Summer Training / Summer PEP

On

**“Sudoku Visualizer, Competitive Coding and MongoDB”**

**A training report**

In partial fulfillment for the requirement of the award of the degree of

**“B.Tech CSE”**

Submitted to

**LOVELY PROFESSIONAL UNIVERSITY**

**PHAGWARA, PUNJAB.**



From 06/05/2024 to 07/19/2024

SUBMITTED BY

**Name of student: Mo Sahil**

**Registration Number: 12216811**

**Student Declaration**

**To whom so ever it may concern**

I, **Mo Sahil**, **12216811**, hereby declare that the work done by me on **“SUDOKU VISUALIZER”** from July 2024, Year, is a record of original work for the partial fulfillment of the requirements for the award of the degree, **Bachelor of Technology in Computer Science and Engineering**.

**Mo Sahil (12216811)**

**A close up of a letter

Description automatically generated**

**Signature of the Student**

**Date: 31-08-2024**

**CERTIFICATE**

**MONGODB**

A certificate of completion with green and white text

Description automatically generated

**Link:** [**https://ti-user-certificates.s3.amazonaws.com/ae62dcd7-abdc-4e90-a570-83eccba49043/0d12115b-fdf4-42bd-a6ce-27471ce26e52-mo-sahil-13fbe836-554a-49d2-ba1f-70870f92dde5-certificate.pdf**](https://ti-user-certificates.s3.amazonaws.com/ae62dcd7-abdc-4e90-a570-83eccba49043/0d12115b-fdf4-42bd-a6ce-27471ce26e52-mo-sahil-13fbe836-554a-49d2-ba1f-70870f92dde5-certificate.pdf)

**Competitive Programming – Hitbullseye**



**Link:** [**https://drive.google.com/file/d/1s4xykR\_CLXscsFBs6-jvoJHK\_iKc91cD/view**](https://drive.google.com/file/d/1s4xykR_CLXscsFBs6-jvoJHK_iKc91cD/view)

**ACKNOWLEDGEMENT**

I, **Mo Sahil**, bearing Registration Number **12216811**, solemnly declare that the information presented in this project is complete, accurate, and based on the best of my knowledge, belief, and information.

Furthermore, I commit to making available the recordings of the verification sessions promptly upon request by the relevant university authorities. If any of the aforementioned information is discovered to be incorrect or misleading, I am aware that disciplinary action may be initiated against me by the university.

**Name: Mo Sahil**

**Reg. No: 12216811**

**Date: 31-08-2024**

**Chapter 1**

**Understanding the Basics of Sudoku Solving**

**1.1 Introduction to Sudoku**

Sudoku is a logic-based, combinatorial number-placement puzzle. The goal is to fill a 9x9 grid with digits so that each column, each row, and each of the nine 3x3 sub grids contain all the digits from 1 to 9. Although Sudoku puzzles appear complex, they can be solved using various algorithms, which we will explore in this chapter.

**1.1.1 Core Concepts**

1. **Backtracking Algorithm:**

* Backtracking is a general algorithm for finding solutions to problems by incrementally building candidates and abandoning candidates ("backtracking") as soon as it determines they cannot possibly lead to a valid solution.
* In Sudoku, backtracking tries placing a number in an empty cell, checks if it leads to a solution, and backtracks if it does not.

1. **Validation of Sudoku Rules:**

A valid Sudoku grid must meet three criteria:

* **Row Validity**: No repeated digits in any row.
* **Column Validity**: No repeated digits in any column.
* **Sub grid Validity**: No repeated digits in any 3x3 sub grid.

**1.1.2 Implementing the Sudoku Solver**

Let's start by looking at how we can implement a Sudoku solver using Java. This implementation will lay the groundwork for building a graphical interface in later chapters.

