

# SUDOKU CHALLENGER

A Real-Time / Field-Based Research Project (22AM284) report submitted to the  
Jawaharlal Nehru Technological University, Hyderabad

Submitted by

<b>T ROHAN REDDY</b>	<b>22B81A6697</b>
<b>K SAI SHYAM SUNDAR</b>	<b>22B81A66A8</b>
<b>D SAI SRUJAN</b>	<b>22B81A66A9</b>

Under the guidance of  
**Mr MOGHAL IRFAN PASHA**  
Assistant Professor



DEPARTMENT OF CSE (ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

## **CVR COLLEGE OF ENGINEERING**

(An Autonomous Institution, NAAC Accredited and Affiliated to JNTUH, Hyderabad)

Vastunagar, Mangalpalli(V), Ibrahimpatnam(M),  
Rangareddy (D), Telangana- 501 510

**JUNE 2024**

# **CVR COLLEGE OF ENGINEERING**

**(An Autonomous institution , NAAC Accredited and Affiliated to JNTUH, Hyderabad)**

Vastunagar,Mangalpalli(V),Ibrahimpattanam(M), Rangareddy  
(D), Telangana- 501 510

## **DEPARTMENT OF CSE (ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)**



### **CERTIFICATE**

This is to certify that the Real time/ Field-Based research project(22AM284) report entitled “**SUDOKU CHALLENGER**” is a record of work carried out by **T ROHAN REDDY, K SAI SHYAM SUNDAR, D SAI SRUJAN** submitted to Department of **CSE (ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)** , CVR College of Engineering, affiliated to Jawaharlal Nehru Technological University, Hyderabad during the year 2023-2024.

**Project Guide**

**Department Mr Moghal Irfan Pasha**  
**Lakshmi H N**

Assistant Professor  
CSE(AI&ML)  
CSE(AI&ML,DS,CS)

**Project Coordinator**

**Dr Surya Bhupal Rao**

Associate Professor  
CSE(AI&ML)

**Head of the**

**Dr**

Professor&HOD

## DECLARATION

We hereby declare that the Real time/ Field-Based research project (22AM284) report entitled “**SUDOKU CHALLENGER**” is an original work done and submitted to **CSE (ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)** Department, CVR College of Engineering, affiliated to Jawaharlal Nehru Technological University, Hyderabad and it is a record of bonafide project work carried out by us under the guidance of **Mr. MOGHAL IRFAN PASHA**, Assistant Professor, **CSE(AI&ML)**

We further declare that the work reported in this project has not been submitted, either in part or in full, for the award of any other degree or diploma in this Institute or any other Institute or University.

Signature of the Student  
**T Rohan Reddy**  
**22B81A6697**

Signature of the Student  
**Sai Shyam Sundar**  
**22B81A66A8**

**K**

Signature of the Student  
**D Sai Srujan**  
**22B81A66A9**

## ACKNOWLEDGEMENT

We are thankful and fortunate enough to get constant encouragement, support, and guidance from all **Teaching staff of CSE (ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)** Department who helped us in successfully completing this project work.

We thank **Dr SURYA BHUPAL RAO**, Project Coordinator and **Ms. K. ANURADHA, Mrs. S. VINEELA KRISHNA**, Project Review Committee members for their valuable guidance and support which helped us to complete the project work successfully.

We respect and thank our internal guide, Mr. MOGHAL IRFAN PASHA, **Assistant Professor, CSE(AI&ML)** for giving us all the support and guidance, which made us complete the project duly.

We would like to express heartfelt thanks to **Dr. H. N. Lakshmi**, Professor & Head of the Department, for providing us an opportunity to do this project and extending support and guidance.

We thank our Vice-Principal **Prof. L. C. Siva Reddy** for providing excellent computing facilities and a disciplined atmosphere for doing our work.

We wish a deep sense of gratitude and heartfelt thanks to Dr. Rama Mohan Reddy, Principal and the **Management** for providing excellent lab facilities and tools. Finally, we thank all those guidance helpful to us in this regard.

## **ABSTRACT**

Basic aim of the project is to build a sudoku puzzle for the users and provide them with the right answers to the puzzle to verify whether it is appropriate or not.

Basic 3 principles in sudoku are :-

In all 9 submatrices 3x3 the elements should be 1-9 without repetition.

In all rows there should be elements between 1-9 without repetition.

In all columns there should be elements between 1-9 without repetition.

## List of Figures

Fig 2.1 .....	13
Fig 2.2 .....	14
Fig 2.3 .....	15
Fig 2.4 .....	18
Fig 3.1 .....	27
Fig 3.2 .....	28
Fig 3.3 .....	29
Fig 3.4 .....	32
Fig 3.5 .....	33
Fig 3.6 .....	32

## TABLE OF CONTENTS

<b>Chapter No.</b>	<b>Contents</b>	<b>Page No.</b>
	Certificate	ii
	Declaration	iii
	Acknowledgement	iv
	Abstract	v
	List of figures	vi
1	<b>Introduction</b>	9-12
	1.1 Problem Statement	10
	1.2 Project Objectives	11
	1.3 Software & Hardware specifications	12
	1.3.1 Software requirements	12
	1.3.2 Hardware requirements	12
2	<b>Design Methodology</b>	13-18
	2.1 System Architecture	13
	2.2 Data flow Diagram or Flowchart	14-15
	2.3 Technology Description	16-18
3	<b>Implementation &amp; Testing</b>	19-32
	3.1 Code snippets	19-27
	3.2 Test cases	27-32
	4.1 <b>Conclusion</b>	33
	4.2 <b>Bibliography</b>	34

## INTRODUCTION

Sudoku is a widely played number game. Sudoku's most basic and popular layout is a  $9 \times 9$  grid with numbers showing up in a few of the squares. The challenge is to fill in the remaining squares by precisely placing each of the digits 1 through 9 in each row, column, and set of nine  $3 \times 3$  subgrids. The quantity and placements of the original numbers define the level of difficulty in Sudoku, which is just a logic game. No arithmetic is involved.

Sudoku puzzles can be quite difficult for novice players. However, you can play more effectively after you comprehend the rules and strategy of the game. To aid you, each grid has a few numbers on it at the start of the game.

As a player, you improve greatly when you attempt to think things through and look for logic in every action. It will become easier for you to solve the puzzle as soon as you begin filling in the grid. You will start the game with fewer numbers as you move up the difficulty ladder.

Artificial Intelligence actively uses Sudoku algorithms to train bots for a variety of jobs. Developers may train bots to comprehend and adjust to human behaviour by using these grids and the reasoning behind them.

Sudoku is renowned for its ability to foster mental clarity, which improves your ability to think clearly and helps you solve difficulties in real life. When players begin playing Sudoku on a regular basis, they typically observe improvements in both their cognitive function and the amount of time it takes them to understand a task.



## 1.1 Problem Statement

The goal of this project is to design and implement an efficient Sudoku solver. Sudoku is a logic-based number-placement puzzle that consists of a 9x9 grid, subdivided into nine 3x3 subgrids. The objective is to fill the grid so that each column, each row, and each of the nine 3x3 subgrids contain all of the digits from 1 to 9. The solver should be able to handle any valid Sudoku puzzle input and provide a solution in a reasonable amount of time.

Develop a software application that can:

Input: Read a partially filled 9x9 Sudoku grid.

Validation: Verify the input grid is a valid Sudoku puzzle.

Solving: Implement an algorithm to solve the puzzle.

Output: Display the solved puzzle.

## 1.2 Project Objectives

### 1. **Difficulty level:**

- We provide three options for the users to choose the difficulty level as
  - 1.Easy
  - 2.Medium
  - 3.Hard

### 2. **Generate:**

- New sudoku puzzle will be generated, when generate option is selected corresponding to the difficulty that they have chosen before.

### 3. **Solving the puzzle:**

- Once the user has completed filling the boxes, they can check their solutions by selecting solve button.
- Wrong once will be highlighted.

### 4. **Backtracking algorithm:**

- Backtracking is a general algorithmic technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, and removing those solutions that fail to satisfy the constraints of the problem at any point in time (i.e., "backtracking" when a solution fails).

### 5. **Back to change difficulty level:**

- We can go back to change the difficulty level that we have chosen before.

### 8. **Optimize Performance:**

- Ensure the recommendation system operates quickly and efficiently, providing realtime suggestions without significant delays.

### 9. **Reset:**

- Allow users to go back the beginning.

