Dhaka International University

Project
On
Sudoku Simulation using java OOP and java swing

Department of CSE

Submitted by :

Shihab Shariar (11)
Batch : D-72

Submitted to :

Abdul Based
Professor
Department of CSE
Dhaka International University

Submission date : 28th April, 2024

Abstract :

Sudoku is a popular number based game . The game is played inside a grid system like 3 by 3 , 9 by 9 or 12 by 12 . The board is filled with some random numbers and some blocks remain empty to be filled with . By following some rules for each grid system the game is played . For example , a 9 by 9 grid to be considered where the puzzle will be fully filled with 1 to 9 in each empty box for row wise or column wise but some rules are fixed to fill those blocks . The rules for the 9 by 9 grid are :

- Each row needs to be filled with numbers between 1 to 9 with no repetitions.
- Each column also needs to be filled with numbers between 1 to 9 without any repetitions.
- Each 3 by 3 quadrant must have numbers between 1 to 9 with no repetitions .

There is only one solution available for the puzzle .

Objectives : Now to design a simulation with java first we have to construct our logic to generate a grid consisting of the full solution filled . Then we need to remove numbers randomly from the blocks to recreate a puzzle .

Key Findings :

- To design the whole concept, you need to learn the java GUI or graphical interface topics.
- Need to learn some programming concepts like recursion , backtracking .
- Need a wide range of knowledge on logic build up .
- Need knowledge on java object oriented programming .
- Need knowledge on java package , class , access modifiers , static elements .
- Need knowledges on java AWT (abstract window toolkit)

Overall after completing all of the required skills in java we will be able to design the simulations for the sudoku game of 9 by 9 grid which will be the simplest form of a sudoku board to play for a single time game . Sudoku is a very challenging and logical game that helps to brainstorm in numbers . Our simulation is divided into three levels which will be discussed as follows to know about that . We also implemented some special features to make the game challenging . We also have some limitations in our simulation which can be resolved day by day while we face any bugs or errors . Our goal is to provide the simplest form of a sudoku game while ignoring the critical concepts or the complications or special terms in the sudoku game .

Table of Contents :

Chapter 1 .
(Introduction)

1.1 Context and Background :

Sudoku is a number based puzzle game which can be played inside a grid where some blocks of the grid are filled with random numbers between a specific range . To fill all the blocks one can solve the puzzle but the numbers filled in the blocks will be unique and in between the range . The numbers must not repeat inside the row and column it is submitted . It is a logic based numeric puzzle game which improves our calculating performance and logic build up timing .

The numeric puzzles were started publishing in the newspapers in the 19th century by the French puzzle setters .But those puzzles were based on magic squares . On **July 6, 1895** the modern sudoku was born . The modern sudoku was designed by a 74 year old retired architect named **Howard Garns .**

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

Figure 1.1(a)

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

Figure 1.1(b)

1.2 Goals and Objectives :

While designing our project our goal was to make the sudoku simulation in a short time period with limited knowledge based on java utilities but in a standard form . Our main motive was to represent a fully functional sudoku 9 by 9 board to play the puzzle game and enjoy logical thinking or challenging behavior expressions .

Our objectives were :

- Make the project as simple as possible by using limited knowledges in a short time period
- Determine complex engineering concepts
- Handling exceptions
- Analyze complex logics and make functional view using those logics
- Provide a stable light graphical interface parallelly keeping the code as simple as possible

1.3 Scopes and Limitations :

In the competitive environment of developing any gaming simulation we can offer our code as the simplest form of any other simulation presented on the market . Our code uses java swing which provides a lightweight graphical user interface with a limited number of codes . Our code is easier to read and understand and it gives transparency to other developers to develop it more efficiently .

Our simulation only can be used to generate a 9 by 9 grid but sudoku has many other variants like 3 by 3 , 12 by 12 , 100 by 100 and many more . Our code has not too much efficient runtime; it can be improved further gradually . Our board generates puzzles randomly in which we do not have any control to manipulate . Our code is not fully optimized to handle all types of exceptions , after testing over and over it might be improved .