* Defining the Initial Sudoku Board

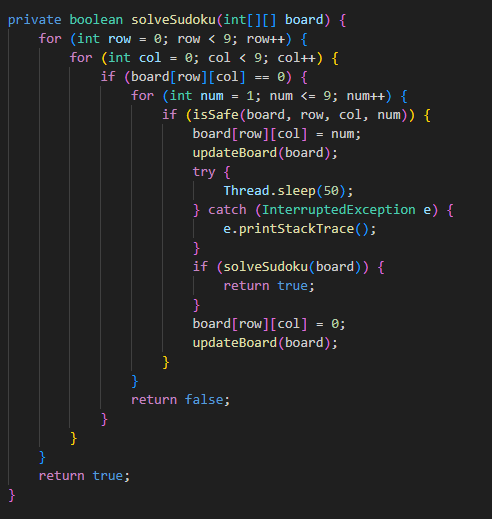
**A screenshot of a computer screen

Description automatically generated**

* The `startingBoard` array represents a partially completed Sudoku puzzle where `0` indicates an empty cell.

**1.1.3 The Sudoku Solving Algorithm**

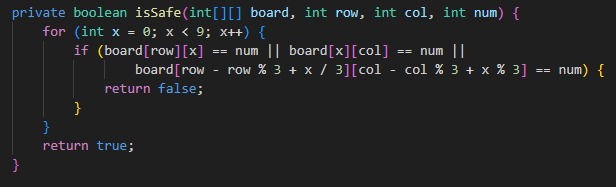
The core of the Sudoku solver is the `solveSudoku` method, which implements the backtracking algorithm.

****

* **Recursive Backtracking:** The algorithm tries each possible number (1-9) in every empty cell. It checks if placing a number in a particular cell is safe using the `isSafe` method. If a number is placed and leads to a valid solution, the method returns true; otherwise, it backtracks by resetting the cell to `0`.

**1.1.4 Checking Validity of Moves**

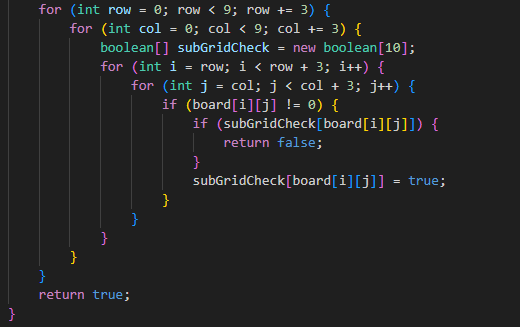
The `isSafe` method checks whether placing a number in a specific cell violates the Sudoku rules.

****

* **Row Check:** Ensures the number isn't already in the row.
* **Column Check:** Ensures the number isn't already in the column.
* **Sub grid Check:** Ensures the number isn't already in the 3x3 sub grid.

**1.1.5 Validating the Sudoku Grid**

Before attempting to solve the puzzle, it's important to validate the input to ensure it doesn't violate the basic Sudoku rules.

** A computer screen shot of text

Description automatically generated**

* **Row and Column Checks:** Ensure no duplicate numbers in any row or column.
* **Sub grid Check:** Ensures no duplicate numbers in any 3x3 sub grid.

**1.1.6 Conclusion**

In this chapter, we introduced the basic concepts needed to solve a Sudoku puzzle using Java. We explored how to implement a backtracking algorithm, validate the Sudoku grid, and check if a move is safe. These foundational concepts are essential as we move toward building a more interactive Sudoku solver with a graphical user interface in the next chapter.

**Chapter 2**

**Graphical User Interfaces in Java**

**2.1 Introduction to GUIs**

Graphical User Interfaces (GUIs) are a crucial part of modern applications, allowing users to interact with software through visual elements like buttons, text fields, and labels. In Java, the Swing library provides a powerful set of components for building GUIs.

**2.2 Key Concepts in Java GUIs**

1. **Frames and Panels**
   * **JFrame**: The top-level container that represents a window. It's the main container in which other components are added.
   * **JPanel:** A generic container often used to organize components within a JFrame.
2. **Layout Managers:** Layout managers control how components are arranged within a container. Common layouts include `BorderLayout`, `GridLayout`, and `FlowLayout`.
3. **Event Handling:** GUIs respond to user actions (e.g., button clicks) through event listeners. Java provides interfaces like `ActionListener` to handle these events.

**2.3 Building a Sudoku Solver GUI**

Let's walk through the implementation of a Sudoku Solver with a GUI in Java, highlighting the concepts mentioned above.