## **1.3 Software & Hardware specifications**

### **1.3.1 SOFTWARE REQUIREMENTS**

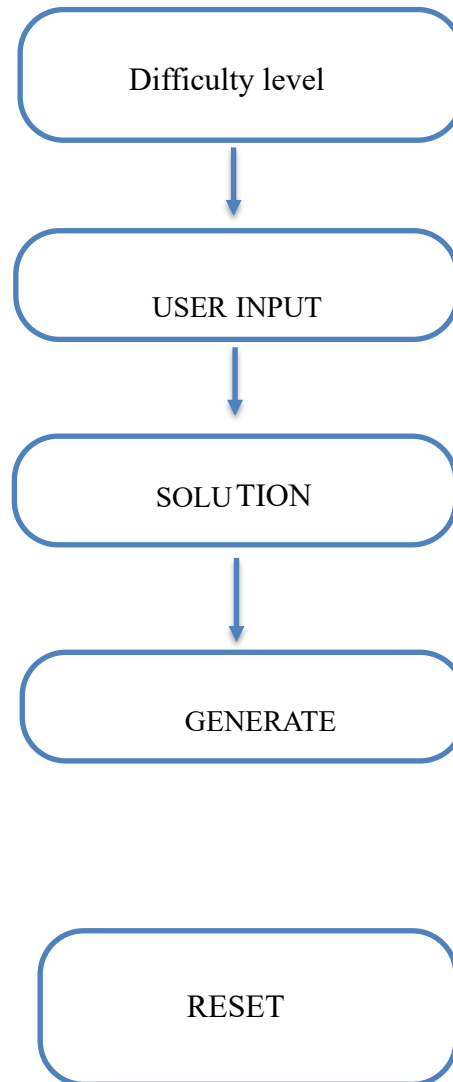
- Operating System: windows 10 or higher
- Programming Environment: java
- Database: N/A

### **1.3.2 HARDWARE REQUIREMENTS**

- Disk Storage: 128MB or higher
- Ram: 4GB or higher
- Processor: intel i3/AMD Ryzen 3 or higher

## DESIGN METHODOLOGY

### 2.1 System Architecture



**Fig 2.1 System Architecture**

## 2.2 Data Flow Diagram or FlowChart

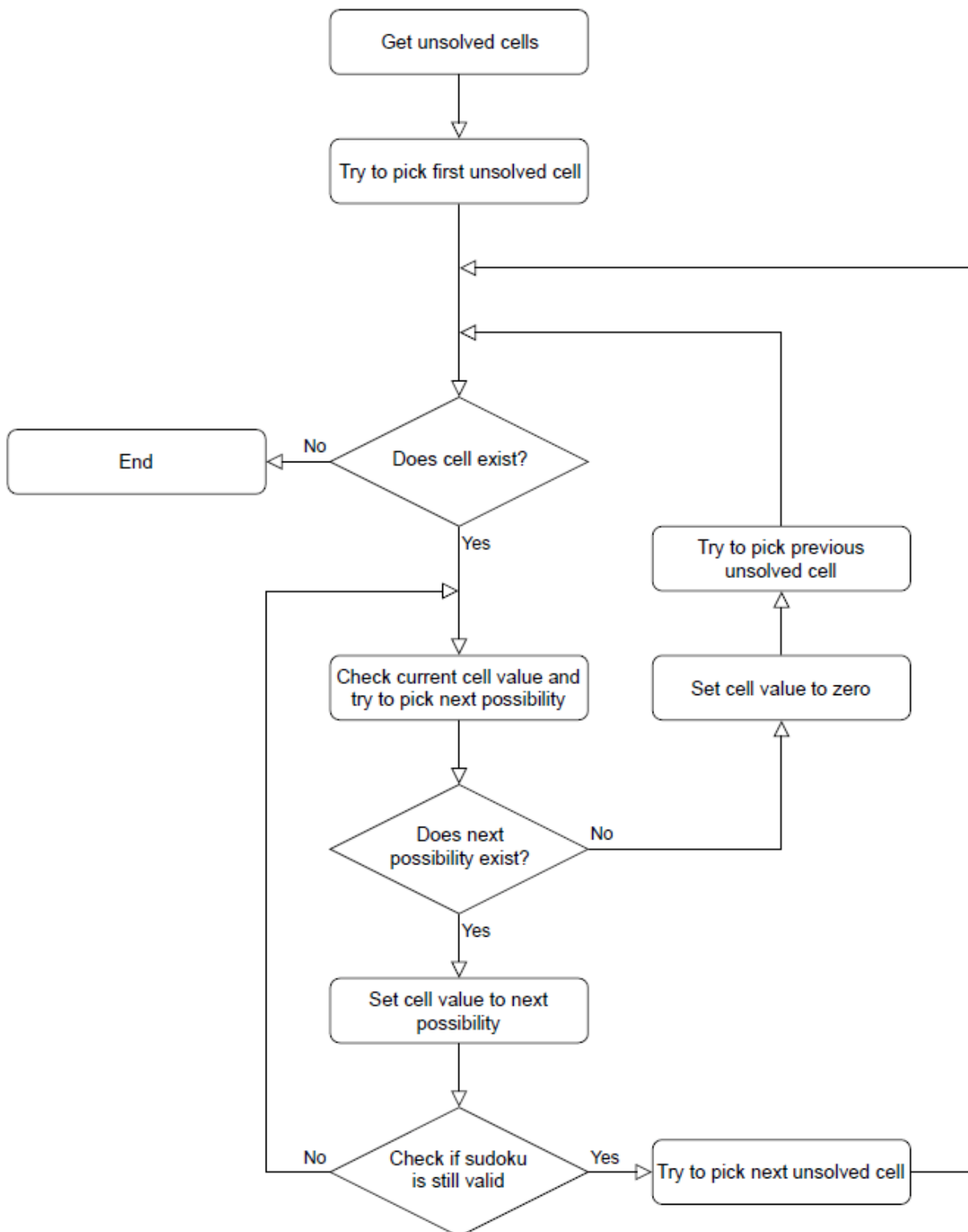
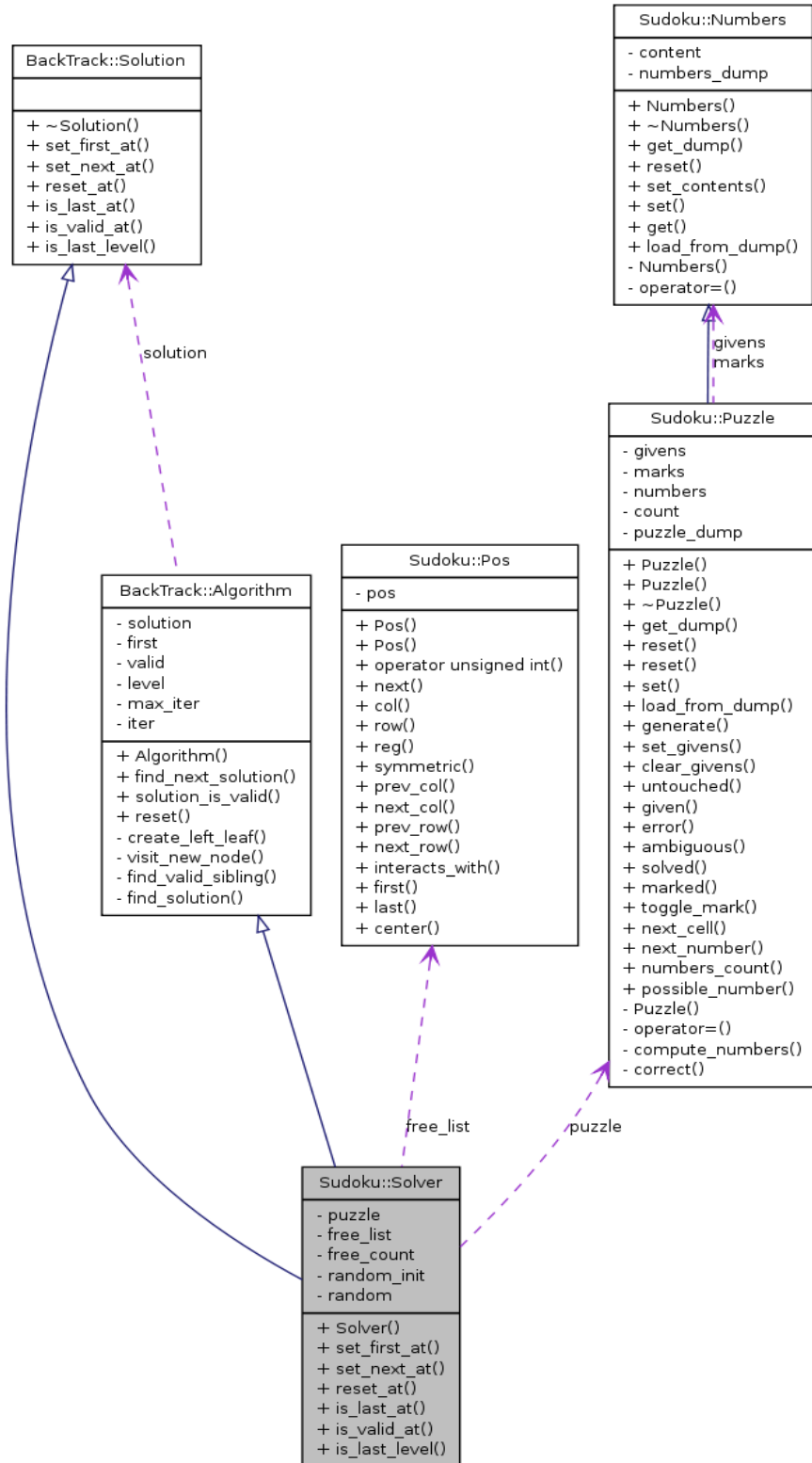