Chapter 2 .
(Methodology)

2.1 Descriptions of design :

Our source code :

First class (main class)

```
package sudokuusingjavaswing;

import java.awt.BorderLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
```

```
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;

public class StartInterface{
   private static JFrame nameFrame;
   private static JFrame levelFrame;
   private static JTextField nameField;
   public static String player_name; // storing name to use later

   // first popup view to take name input
   public void ResponseWindow(){
      nameFrame = new JFrame("Player name : ");
      nameFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      nameFrame.setBounds(600, 300,400,200);

      // making partition for each row like enter your name text , blank field to taka name input
and submit button
      GridLayout layout = new GridLayout(3,1);
      JPanel panel = new JPanel(layout);
      JLabel nameLabel = new JLabel("Enter your name...");
      nameField = new JTextField();

      panel.add(nameLabel);
      panel.add(nameField);

      // create submit button
      JButton submitBtn = new JButton("Submit");

      // customize submit button function for an onclick event
      ActionListener click = new ActionListener(){
         @Override
         public void actionPerformed(ActionEvent a){
            String name = nameField.getText().trim();
            if(name.isEmpty())
               JOptionPane.showMessageDialog(nameFrame,"Invalid Input", "Error",
JOptionPane.ERROR_MESSAGE);
            else{
```

```
                player_name = name;
                LevelSelectionWindow(name);
                nameFrame.dispose();
            }
        }
    };
    // jpanel customization
    panel.add(submitBtn);
    submitBtn.addActionListener(click);
    nameFrame.getContentPane().add(panel,BorderLayout.CENTER);
    nameFrame.getContentPane().add(panel,BorderLayout.SOUTH);
    nameFrame.setVisible(true);
}

// method to create difficulty selection window
private void LevelSelectionWindow(String name){
    levelFrame = new JFrame("Select level");
    levelFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    levelFrame.setBounds(600, 300,400, 200);

    // making grid for each option like easy ,medium,hard , button
    GridLayout layout = new GridLayout(4,1);
    JPanel panel = new JPanel(layout);
    JLabel nameLable = new JLabel (name+" Chose level : ");
    JButton easyBtn = new JButton("Easy");
    JButton mediumBtn = new JButton("Medium");
    JButton hardBtn = new JButton("Hard");

    // making objects to get response for a onclick event for each level or difficulty chose by
user
    ActionListener c1 = new ActionListener(){
        @Override
        public void actionPerformed(ActionEvent a){
            JOptionPane.showMessageDialog(levelFrame, "Easy Game starting ...");
            levelFrame.dispose();
            SudokuGame ob = new SudokuGame(3);
        }
    };
    ActionListener c2 = new ActionListener(){
        @Override
```

```
        public void actionPerformed(ActionEvent a){
            JOptionPane.showMessageDialog(levelFrame, "Medium Game starting ...");
            levelFrame.dispose();
            SudokuGame ob = new SudokuGame(7);
        }
    };
    ActionListener c3 = new ActionListener(){
        @Override
        public void actionPerformed(ActionEvent a){
            JOptionPane.showMessageDialog(levelFrame, "Hard Game starting ...");
            levelFrame.dispose();
            SudokuGame ob = new SudokuGame(10);
        }
    };
    // add actionlisteners to the buttons
    easyBtn.addActionListener(c1);
    mediumBtn.addActionListener(c2);
    hardBtn.addActionListener(c3);

    // adding buttons to the jpanel
    panel.add(nameLable);
    panel.add(easyBtn);
    panel.add(mediumBtn);
    panel.add(hardBtn);

    // adding panel to the jframe for level frame
    levelFrame.getContentPane().add(panel,BorderLayout.CENTER);
    levelFrame.setVisible(true);
  }
  public static void main(String[] args) {
    StartInterface ob = new StartInterface();
    ob.ResponseWindow(); // this method holds all the design for frame functionality and
design
  }
}
```