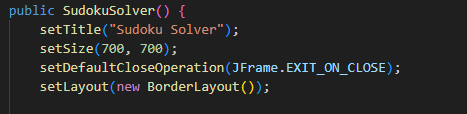
**2.3.1 Setting Up the JFrame**

**A screen shot of a computer program

Description automatically generated**

Here, we define the main components:

* A 9x9 grid of `JTextField` for entering numbers.
* Buttons for solving and clearing the grid.
* A label to show messages to the user.

****

* We set up the frame's title, size, and close operation.
* We use a `BorderLayout` to organize components in the frame.

**2.3.2 Creating the Grid of Text Fields**

**A black background with white text

Description automatically generated**

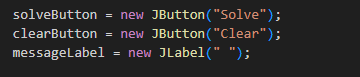
* We create a `JPanel` with a `GridLayout` to hold the 9x9 grid.
* A custom font is applied to the text fields.

**A screen shot of a computer code

Description automatically generated**

* We loop through the 9x9 grid to initialize each `JTextField`.
* If the starting board contains a non-zero value, it's displayed in the corresponding text field.

**2.3.3 Adding Buttons and Label**

****

* Buttons and a label are created for interaction.

**A group of letters on a black background

Description automatically generated**

* We attach action listeners to the buttons to handle user actions.

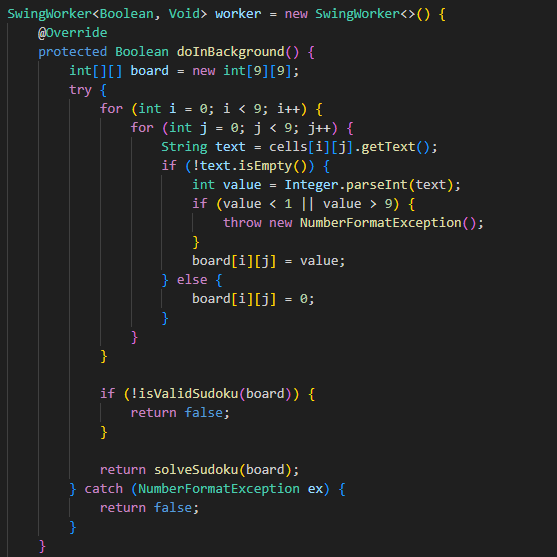
**2.4 Solving the Sudoku**

The Sudoku-solving logic is encapsulated in the `SolveButtonListener` class, which is triggered when the "Solve" button is clicked.

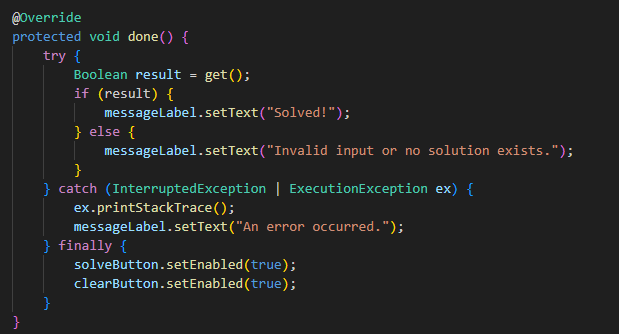
**A screen shot of a computer code

Description automatically generated**

* We disable the buttons while solving is in progress.

****

* The `doInBackground` method checks the user's input and attempts to solve the Sudoku using the `solveSudoku` method.

****

* Once the solving process is complete, the result is displayed, and the buttons are re-enabled.

**2.5 Conclusion**

This chapter introduced the basics of building GUIs in Java using the Swing library. We covered frames, panels, layout managers, event handling, and applied these concepts to create a functional Sudoku Solver. In the next chapter, we will dive deeper into optimizing the solving algorithm and explore more advanced Swing components.

**Chapter 3**

**Performance Evaluation**

**3.1** **Simulation Environment/ Simulation Procedure**

The simulation environment for the Sudoku Solving project was developed using Java, with a particular focus on the Java Swing library for the graphical user interface (GUI). The environment provides a user-friendly interface that enables users to interact with the Sudoku puzzle. The development tools and setup used in this project include:

* **Java Development Kit (JDK)**: Version 7 or higher.
* **Integrated Development Environment (IDE)**: Visual Studio Code was used as the IDE for this project.
* **Device Specifications:**
* **Brand**: HP
* **Model**: HP Elitebook 840 G3
* **Screen Size**: 14 inches
* **Colour**: Silver
* **SSD**: 256GB
* **CPU Model**: i5-6300U
* **RAM**: 8 GB
* **Operating System**: Windows 11 Pro

These specifications ensured a smooth development and testing process for the Sudoku solver, allowing for quick compilation, execution, and visualization of the results.