Fig 2.2 Data Flow Diagram



**Fig 2.3 Class Diagram for Sudoku**

## 2.3 Technology Description

### Overview

This project entails the development of a Java-based Sudoku application with GUI capabilities to generate, solve, and validate Sudoku puzzles at varying difficulty levels. The application uses the Swing framework for the user interface and employs algorithms for puzzle generation and solving.

### Key Features

1. **Sudoku Generator:**
  - **Puzzle Generation:** Create puzzles with different difficulty levels (Easy, Moderate, Hard) by adjusting the number of pre-filled cells.
  - **Randomization:** Utilize algorithms to generate a wide range of unique puzzles.
  - **Difficulty Adjustment:** Allow users to choose the difficulty level, which impacts the puzzle's complexity.
2. **Sudoku Solver:**
  - **Backtracking Algorithm:** Implement a backtracking algorithm to solve puzzles efficiently.
  - **Input Validation:** Check user inputs for validity and provide visual feedback.
  - **Real-time Solving:** Solve user-input puzzles and display the solution instantly.
3. **User Interface:**
  - **Interactive Grid:** Allow users to input numbers and interact with the puzzle grid.
  - **Level Selection:** Provide an intuitive interface for selecting puzzle difficulty levels.
  - **Controls:** Include buttons for generating puzzles, resetting the grid, solving the puzzle, and navigating between game screens.
4. **Additional Features:**
  - **Highlight Invalid Inputs:** Indicate incorrect user inputs visually.
  - **Non-editable Pre-filled Cells:** Prevent modification of pre-filled cells to avoid mistakes.
  - **User Guidance:** Provide feedback and hints to assist users in solving puzzles.

### Technical Stack

1. **Java Swing:**
  - **GUI Components:** Use Swing components (JFrame, JPanel, JButton, JTextField, etc.) to build the user interface.
  - **Event Handling:** Implement ActionListener for handling button clicks and other user interactions.
  - **Layouts:** Use various layout managers (BorderLayout, GridLayout, etc.) to organize the interface elements.
2. **Algorithms:**

- **Puzzle Generation Algorithm:** Implement a custom algorithm for generating Sudoku puzzles with varying levels of difficulty based on the number of initially filled cells.
- **Backtracking Algorithm:** Utilize backtracking for solving the Sudoku puzzles, ensuring correctness by checking the validity of each input during the solving process.
- **Input Validation:** Implement validation logic to ensure user inputs conform to Sudoku rules.

## Workflow and Architecture

1. **User Interface Setup:**
  - **Level Selector Panel:** Create a panel for selecting difficulty levels (Easy, Moderate, Hard) with custom-styled buttons.
  - **Game Panel:** Create a panel for the Sudoku grid and control buttons (Generate, Reset, Solve, Back to Level Selection).
  - **Dynamic Panel Switching:** Implement functionality to switch between the level selector panel and the game panel based on user actions.
2. **Puzzle Generation:**
  - **Difficulty Level Selection:** User selects a difficulty level, triggering the generation of a new puzzle.
  - **Grid Initialization:** Initialize the Sudoku grid with the generated puzzle, ensuring pre-filled cells are non-editable.
3. **Puzzle Solving:**
  - **User Input:** Users input numbers into the Sudoku grid.
  - **Validation:** Validate the inputs and highlight any incorrect values.
  - **Solving Mechanism:** Use the backtracking algorithm to solve the puzzle and display the solution.
4. **Reset and Navigation:**
  - **Reset Functionality:** Clear the current grid to allow users to start over.
  - **Back Navigation:** Provide a way to return to the level selection screen from the game panel.

### 2.3.1 Backtracking algorithm

**Backtracking** is a problem-solving algorithmic technique that involves finding a solution incrementally by trying different options and undoing them if they lead to a dead end. It is commonly used in situations where you need to explore multiple possibilities to solve a problem, like searching for a path in a maze or solving puzzles like Sudoku. When a dead end



is reached, the algorithm backtracks to the previous decision point and explores a different path until a solution is found or all possibilities have been exhausted.

```

procedure EXPLORE(node n)
  if REJECT(n) then return
  if COMPLETE(n) then
    OUTPUT(n)
  for  $n_i$  : CHILDREN(n) do EXPLORE( $n_i$ )
  
```

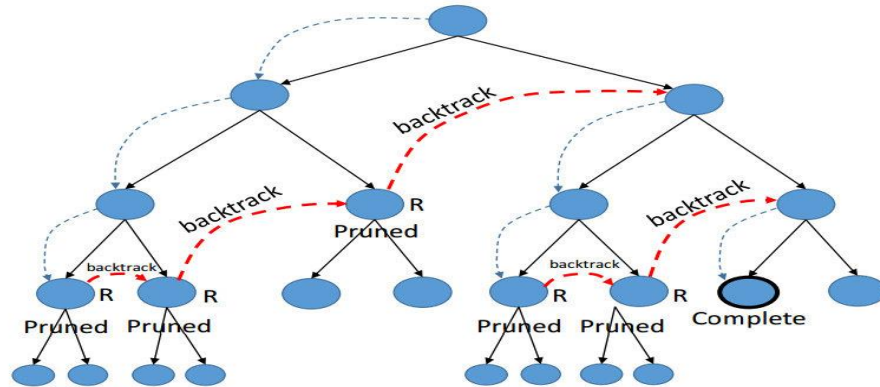


Fig 2.4 Backtracking Algorithm

A **backtracking algorithm** works by recursively exploring all possible solutions to a problem. It starts by choosing an initial solution, and then it explores all possible extensions of that solution. If an extension leads to a solution, the algorithm returns that solution. If an extension does not lead to a solution, the algorithm backtracks to the previous solution and tries a different extension.

The following is a general outline of how a backtracking algorithm works:

1. Choose an initial solution.
2. Explore all possible extensions of the current solution.
3. If an extension leads to a solution, return that solution.
4. If an extension does not lead to a solution, backtrack to the previous solution and try a different extension.
5. Repeat steps 2-4 until all possible solutions have been explored.

## CHAPTER 3 - IMPLEMENTATION AND TESTING

### 3.1 Algorithm and Code snippets

We can observe that all 3 x 3 matrices, which are diagonally present are independent of other 3 x 3 adjacent matrices initially, as others are empty.

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

So if we fill them first, then we will only have to do box check and thus column/row check not required.

Following is the improved logic for the problem.

1. Fill all the diagonal 3x3 matrices.
2. Fill recursively rest of the non-diagonal matrices.  
For every cell to be filled, we try all numbers until we find a safe number to be placed.
3. Once matrix is fully filled, remove K elements randomly to complete game.

### Sudoku Generator

```
public void fillValues()
{
    // Fill the diagonal of SRN x SRN matrices
    fillDiagonal();

    // Fill remaining blocks
    fillRemaining(0, SRN);

    // Remove Randomly K digits to make game
    removeKDigits();
}

int randomGenerator(int num)
```

```

{
    return (int) Math.floor( (Math.random() *num+1) );
}

```

These above methods are used to generate the sudoku puzzle. Here we use Random class present in java to get random numbers.