Second class

```
package sudokuusingjavaswing;

import java.awt.Color;
```

```
import java.awt.GridLayout;
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Random;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;
import java.time.LocalTime;
import java.time.Duration;

public class SudokuGame extends JFrame {
    private JTextField [][] GridCells;
    private int [][] Solution;
    private boolean[][] FixedCells;
    private LocalTime StartTime;
    private LocalTime EndTime;

    public SudokuGame(int difficulty){
        StartTime = LocalTime.now();
        setTitle("9 by 9 Sudoku Board");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(500,150,600,600);
        setResizable(false);

        // using jpanel to make sudoku grid
        JPanel SudokuPanel = new JPanel();
        GridLayout GameGrid = new GridLayout(9,9);
        SudokuPanel.setLayout(GameGrid);

        //initialize the gridlayout
        GridCells = new JTextField[9][9];
        Solution = new int[9][9];
        FixedCells = new boolean[9][9];

        GenerateSudokuPuzzle(difficulty);

        // making text fields for the sudoku grid
```

```
        for(int i = 0;i<9;i++){
            for(int j = 0; j<9;j++){
                GridCells[i][j] = new JTextField(1);
                if(!FixedCells[i][j]){
                  GridCells[i][j].setBackground(Color.LIGHT_GRAY);
                  GridCells[i][j].setForeground(Color.BLACK);
                }else{
                   GridCells[i][j].setText(String.valueOf(Solution[i][j]));
                   GridCells[i][j].setEditable(false);
                   GridCells[i][j].setBackground(Color.BLACK);
                   GridCells[i][j].setForeground(Color.CYAN);
                }
                SudokuPanel.add(GridCells[i][j]);
            }
        }
        // making frame for the sudoku panel
        getContentPane().add(SudokuPanel, BorderLayout.CENTER);

        JButton doneButton = new JButton("Done ...");
        doneButton.addActionListener(new ActionListener(){
            @Override
            public void actionPerformed(ActionEvent e){
                if(CheckSolution()){
                    EndTime = LocalTime.now();
                    Duration duration = Duration.between(StartTime, EndTime);
                    long minutes =duration.toMinutes() ;
                    long seconds = duration.minusMinutes(minutes).getSeconds();
                    String time = "Congratulation ! "+StartInterface.player_name+" You Solved the
Game in \n"+String.valueOf(minutes)+" minutes "+String.valueOf(seconds)+" seconds";
//                    System.out.println(time);  // testing purpose
                    JOptionPane.showMessageDialog(SudokuGame.this,time);
                    dispose();
                }else{
                    JOptionPane.showMessageDialog(SudokuGame.this,"Oops ! Try again ");
//                    printSolution();  // this was for test purpose
                }
            }
        });
        getContentPane().add(doneButton,BorderLayout.SOUTH);
        setLocationRelativeTo(null);
```

```
    setVisible(true);
}

private void GenerateSudokuPuzzle(int difficulty){
    Random random = new Random();

    // generate a fully solved board
    GenerateSolutionFully(Solution,0,0);

    // filling all the cells with the solution
    for(int i = 0 ;i<9;i++){
        for(int j = 0;j<9;j++){
            GridCells[i][j] = new JTextField(String.valueOf(Solution[i][j]));
            FixedCells[i][j] = true;
        }
    }

    // adding difficulty low , medium and hard
    int level = difficulty;
    while (level > 0){
        int row = random.nextInt(9);
        int col = random.nextInt(9);
        if(FixedCells[row][col]){
            GridCells[row][col].setText("");
            FixedCells[row][col] = false;
            level --;
        }
    }
}

private boolean GenerateSolutionFully(int [][] Board, int row, int col){
    if(row == 9){
        row = 0;
        if(++col == 9){
            return true;
        }
    }
    if(Board[row][col] !=0){
        return GenerateSolutionFully(Board, row+1,col);
    }
```

```
        for(int n = 1 ; n <= 9; n++){
            if(isValid(Board,row,col,n)){
                Board[row][col]= n;
                if(GenerateSolutionFully(Board,row+1,col)){
                    return true;
                }
                Board[row][col]=0;
            }
        }
        return false;
    }

    private boolean isValid(int [][]Board, int row, int col, int val){
        for(int i=0;i<9;i++){
            if(Board [row][i]== val || Board[i][col] == val || Board[3*(row / 3) + i / 3 ][ 3 * (col / 3)+
i % 3] == val ){
                return false;
            }
        }
        return true;
    }

    private boolean CheckSolution(){
        int [][] UserSolution = new int [9][9];

        // take input from user and checking validity also
        for(int i =0 ;i<9;i++){
            for(int j=0;j<9;j++){
                String input = GridCells[i][j].getText();
                if(!input.isEmpty()){
                    try{
                        int num = Integer.parseInt(input);
                        UserSolution[i][j] = num ;
                    }catch ( NumberFormatException e){
                        return false; //error message box will activate
                    }
                }else{
                    return false;  //error message box will activate
                }
            }
```

```
        }
        for(int i = 0;i<9;i++){
            for(int j =0 ;j< 9 ;j++){
                if( UserSolution[i][j] != Solution[i][j]) {
                    return false;
                }
            }
        }
        return true;
    }

    // method to see the exact solution of the generated board , it was for test purpose
     private void printSolution() {
    System.out.println("Solved Sudoku Puzzle:");
    for (int i = 0; i < 9; i++) {
      for (int j = 0; j < 9; j++) {
        System.out.print(Solution[i][j] + " ");
      }
      System.out.println();
    }
  }
}
```