**3.2** **Results Analysis/Testing**

The Sudoku Solving project aims to provide a user-friendly interface for generating and solving Sudoku puzzles. It utilizes a 9x9 grid panel and incorporates two main buttons: "Solve," and "Clear". The "Solve" button triggers the solving algorithm, visually displaying the step-by-step solving process on the grid. It employs time delay visualization to enhance the user experience. The "Clear" button allows users to reset the grid, providing a clean slate for new puzzles or starting over. Together, these buttons create an interactive environment for users to generate, solve, and engage with Sudoku puzzles..

**3.2.1** **Result\_Case\_1**

This is the picture of the GUI of our project

A screenshot of a game

Description automatically generated

**3.2.2** **Result\_Case\_2**

**Solving of the Sudoku:**

This is the picture of the Solving of sudoku that is triggered by clicking the solve button. Solved cells numbers are highlighted in green.

A screenshot of a game

Description automatically generated

Figure 3.2: Live Solving Visualization

**3.2.3** **Result\_Case\_3**

**Clear Button:**

This is the picture of the sudoku grid after clicking clear button.

A screenshot of a computer

Description automatically generated

Figure 3.3: Clear Button Visualization

* 1. **Results Overall Discussion**

The Sudoku Solving project combines the functionality of generating new puzzles, solving them with a sophisticated algorithm, and resetting the grid as needed. The user-friendly interface makes it accessible to Sudoku enthusiasts of all skill levels.

* **Puzzle Generation**: User can generate a puzzle with any level of difficulty by just simply typing the numbers into the grid.
* **Solving Algorithm**: The backtracking algorithm used by the "Solve" button efficiently fills in the puzzle's empty cells, with a real-time visual display that highlights the solving process.
* **Grid Reset**: The "Clear" button offers flexibility by allowing users to start fresh at any point.

By combining these features, the Sudoku Solving project provides an interactive and engaging experience that appeals to both casual users and puzzle enthusiasts.

**3.3.1** **Complex Engineering Problem Discussion**

Several complex engineering problems across different categories.

1. **Depth of Knowledge Required:** The Sudoku Solving project requires a depthof knowledge in various areas. A strong understanding of Sudoku rules and logic is necessary, along with proficiency in algorithmic problem solving to implement the Sudoku solving algorithm effectively. Competence in a programming lan-guage, such as Python, Java, C++, or JavaScript, is essential for developing the project. Familiarity with data structures and grid representation is required to manipulate and store Sudoku puzzles. Basic knowledge of user interface design principles and development frameworks like HTML, CSS, JavaScript, Swing, or JavaFX is necessary to create an intuitive user interface. Understanding how to incorporate time delay and visualization techniques to demonstrate the solving process is important. Proficiency in testing and debugging is crucial for ensur-ing the correctness and reliability of the project. This broad range of knowledge ensures the successful development and implementation of the Sudoku Solving project.
2. **Range of Conflicting Requirements:** The Sudoku Solving project encompassesa range of conflicting requirements that must be carefully addressed. On one hand, the project aims to generate Sudoku puzzles that are challenging and di-verse, requiring a sophisticated algorithm for puzzle generation. On the other hand the solving algorithm must be efficient and capable of solving puzzles in a reasonable amount of time. The user interface should be intuitive and visually appealing while also providing clear feedback and visualizations of the solving process. Balancing these conflicting requirements requires careful consideration, such as finding the right balance between generating complex puzzles and en-suring solvability, optimizing the solving algorithm for speed without sacrificing accuracy, and designing a visually appealing and informative interface. Striking the right balance is crucial to create a Sudoku Solving project that offers enjoy-able and challenging puzzles while providing a seamless user experience.
3. **Depth of Analysis Required:** The Sudoku Solving project requires a depth ofanalysis in various areas. This includes analyzing the puzzle generation algorithm to ensure unique solutions and varying difficulty levels, assessing the correctness and efficiency of the solving algorithm, evaluating user interaction and interface usability, conducting performance analysis to optimize speed and responsiveness, analyzing error handling and validation mechanisms, assessing cross-platform compatibility, and evaluating overall usability and user experience. Thorough analysis in these areas is essential to refine and improve the project, ensuring it meets high standards of puzzle generation, solving accuracy, user interaction, performance, error handling, compatibility, and user satisfaction.
4. **Extent of Applicable Codes:** The Sudoku Solving project involves the imple-mentation of various codes. This includes code for puzzle generation to create Sudoku puzzles of different difficulty levels, code for the solving algorithm to find solutions using backtracking or other search algorithms, code for the user interface to provide an interactive grid, buttons, and visual feedback, code for validation and error handling to ensure the entered puzzles are valid and solvable, code for time delay and visualization techniques to demonstrate the solving pro-cess gradually, code for performance optimization to enhance efficiency, and code for testing to validate the implemented functionalities. The extent of applicable codes for the project spans puzzle generation, solving, user interface, validation, time delay, visualization, performance optimization, and testing, all of which con-tribute to the successful implementation and functionality of the Sudoku Solving project.