Difficulty level:

We will choose difficulty based on the empty boxes the user have to fill. As the difficulty increases empty boxes also increase.

```

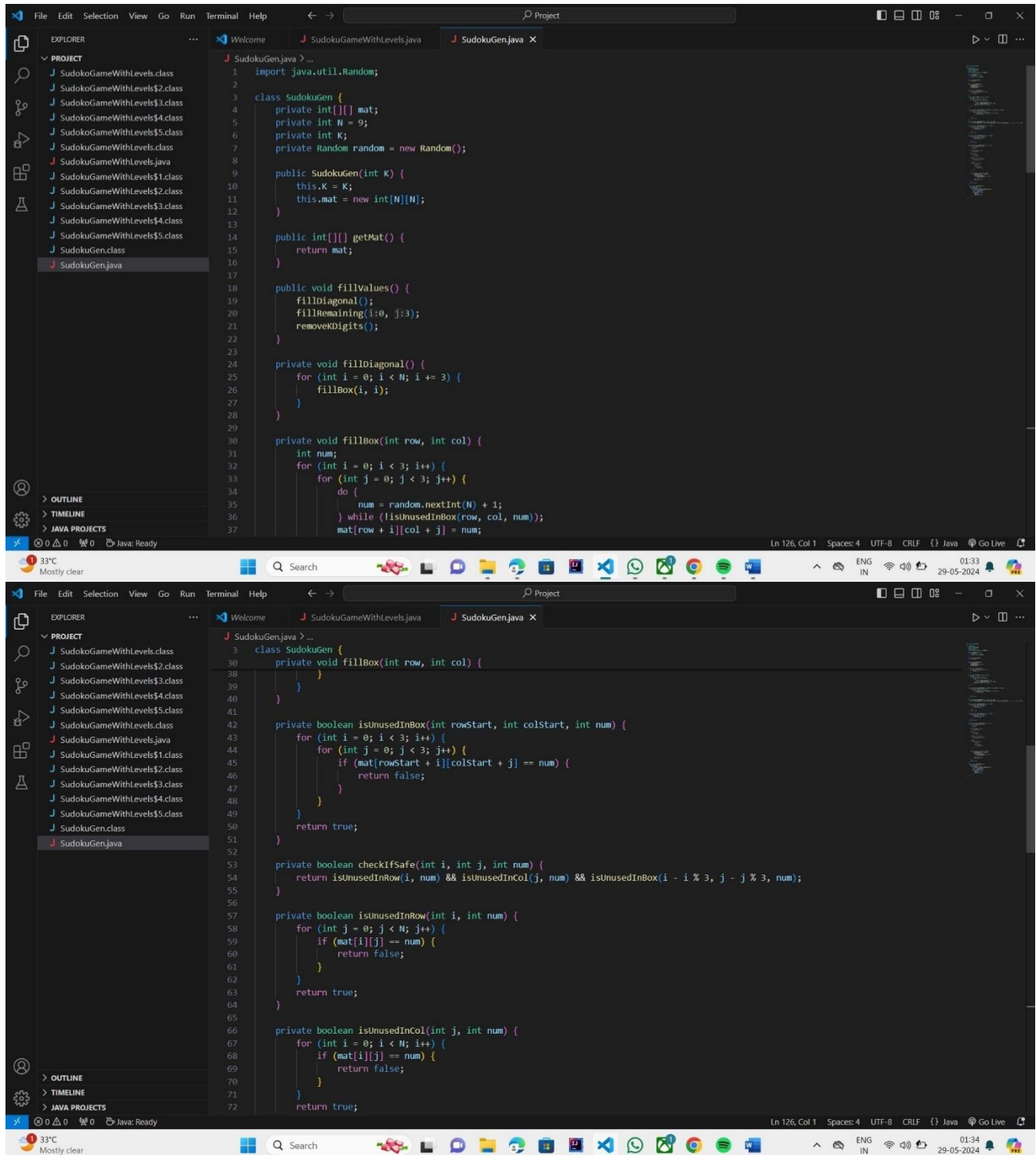
public void removeKDigits()
{
    int count = K;
    while (count != 0)
    {
        int cellId = randomGenerator(N*N)-1;

        // System.out.println(cellId);
        // extract coordinates i and j
        int i = (cellId/N);
        int j = cellId%N;

        // System.out.println(i+" "+j);
        if (mat[i][j] != 0)
        {
            count--;
            mat[i][j] = 0;
        }
    }
}

```

### 3.1.3 Deployment stage

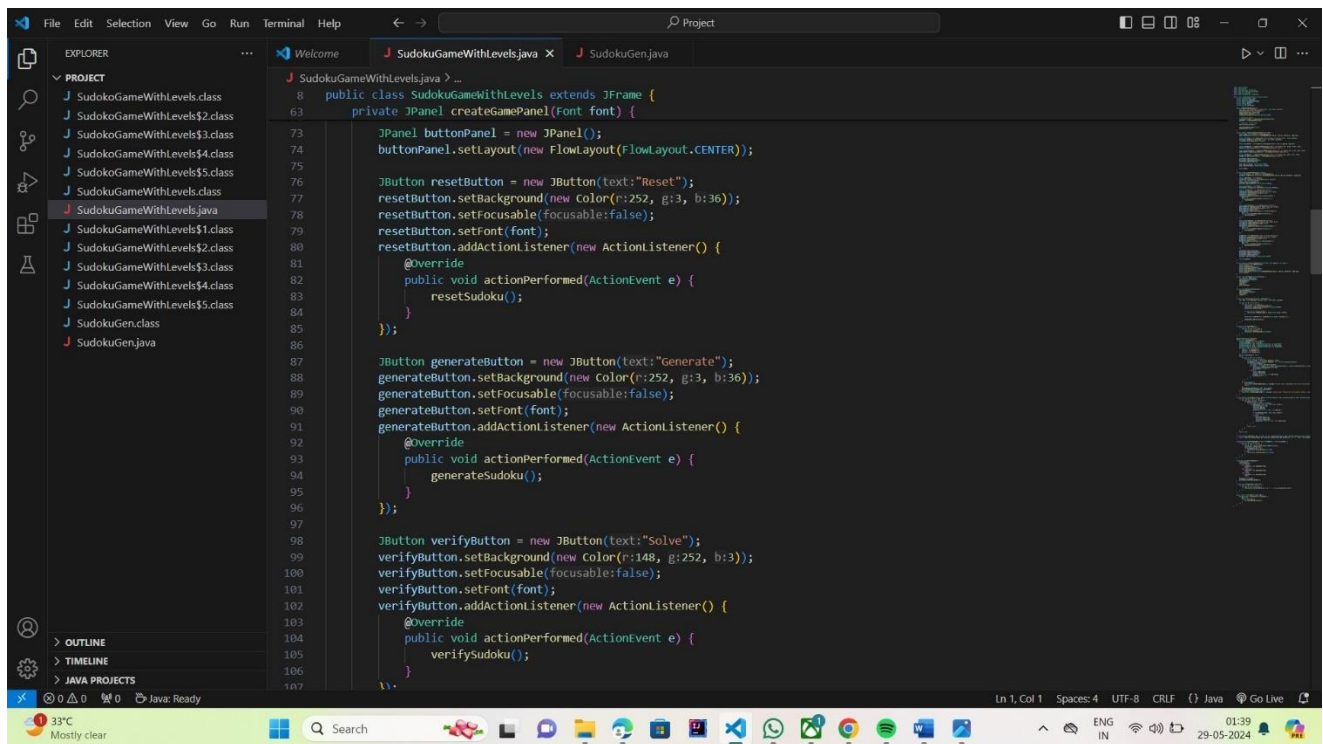


```
3 class SudokuGen {
66 private boolean isEmptyCell(int i, int j) {
73 }
74
75 private boolean fillRemaining(int i, int j) {
76     if (j >= N && i < N - 1) {
77         i = i + 1;
78         j = 0;
79     }
80     if (i >= N && j >= N) {
81         return true;
82     }
83     if (i < 3) {
84         if (j < 3) {
85             j = 3;
86         }
87     } else if (i < N - 3) {
88         if (j == (i / 3) * 3) {
89             j = j + 3;
90         }
91     } else {
92         if (j == N - 3) {
93             i = i + 1;
94             j = 0;
95             if (i >= N) {
96                 return true;
97             }
98         }
99     }
100
101     for (int num = 1; num <= N; num++) {
102         if (checkIfSafe(i, j, num)) {
103             mat[i][j] = num;
104             if (fillRemaining(i, j + 1)) {
105                 return true;
106             }
107             mat[i][j] = 0;
108         }
109     }
110     return false;
111 }
112
113 private void removeKDigits() {
114     int count = K;
115     while (count != 0) {
116         int cellid = random.nextInt(N * N);
117         int i = (cellid / N);
118         int j = cellid % N;
119         if (mat[i][j] != 0) {
120             count--;
121             mat[i][j] = 0;
122         }
123     }
124 }
125
126 }
```