The whole code implementation was based on some premade concepts , those are :

- In the **first class main code** we worked on the starting works like taking name input , selecting level of difficulty and then redirecting into the **second class code** based on the selected level .
- In the **second class code** we implemented the whole gaming interface , performing sudoku logic to get a solution , checking the user given solution to the exact solution , showing everything in each graphical interface , making some line and method to test and debug which are commented inside the code .

2.2 Tools and Techniques :

We can separate our used tools and techniques into two parts as built in tools and our custom made tools and techniques .

Built in tools :

java.awt from this package we used BorderLayout , GridLayout, event , Color .

BorderLayout : set position of panel .

GridLayout : used to make Grid views .
event : ActionListner and ActionEvent are used to create functional buttons .
Color : It is used to set foreground or background color .

javax.swing from this package we used JButton , JFrame , JLabel, JOptionPane, JPanel, JTextField.

JButton : Used to create buttons .
JFrame : frames are created using this .
JLabel : Creates label for a frame .
JOptionPane : it is used to call different dialogue ,messagebox or inputbox.
JPanel : used to create panels for frame
JTextField : it is used to handle text related fields in jframe .

java.util from this package we used Random .

Random : used to generate random numbers .

java.time from this package we used LocalTime, Duration.

LocalTime : used to get time data .
Duration : It is used to calculate time related calculations .

Custom made tools :

We just applied logic and created some methods . They are :

ResponseWindow() : the whole code starts with this method .

LevelSelectionWindow(receives string parameter **):** Gives a pop up window to select levels.

GenerateSudokuPuzzle(receives integer parameter to set difficulty**) :** generate sudoku board.

GenerateSolutionFully(takes integer array to store complete solution in rows and columns**):** generate actual solution for the board and store the solution.

isValid(receives integer array, row,column and value**):** check input value's validity.

CheckSolution(): compare users completed game solution to actual solution and returns true false as answer.

printSolution(): it was created for test purposes .

Chapter 3 .
(Results)

