandomNum() {
        ArrayList<Integer> numbers = new ArrayList<Integer>();
        for (Integer i = 1; i < 10; i++) {
            numbers.add(i);
        }
        Collections.shuffle(numbers);
        return numbers;
    }

    public static void setlevel(int lev) {
        level = lev;
    }
}
```

CHAPTER 5
OUTPUT IMPLEMENTATION

The snapshot of output of project:

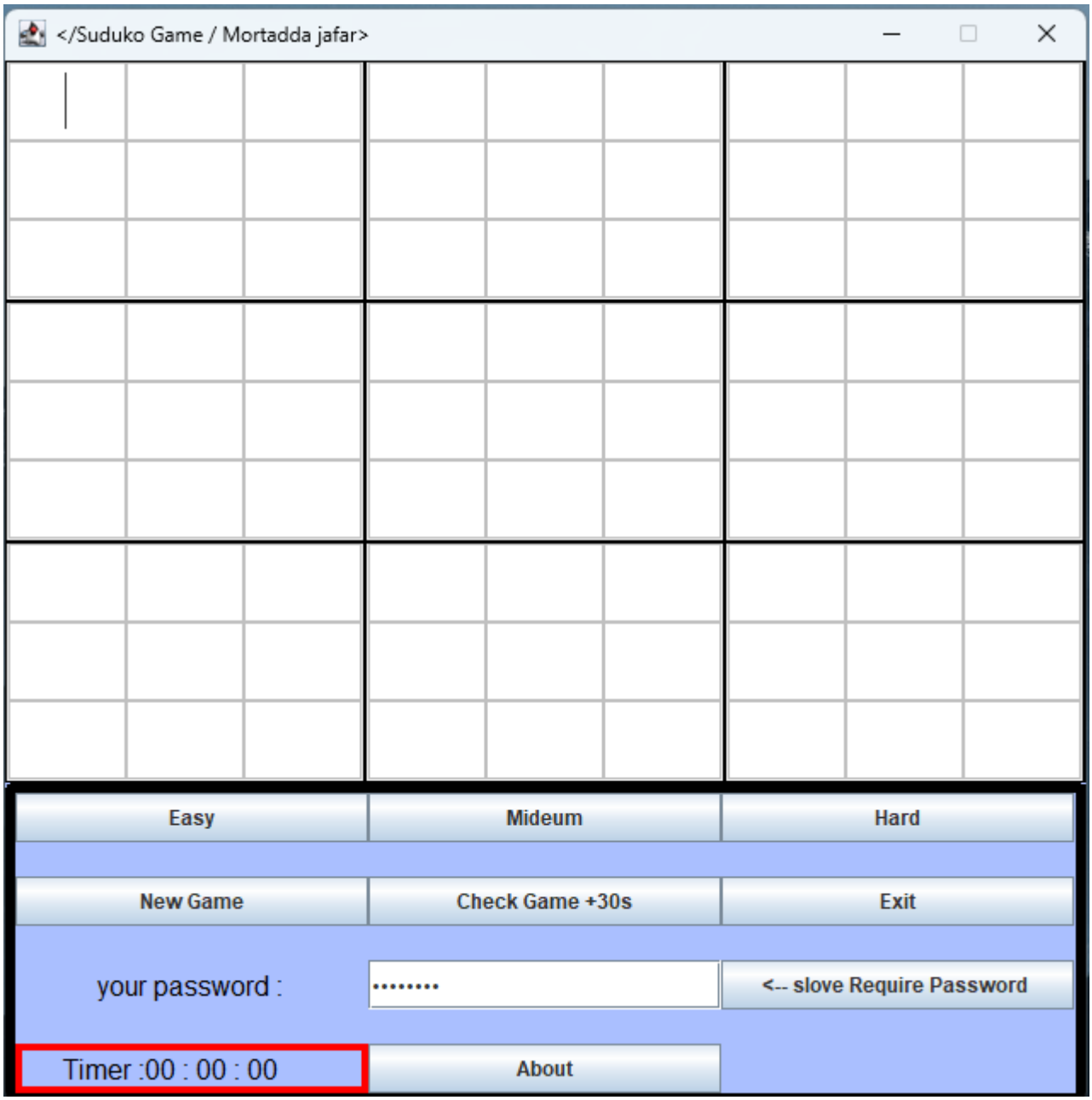


Fig 5.1 template of sudoku game.