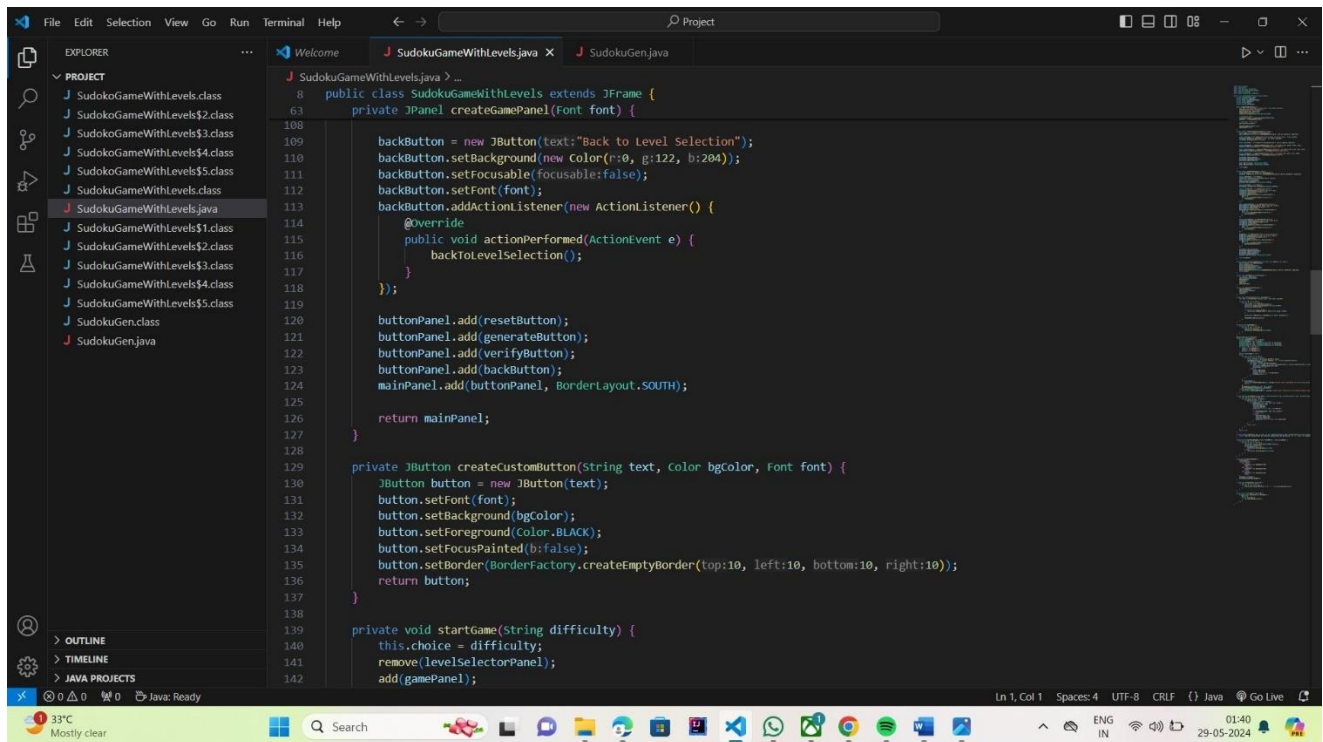
```
3 class SudokuGen {
75 private boolean fillRemaining(int i, int j) {
92     if (j == N - 3) {
93         i = i + 1;
94         j = 0;
95         if (i >= N) {
96             return true;
97         }
98     }
99
100     for (int num = 1; num <= N; num++) {
101         if (checkIfSafe(i, j, num)) {
102             mat[i][j] = num;
103             if (fillRemaining(i, j + 1)) {
104                 return true;
105             }
106             mat[i][j] = 0;
107         }
108     }
109     return false;
110 }
111
112 private void removeKDigits() {
113     int count = K;
114     while (count != 0) {
115         int cellid = random.nextInt(N * N);
116         int i = (cellid / N);
117         int j = cellid % N;
118         if (mat[i][j] != 0) {
119             count--;
120             mat[i][j] = 0;
121         }
122     }
123 }
124
125
126 }
```



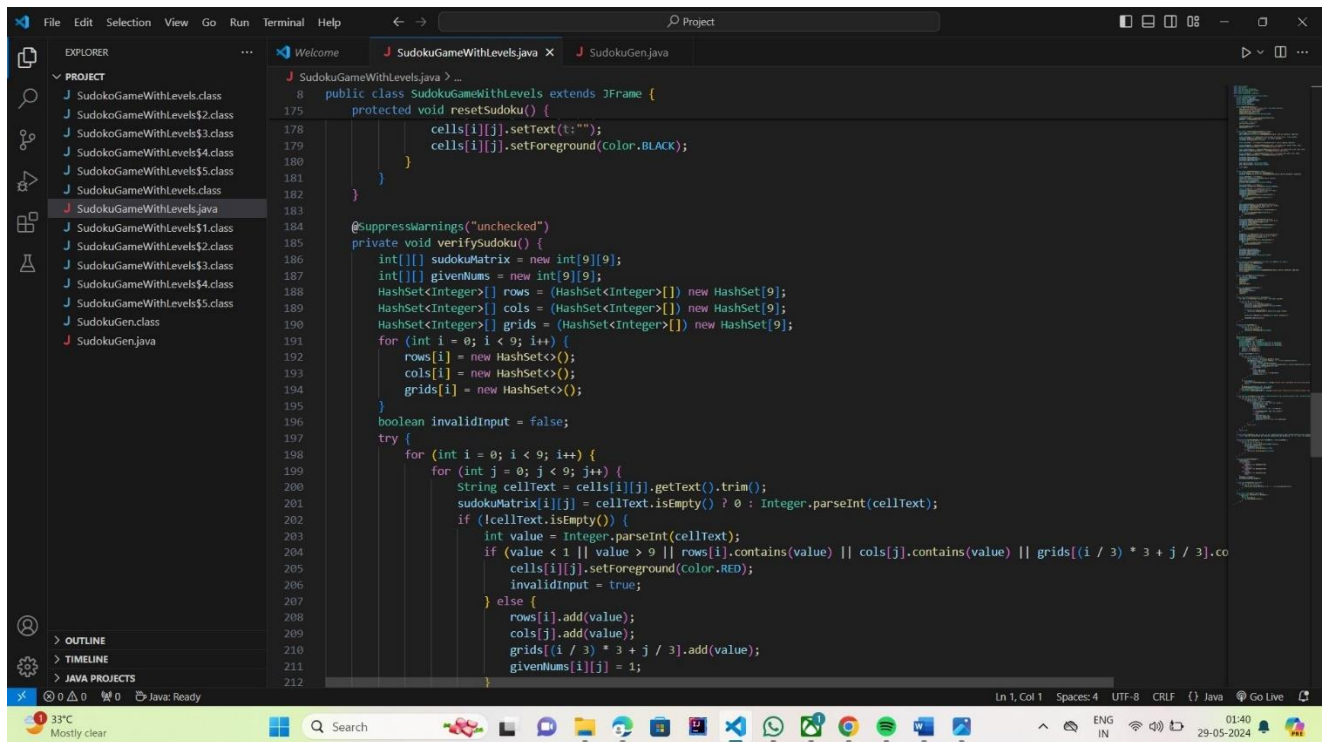
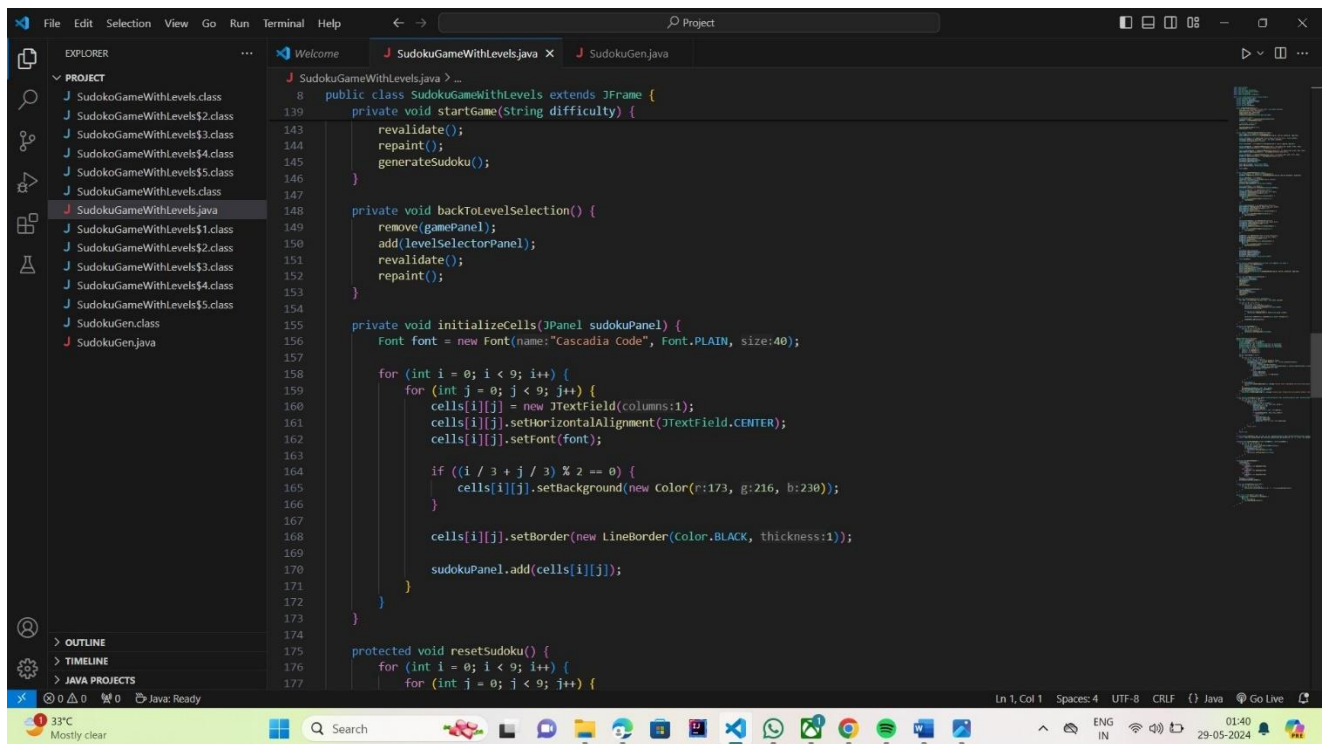




```
8 public class SudokuGameWithLevels extends JFrame {
63 private JPanel createGamePanel(Font font) {
73     JPanel buttonPanel = new JPanel();
74     buttonPanel.setLayout(new FlowLayout(FlowLayout.CENTER));
75
76     JButton resetButton = new JButton(text:"Reset");
77     resetButton.setBackground(new Color(r:252, g:3, b:36));
78     resetButton.setFocusable(false);
79     resetButton.setFont(font);
80     resetButton.addActionListener(new ActionListener() {
81         @Override
82         public void actionPerformed(ActionEvent e) {
83             resetSudoku();
84         }
85     });
86
87     JButton generateButton = new JButton(text:"Generate");
88     generateButton.setBackground(new Color(r:252, g:3, b:36));
89     generateButton.setFocusable(false);
90     generateButton.setFont(font);
91     generateButton.addActionListener(new ActionListener() {
92         @Override
93         public void actionPerformed(ActionEvent e) {
94             generateSudoku();
95         }
96     });
97
98     JButton verifyButton = new JButton(text:"Solve");
99     verifyButton.setBackground(new Color(r:148, g:252, b:3));
100    verifyButton.setFocusable(false);
101    verifyButton.setFont(font);
102    verifyButton.addActionListener(new ActionListener() {
103        @Override
104        public void actionPerformed(ActionEvent e) {
105            verifySudoku();
106        }
107    });
108 }
```