3.1 Graphical View :

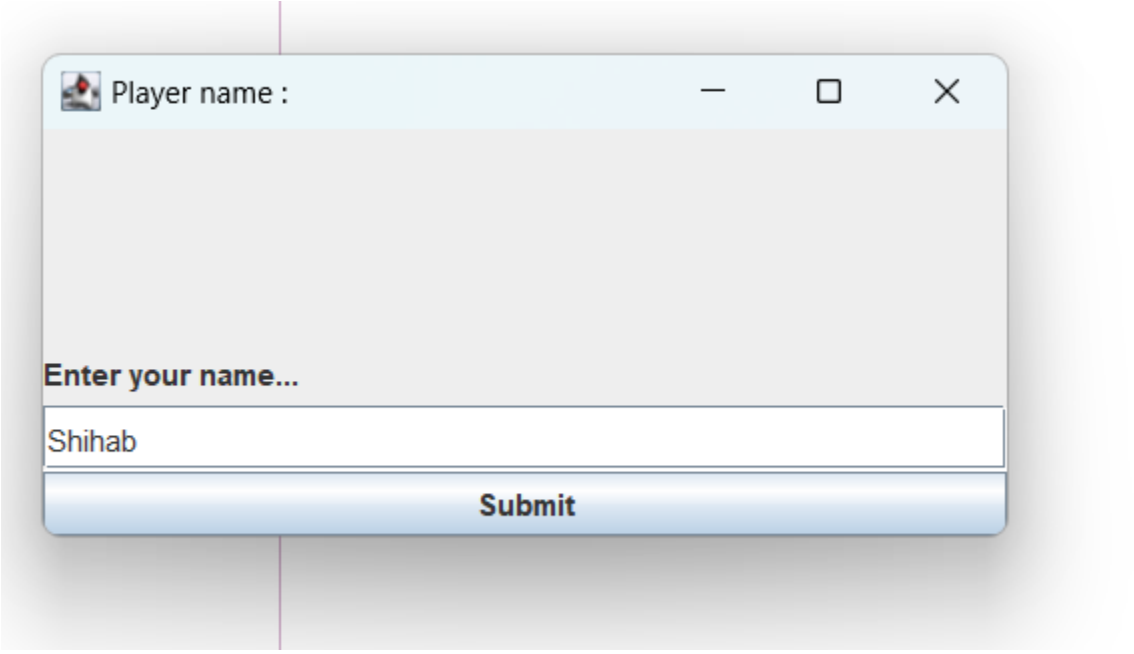

Figure (1)

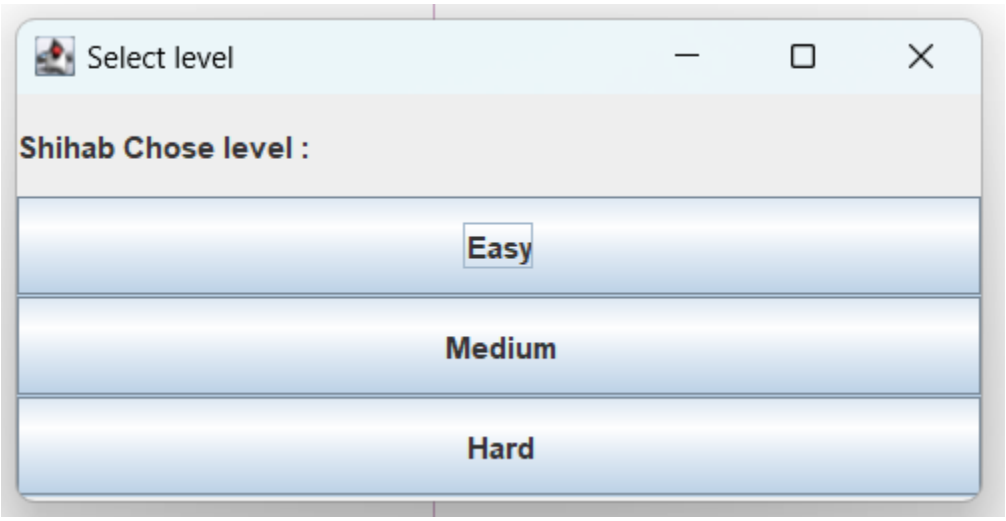

Figure (2)