Fig 5.2 Easy challenge mode

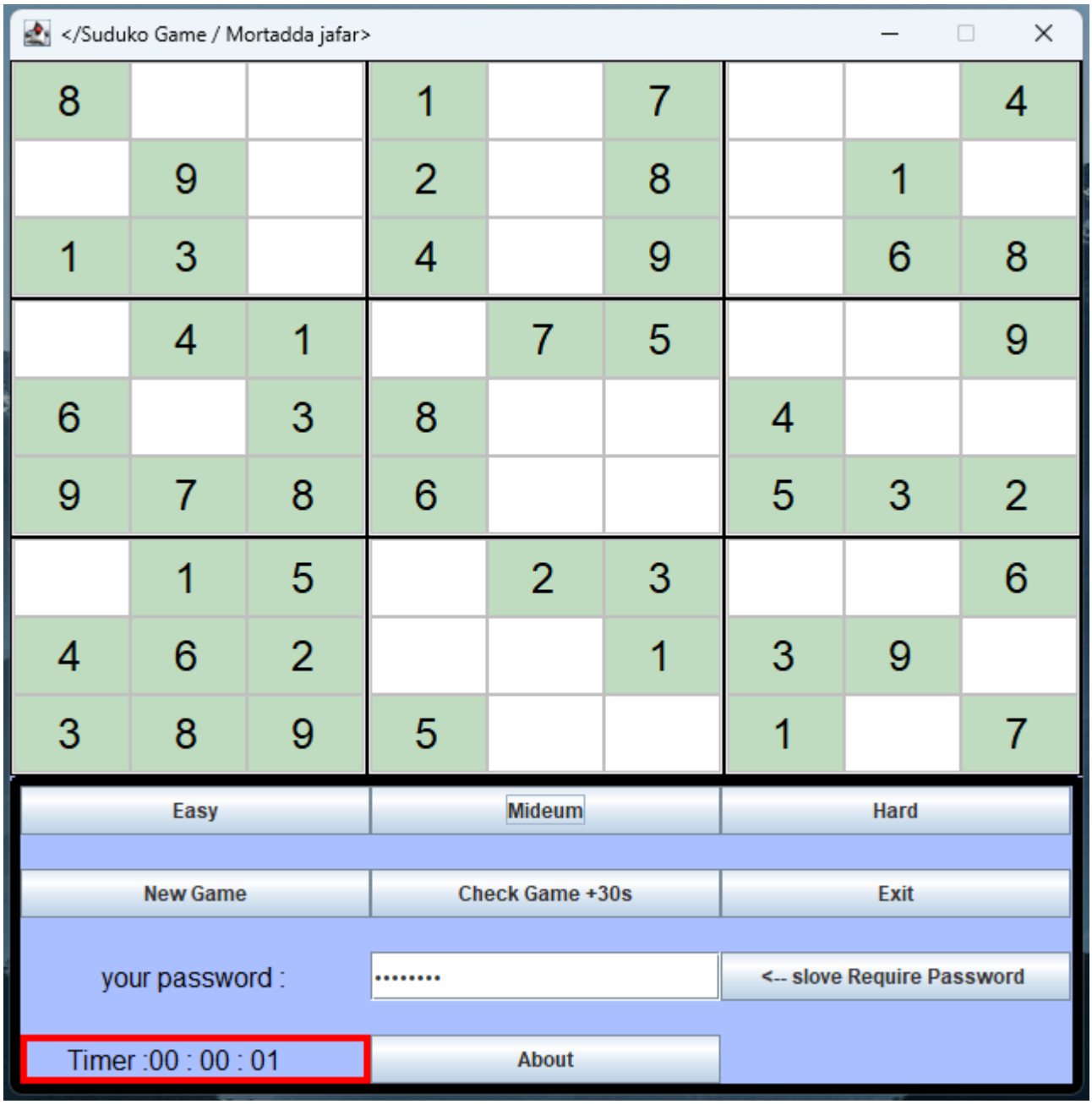


Fig 5.3 Medium challenge mode

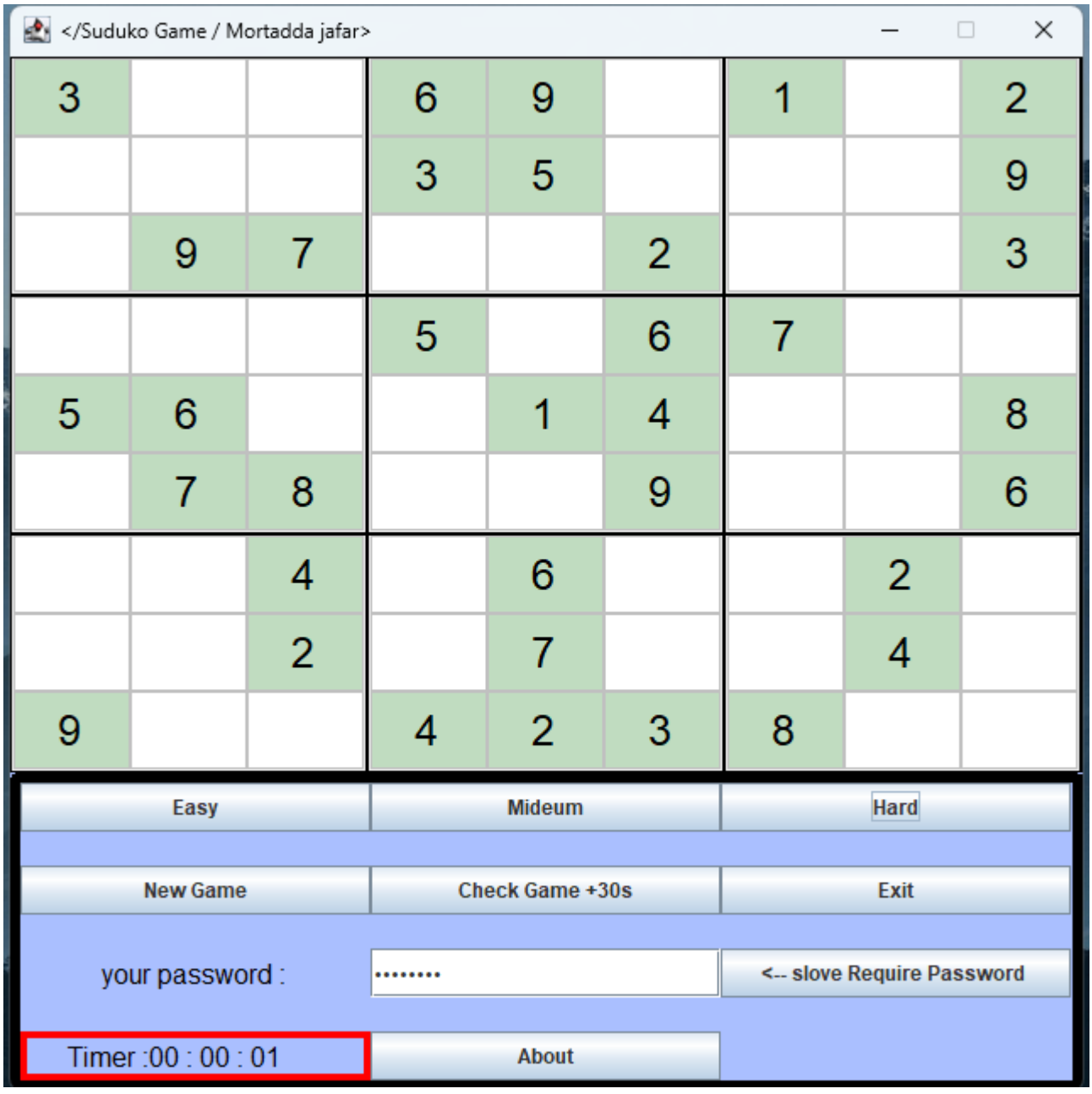


Fig 5.4 Hard challenge mode

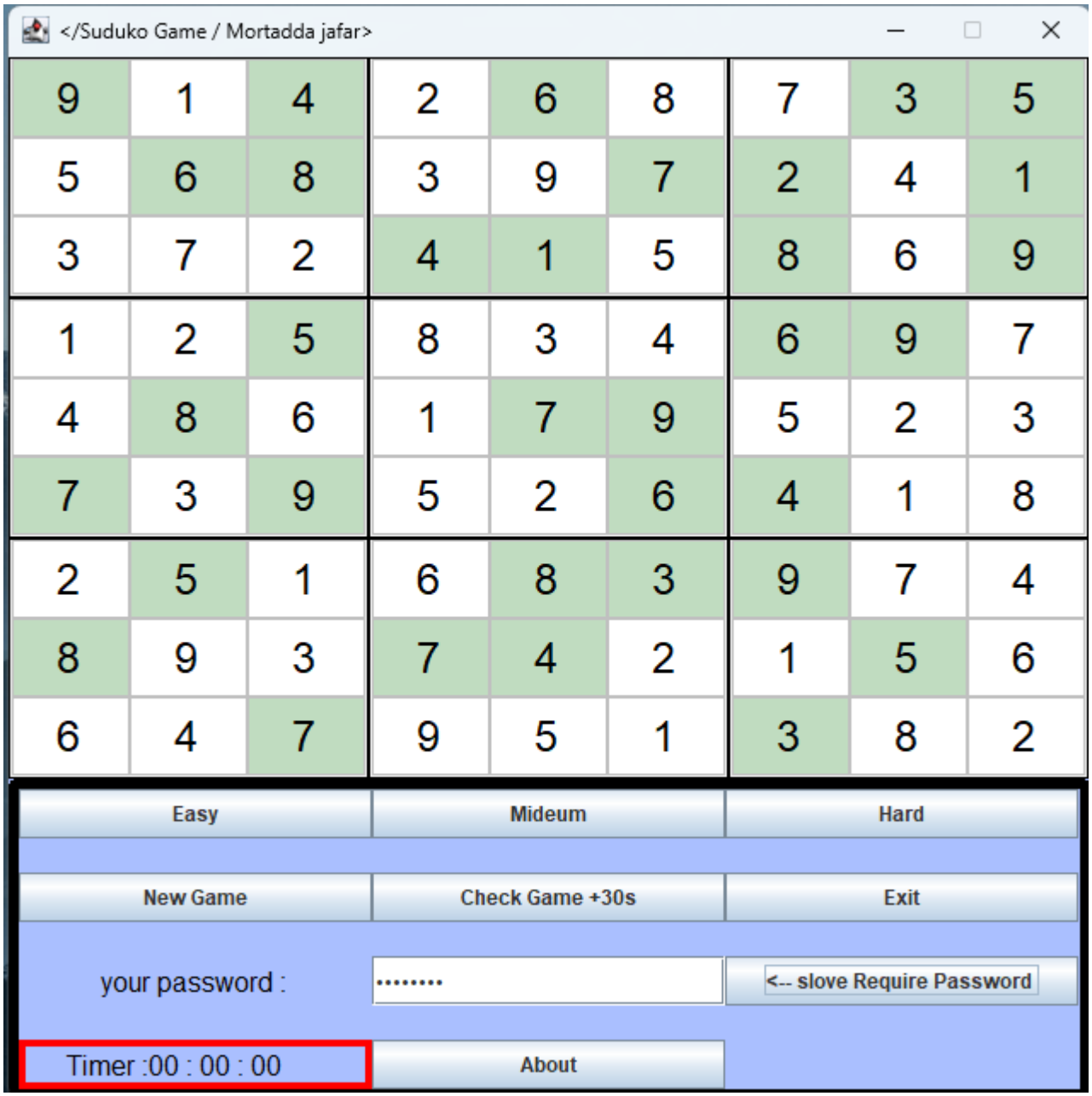


Fig 5.5 Solved sudoku using password!

CHAPTER 6
CONCLUSION

6.1 Achievements

Through generating this Sudoku solver and generator I feel I have improved my programming ability. This was perhaps the largest program in terms of time invested and lines of code written that I have created. The code is not of the highest quality, and it is severely lacking in documentation, but some of the problems posed by the project were a good challenge to solve. Writing the solver has demonstrated the advantages of ‘smart’ algorithms over naïve algorithms evidenced in the measurements above. Finally, I have experienced participating in what could be called a small research project (useful for future work).

6.2 Project Status

The logic solver portion of the program is sufficient for solving many Sudoku puzzles. However, the implementation could likely be improved to execute faster. Furthermore, there are many logic solving techniques that have not been implemented.

The generator, on the other hand, is currently rather poorly implemented. It works In that it successfully generates grids of a given number of hints, but the variability of the time taken to generate is a significant weakness. Additionally, the generator currently only offers two levels of puzzle difficulty (‘easy’ and ‘hard’) a scale of difficulties would be perhaps more useful.

CHAPTER 7

FUTURE ENHANCEMENTS

These are some of the main suggestions given by the user for improvement.

I. Instant help by giving hints and users progress while playing.

II. Score system based on time and accuracy, and database to keep track of top ten record.

CHAPTER 8

REFERENCES

1. https://www.youtube.com/watch?v=Wb76EMal9no
2. https://www.slideshare.net/TARUNKUMAR362/project-report-on-sudoku?from_action=save
3. https://www.researchgate.net/publication/299458217_Technical_Report_For_Simple_Sudoku
4. https://www.cs.upc.edu/~atort/documents/Sudoku.pdf
5. https://en.wikipedia.org/wiki/Sudoku