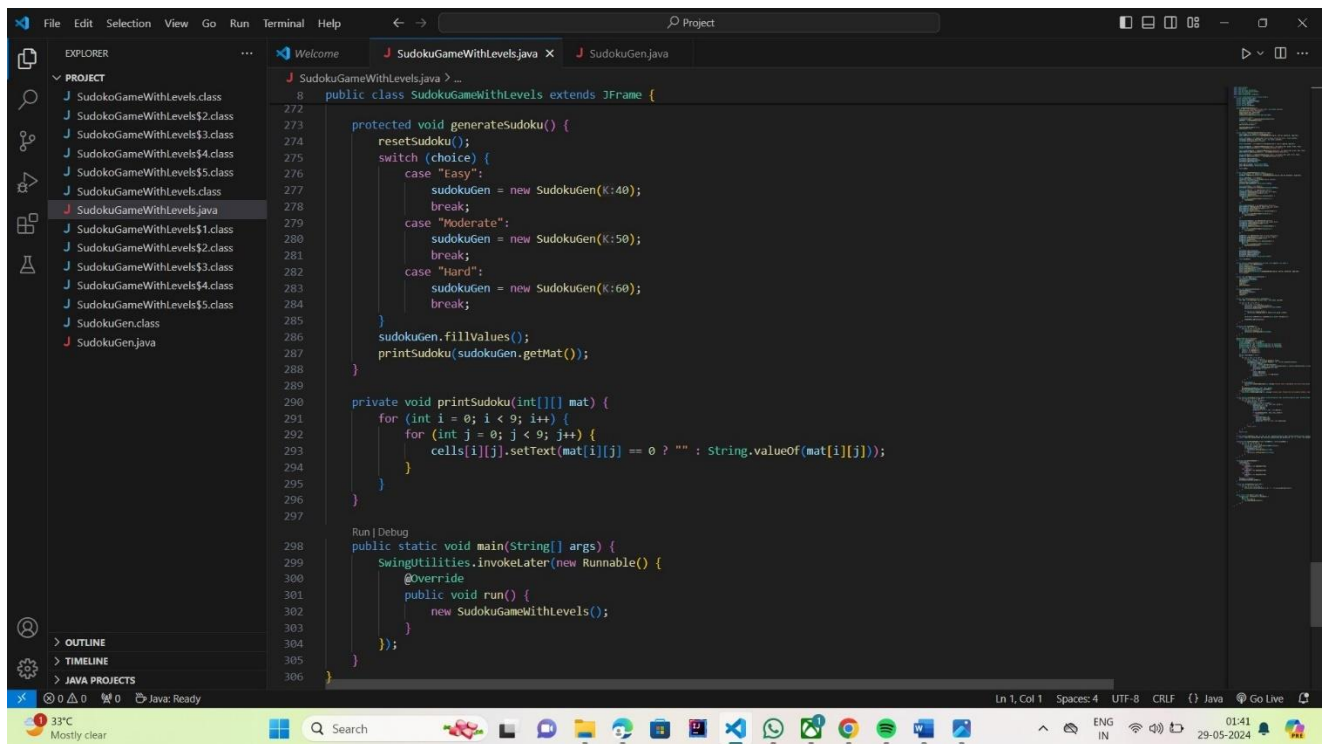
```
108     backButton = new JButton(text:"Back to Level Selection");
109     backButton.setBackground(new Color(r:0, g:122, b:284));
110     backButton.setFocusable(false);
111     backButton.setFont(font);
112     backButton.addActionListener(new ActionListener() {
113         @Override
114         public void actionPerformed(ActionEvent e) {
115             backToLevelSelection();
116         }
117     });
118
119     buttonPanel.add(resetButton);
120     buttonPanel.add(generateButton);
121     buttonPanel.add(verifyButton);
122     buttonPanel.add(backButton);
123     mainPanel.add(buttonPanel, BorderLayout.SOUTH);
124
125     return mainPanel;
126 }
127
128 private JButton createCustomButton(String text, Color bgColor, Font font) {
129     JButton button = new JButton(text);
130     button.setFont(font);
131     button.setBackground(bgColor);
132     button.setForeground(Color.BLACK);
133     button.setFocusPainted(false);
134     button.setBorder(BorderFactory.createEmptyBorder(top:10, left:10, bottom:10, right:10));
135     return button;
136 }
137
138 private void startGame(String difficulty) {
139     this.choice = difficulty;
140     remove(levelSelectorPanel);
141     add(gamePanel);
142 }
```





```
8 public class SudokuGameWithLevels extends JFrame {
185 private void verifySudoku() {
213 }
214 }
215 }
216 if (invalidInput) {
217 JOptionPane.showMessageDialog(this, message: "Invalid input. Highlighted cells have wrong values.");
218 return;
219 }
220 solveSudoku(sudokuMatrix, rows, cols, grids);
221 printSolvedSudoku(sudokuMatrix, givenNums);
222 } catch (NumberFormatException ex) {
223 JOptionPane.showMessageDialog(this, message: "Invalid input. Please enter only numbers between 1 and 9.");
224 }
225 }
226 }
227 private boolean solveSudoku(int[][] board, HashSet<Integer>[] rows, HashSet<Integer>[] cols, HashSet<Integer>[] grids) {
228 for (int row = 0; row < 9; row++) {
229 for (int col = 0; col < 9; col++) {
230 if (board[row][col] == 0) {
231 for (int num = 1; num <= 9; num++) {
232 if (isValid(num, row, col, rows, cols, grids)) {
233 board[row][col] = num;
234 rows[row].add(num);
235 cols[col].add(num);
236 grids[(row / 3) * 3 + (col / 3)].add(num);
237 }
238 if (solveSudoku(board, rows, cols, grids)) {
239 return true;
240 } else {
241 board[row][col] = 0;
242 rows[row].remove(num);
243 cols[col].remove(num);
244 grids[(row / 3) * 3 + (col / 3)].remove(num);
245 }
246 }
247 }
```

```
227 private boolean solveSudoku(int[][] board, HashSet<Integer>[] rows, HashSet<Integer>[] cols, HashSet<Integer>[] grids) {
247 }
248 return false;
249 }
250 }
251 return true;
252 }
253 }
254 }
255 private boolean isValid(int num, int row, int col, HashSet<Integer>[] rows, HashSet<Integer>[] cols, HashSet<Integer>[] grids) {
256 return !rows[row].contains(num) && !cols[col].contains(num) && !grids[(row / 3) * 3 + (col / 3)].contains(num);
257 }
258 }
259 protected void printSolvedSudoku(int[][] solvedMatrix, int[][] givenNums) {
260 for (int i = 0; i < 9; i++) {
261 for (int j = 0; j < 9; j++) {
262 String num = Integer.toString(solvedMatrix[i][j]);
263 cells[i][j].setText(num);
264 if (givenNums[i][j] != 1) {
265 cells[i][j].setForeground(Color.RED);
266 } else {
267 cells[i][j].setForeground(Color.BLACK);
268 }
269 }
270 }
271 }
272 }
273 protected void generatesudoku() {
274 resetSudoku();
275 switch (choice) {
276 case "Easy":
277 sudokuGen = new SudokuGen(K:40);
278 break;
279 case "Moderate":
280 sudokuGen = new SudokuGen(K:50);
281 break;
```