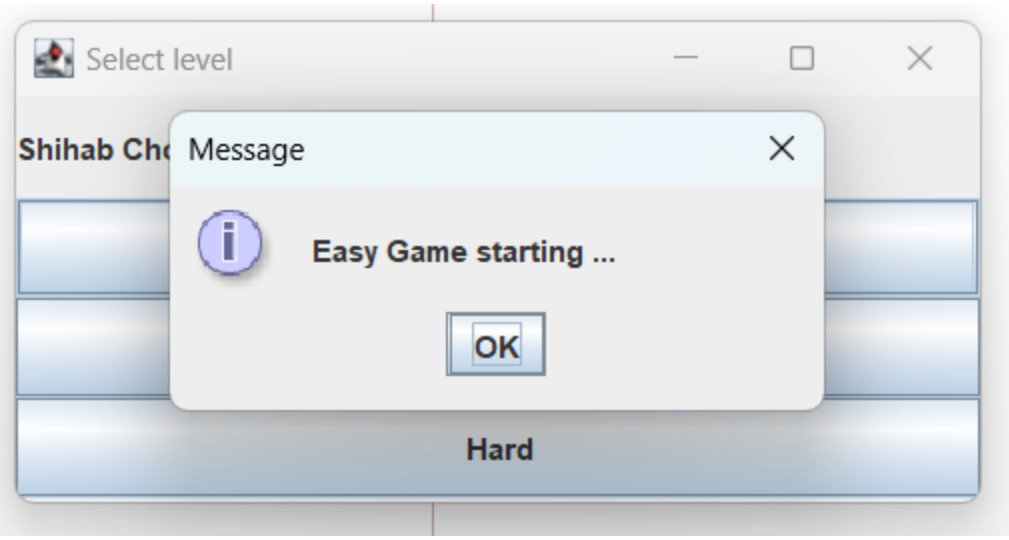

Figure (3)