**Chapter 4**

**Conclusion**

**4.1** **Discussion**

The Sudoku Solving project utilizes a user-friendly interface with two buttons: "Solve" and "Clear. The "Solve" button applies an algorithm to solve the puzzle step-by-step, visually displaying the progress. The "Clear" button resets the grid. Overall, the project offers an enjoyable Sudoku-solving experience with puzzle generation, solving algorithm implementation, and user interaction.

**4.2** **Limitations**

limitations include:

* **Complexity of Puzzle Generation:** Generating Sudoku puzzles with unique so-lutions can be a computationally intensive task, especially when aiming to provide puzzles of varying difficulty levels. As a result, the puzzle generation process may take longer for more challenging puzzles, which can impact user experience.
* **Inability to Solve Invalid Puzzles:** The solving algorithm assumes that the in-put puzzle is a valid Sudoku configuration. If an invalid or unsolvable puzzle is provided, the algorithm may run indefinitely or produce incorrect results. It is important to ensure that the project includes appropriate error handling and validation mechanisms to prevent such scenarios.
* **Lack of Advanced Solving Techniques:** The current implementation of thesolving algorithm relies solely on backtracking. While backtracking is an effec-tive method for solving Sudoku puzzles, it may not utilize more advanced solving techniques like constraint propagation or advanced heuristics. Incorporating ad-ditional solving techniques could enhance the algorithm’s efficiency and provide more optimized solutions.

**4.3** **Scope of Future Work**

Scope of Future Work for the Sudoku Solving Project:

* **Advanced Solving Techniques:** The project can be expanded to incorporatemore advanced solving techniques, such as constraint propagation, hidden sin-gles, naked pairs, and other advanced strategies used in Sudoku solving. Imple-menting these techniques would enhance the solving algorithm’s efficiency and provide faster and more optimized solutions.
* **Puzzle Validation and Error Detection:** Enhancing the project to include puzzlevalidation and error detection mechanisms would help identify invalid or unsolv-able puzzles at the input stage. This would prevent the solving algorithm from running indefinitely on incorrect puzzles and provide users with immediate feed-back on the validity of their inputs.
* **Mobile and Cross-Platform Compatibility:** Adapting the project for mobiledevices and ensuring cross-platform compatibility would expand its reach and make it accessible to a wider audience. This could involve developing a respon-sive user interface or creating dedicated mobile applications for iOS and Android platforms.
* **Additional Grid Sizes and Variations:** Extending the project to support dif-ferent grid sizes and variations of Sudoku, such as 4x4, 16x16, or irregularly shaped grids, would provide users with more puzzle options and challenges. Im-plementing additional grid sizes would require modifying the solving algorithm and adapting the user interface to accommodate the changes.
* **Multiplayer and Online Challenges:** Implementing multiplayer functionalitywould enable users to compete against each other in solving Sudoku puzzles. This could involve real-time puzzle synchronization, leaderboards, and timed challenges, adding a social and competitive aspect to the project.