## 3.2 Test cases

### Test Cases


#### 3.2.1 Pilot page

Scenario TID	Scenario Description	Test Case ID	Pre Condition	Steps to Execute	Expected Result	Actual Result	Status
1	Execute .class file for sudoku	TID_1	java SudokuGameWithLevels	Command Prompt	file executed and a panel pops up	file executed and a panel pops up	Pass
2	Level selection for user	TID_2	Easy,Moderate,Hard	Jpanel	directs to the new panel based on level	directs to the new panel based on level	Pass
3	Generates the sudoku puzzle based on the level selected	TID_3	Level selection	Jpanel	Generated puzzle based on level	Generated puzzle based on level	Pass
4	User gives Valid input	TID_4	with valid input	Jpanel	No error	No error	Pass
5	User gives invalid input	TID_5	with invalid input	Jpanel	Highlights the cell with invalid input on clicking solve	Highlights the cell with invalid input on clicking solve	Pass
6	User selects solve puzzle	TID_6	with valid data	Jpanel	No error is generated and solve done	No error is generated and solve done	Pass
7	User selects solve puzzle	TID_7	with invalid data	Jpanel	Highlights the cell with invalid input on clicking solve	Highlights the cell with invalid input on clicking solve	Pass
8	User selects solve puzzle	TID_8	without any data	Jpanel	Solves the puzzle correctly	Solves the puzzle correctly	Pass
9	Reset the puzzle	TID_9	Puzzle completed	Jpanel	Empty cells generated and the previous data gets lost	Empty cells generated and the previous data gets lost	Pass

Fig 3.1 Test Cases for Sudoku Deployment



Fig3.2 user will be given 3 options of 1. Easy 2.Moderate 3.Hard to choose the difficulty.


Sudoku Challenger
—
□
×

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

Reset
Generate
Solve
Back to Level Selection

Fig3.3

The fig3.3 shows the generated sudoku puzzle of the respective difficulty. User is also provided with certain features of Reset,Generate,Solve,Back to Level Selection.

### 3.2.2 Solving the puzzle

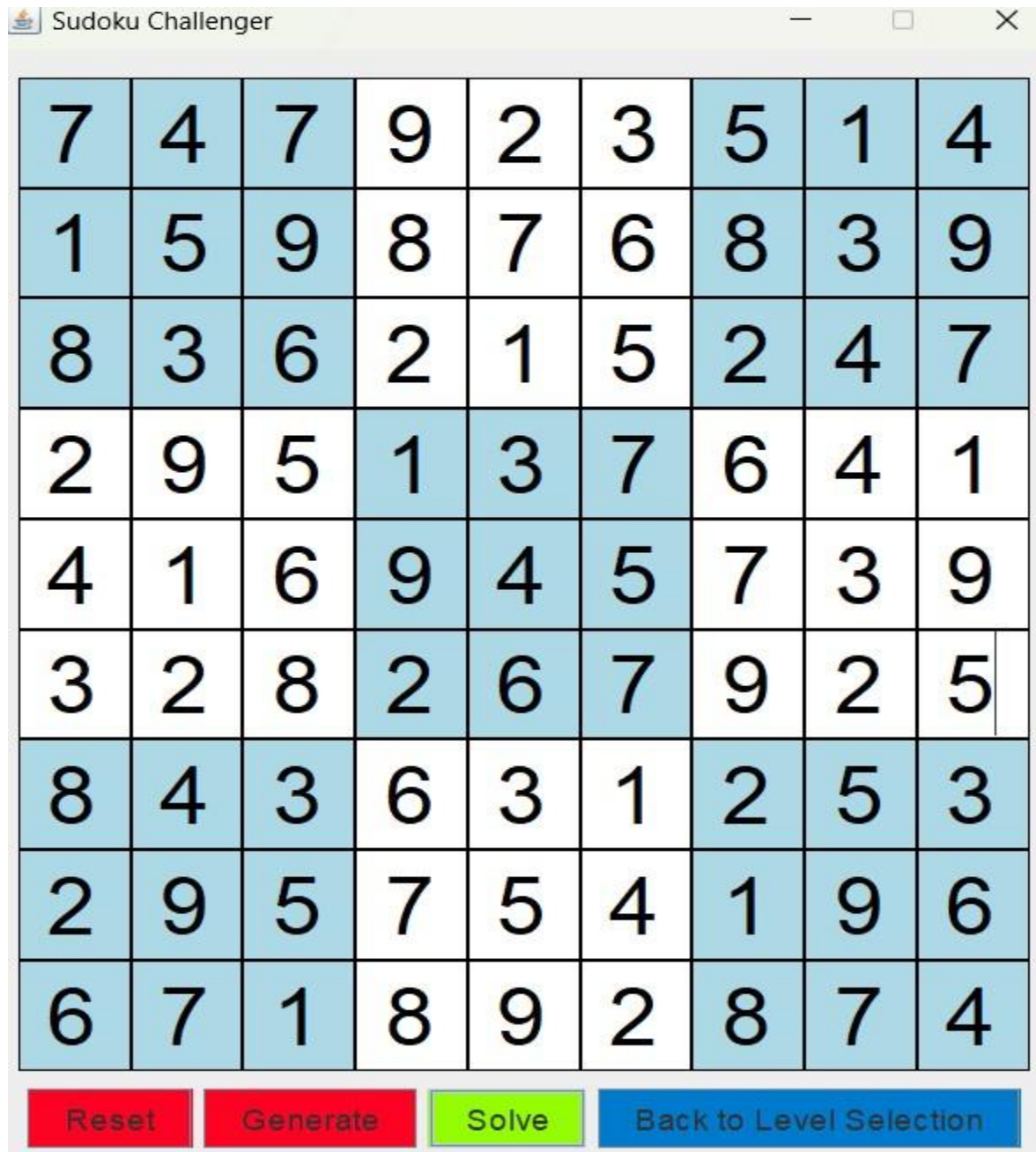




Fig 3.4 demo puzzle filled by a user.