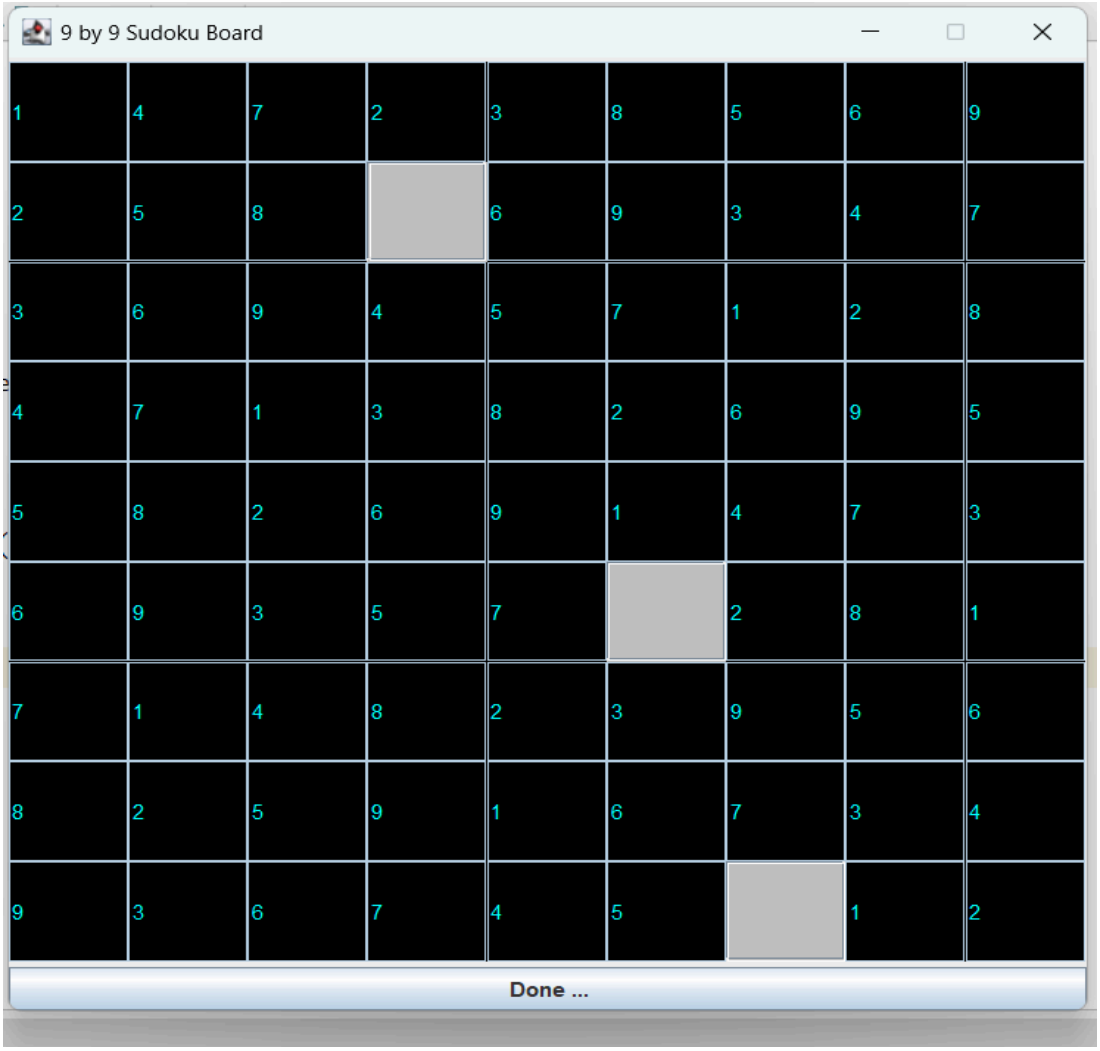

Figure (4)

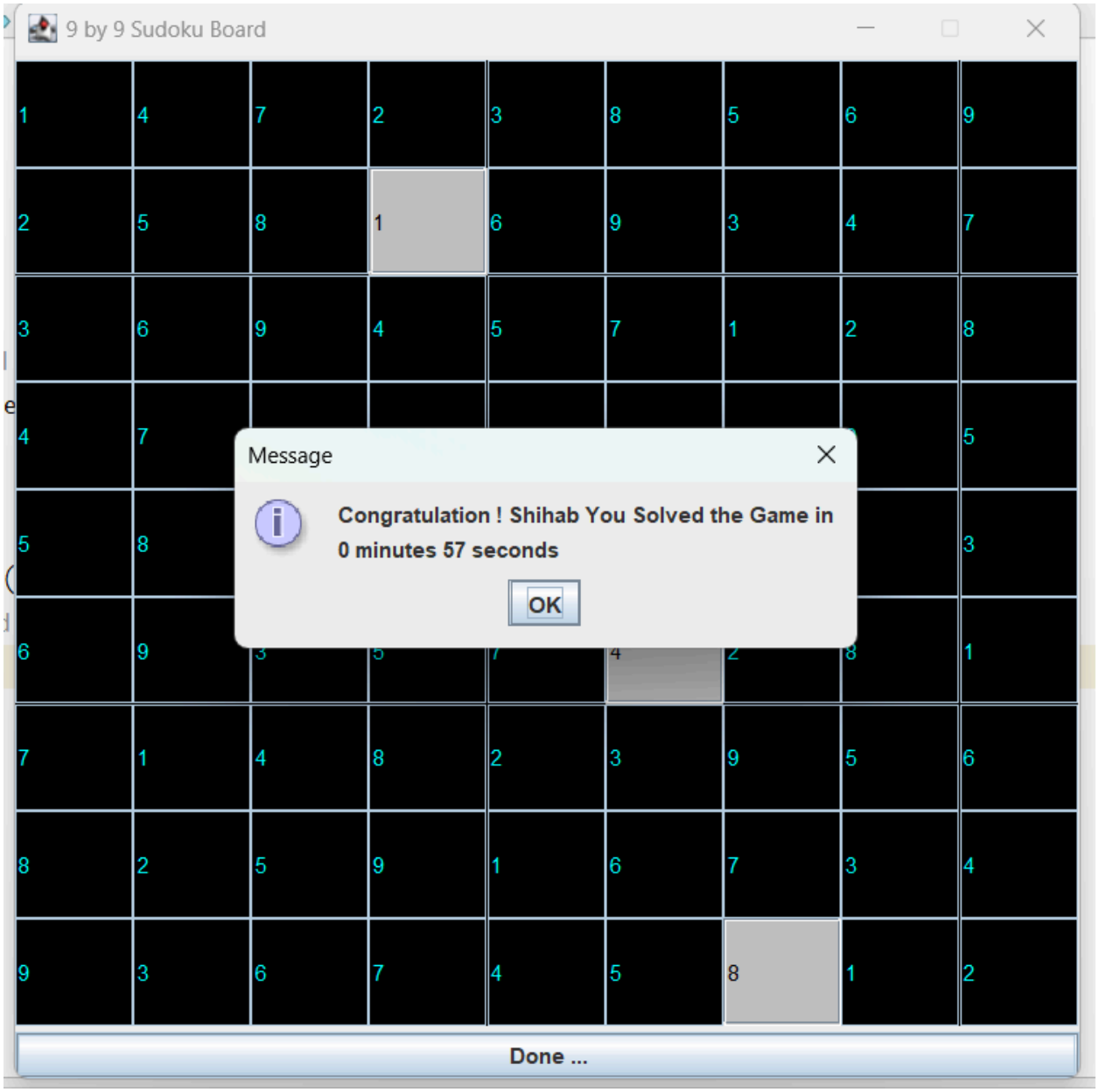

Figure (5)

3.2 Figure Explanations :

After running the code we can see **Figure (1)** means the design inside **ResponseWindow()** method is activated . After a successful name submission the **LevelSelectionWindow(**receives string parameter **)** method activates . Then selecting any level option activates the 2nd class code where the SudokuGame() constructor runs where the Frame design is created . Sequentially the

constructor activates the **GenerateSudokuPuzzle(**receives integer parameter to set difficulty**)** method . Inside this method the **GenerateSolutionFully(**takes integer array to store complete solution in rows and columns**)** method runs and makes a fully solved sudoku board and stores it to compare later , then with the parameter it sets game levels and empty blocks for generating puzzles . This method also uses **isValid(**receives integer array, row,column and value**)** to determine valid values. The SudokuGame() constructor also uses the **CheckSolution()** method to announce the winner . This **printSolution()** method is created for testing and debug purposes so it is optional . **Figure (1)** to **Figure (3)** functionable for the first class and **Figure (4)** to **Figure (6)** is functional for the second class . After a game pressing ok will result in an exit from the program .

Chapter 4 .
(Discussion)

4.1 Analysis strength of the project :

With this simplest project covering almost each functional GUI functionality of java . The use of each keyword is transparent and understandable .How use of inheritance reduces repetition of code is clearly visible here . Default constructor modification can reduce code and is also visible in this project . With a limited code collection a perfect functional interface is created .

4.2 Weakness Analysis :

The code is complex for using recursion in the **GenerateSolutionFully()** method . This might result in stack overflow sometimes . The code is not optimized for dealing with any kind of exceptions. The sudoku board is only playable in a single 9 by 9 grid .

Chapter 5 .
(Conclusions)

5.1 Achievements from the project :

- Learned many concepts of java awt (abstract window toolkit)

- Helped to build and think complex logic and design
- Improved coding skill and situations dealing
- Improved error handling and debugging skill

5.2 Future improvements :

- The code can be optimized and reduced in size .
- There might be a menu bar to control levels .
- There might be a game history to see previous game data .
- Color correction and design positioning can be improved .
- Multiple variants of sudoku board can be added .

Chapter 6 .
(References)

6.1 Websites to further research on java OOP :

Link 1 : https://www.w3schools.com/java/java_oop.asp
Link 2 : https://www.javatpoint.com/java-swing
Link 3 : https://www.geeksforgeeks.org/introduction-to-java-swing/

Youtube playlists :
Link 1 : Java Swing Bangla Tutorial 1 : Java Swing course plan

Concept clearing :
AI helper link 1 : https://chat.openai.com/
Link 2 : https://gemini.google.com/app

6.2 Media resources :

Figure 1.1(a) :
https://en.wikipedia.org/wiki/Sudoku#/media/File:Sudoku_Puzzle_by_L2G-20050714_standardized_layout.svg

Figure 1.1(b) :
https://en.wikipedia.org/wiki/Sudoku#/media/File:Sudoku_Puzzle_by_L2G-20050714_solution_standardized_layout.svg