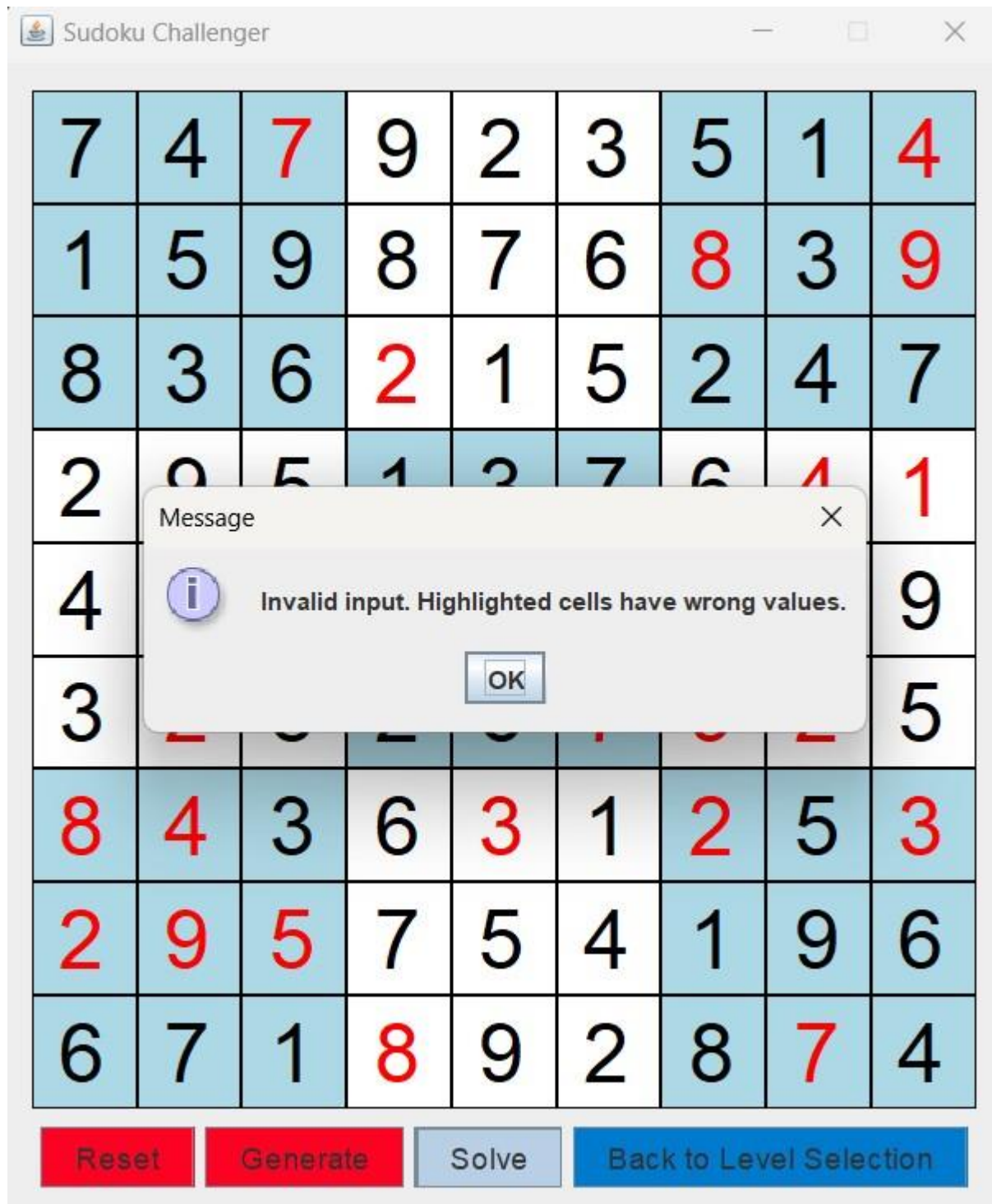


Fig3.5 user entered values are incorrect and the incorrect one's are highlighted

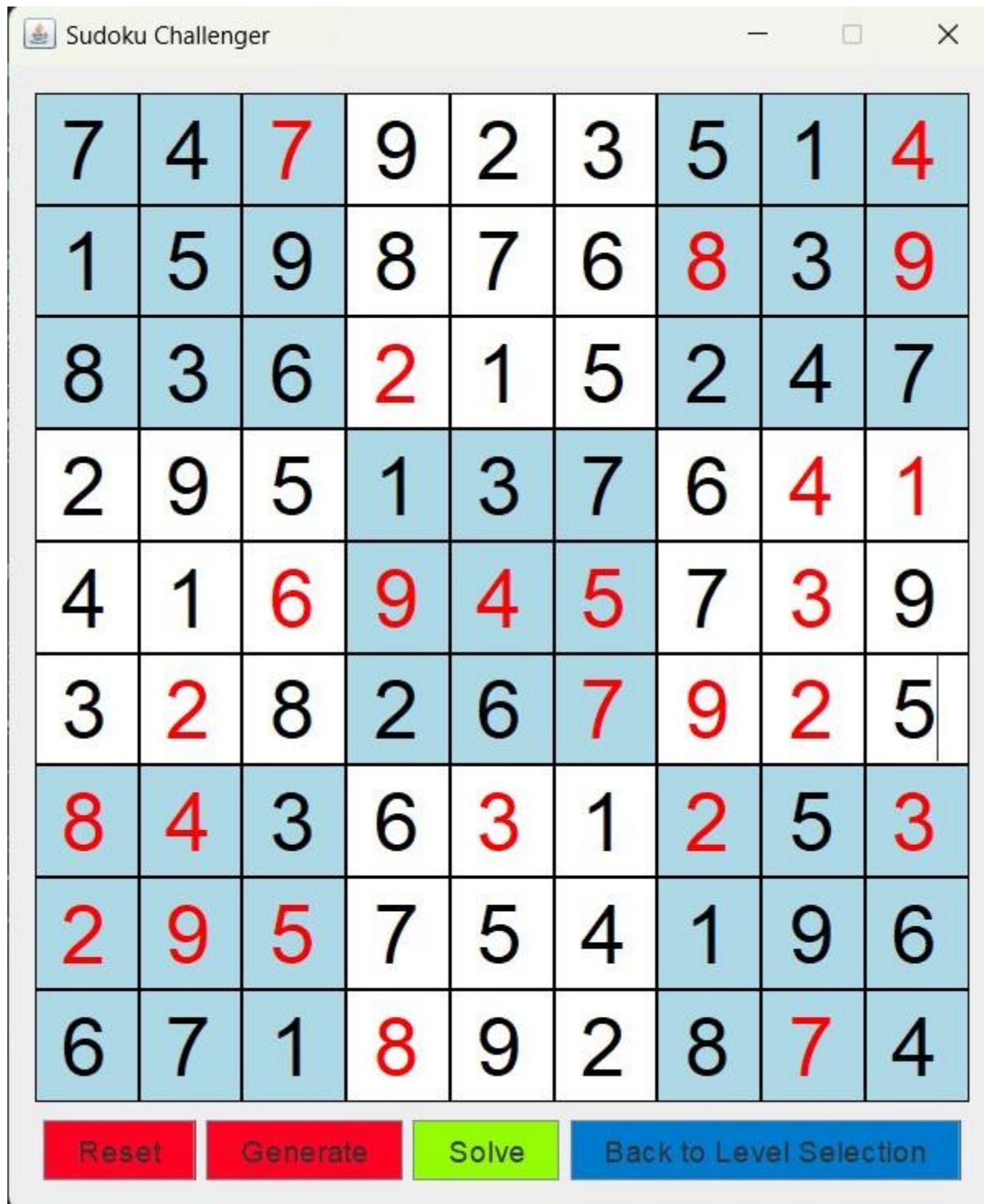


Fig 3.6 Highlighted incorrect values

## 4.1 CONCLUSION

This project aims to deliver a robust and efficient Sudoku solver capable of handling a wide range of puzzles. By focusing on accuracy, performance, and user experience, application will provide valuable assistance to Sudoku enthusiasts and contribute to the field of puzzle-solving algorithms.

## 4.2 BIBLIOGRAPHY

1.Java: The Complete Reference 12<sup>th</sup> edition by Herbert Schildt.

### About the book

Java™: The Complete Reference, Twelfth Edition explains how to develop, compile, debug, and run Java programs. This book covers the entire Java language, including its syntax, keywords, and fundamental programming principles. You'll also find information on key portions of the Java API library, such as I/O, the Collections Framework, the stream library, and the concurrency utilities. Swing, JavaBeans, and servlets are examined, and numerous examples demonstrate Java in action.

2.Java Swing Tutorial- javatpoint and Geeks for geeks

Retrieved From: <https://www.javatpoint.com/java-swing> ,  
<https://www.geeksforgeeks.org/introduction-to-java-swing/>

3.Sudoku Generator Algorithm – 101 Computing

Retrieved From: <https://www.101computing.net/sudoku-generator-algorithm/>

4.BackTracking Algorithm -Geeks for geeks

Retrieved From: <https://www.geeksforgeeks.org/backtracking-algorithms/>