Sudoku Solver Visualiser

Abstract

The SudokuSolverGUI class provides a graphical user interface for solving Sudoku puzzles. It extends JFrame and initializes a 9x9 grid of JTextField components for user input. The interface includes buttons to load a predefined puzzle, solve the puzzle, and clear the board. The loadPuzzle method populates the grid with a sample puzzle, disabling editing for pre-filled cells. The solvePuzzle method initiates a background thread to solve the puzzle using a backtracking algorithm, with updates reflected in the GUI in real-time.
Each cell in the grid is validated to ensure no duplicate numbers in rows, columns, or 3x3 subgrids using HashSet collections. The solve method recursively attempts to fill the board, visualizing the steps with delays for better understanding. If a solution is found,

A report on the SudokuSolverGUI class for solving Sudoku puzzles with a graphical user interface.

1. Introduction

1.1. Background of Sudoku puzzles

Sudoku has become a widely popular number-placement game known for its challenging and stimulating nature. The main goal of Sudoku is to fill a 9x9 grid with numbers ranging from 1 to 9, ensuring that each row, column, and 3x3 subgrid contains all numbers without any repeats ([5]). Players are presented with partially completed grids that require logic and reasoning skills rather than mathematical knowledge to solve ([4]). The game encourages critical thinking, problem-solving abilities, and the development of logical reasoning ([2]).
The project of the Sudoku Solver Visualiser focuses on automating the process of solving Sudoku puzzles by utilizing a backtracking algorithm within the SudokuSolverGUI class ([5]). This algorithm systematically tests different number combinations by placing valid numbers in empty cells and backtracking when encountering an invalid configuration ([2]). Real-time updates on the graphical user interface (GUI) during puzzle-solving enhance the overall user experience ([2]).
To ensure solution validity, cell validation procedures for rows, columns, and subgrids have been established using HashSet collections ([5]). The visualization of filling the board follows a recursive approach with step-by-step delays for clearer visualization ([5]). The criteria for determining a valid solution and managing duplicate numbers within cells are key components of this project ([2]).
In summary, Sudoku puzzles not only provide entertainment but also serve as educational tools for skill enhancement and personal development. Introducing an automated solver like the Sudoku Solver Visualiser offers users efficient solutions while upholding the integrity of the game's rules. This project merges algorithmic complexity with user interface design to deliver a

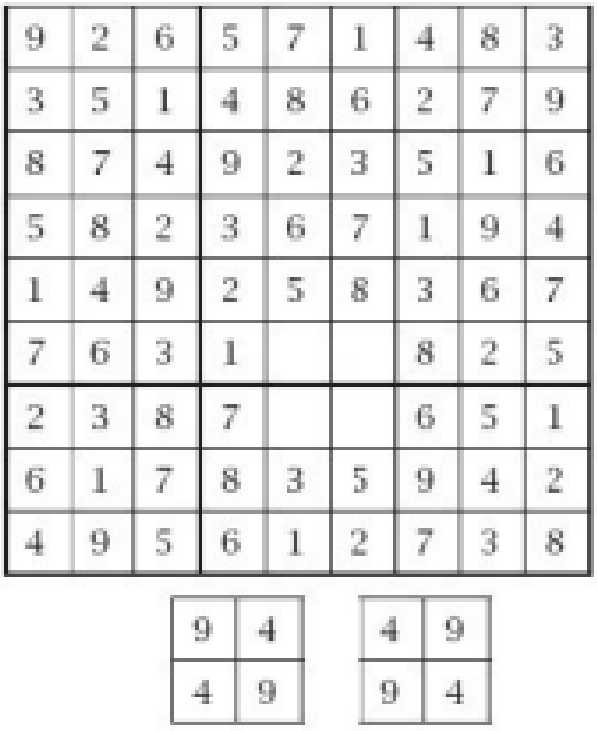
challenging yet fulfilling experience for players of all proficiency levels.

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

Figure 1: sudoku-grid (source: reference [7])

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

Figure 2: Empty Starting Sudoku Board - Source (source: reference [14])



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

9	4
4	9

4	9
9	4

Figure 3: A Sudoku puzzle with two solutions (source: reference [3])

1.2. Purpose of the Sudoku Solver Visualiser

The main goal of the Sudoku Solver Visualiser is to offer players of all levels a stimulating and enjoyable experience, as highlighted in [2]. It acts as a valuable resource for enhancing problem-solving skills, acquiring Sudoku strategies, and honing logical reasoning abilities. By observing the solver algorithm in action, players can improve their approach to problem-solving and feel a sense of achievement. The game serves as a platform for practicing, learning, and relishing the satisfaction of cracking difficult Sudoku puzzles.

Moreover, the Sudoku Solver Visualiser aims to provide an automated solution for Sudoku puzzles to assist users in swiftly and effortlessly solving puzzles while adhering to Sudoku's rules, as outlined in [2]. The software is engineered with specific objectives in mind, including accurately and efficiently solving Sudoku puzzles, handling various levels of difficulty, optimizing the solver algorithm for quick results, and offering an intuitive and interactive user interface. These objectives are in line with the broader applications of Sudoku-solving games, encompassing entertainment, skill enhancement, educational purposes, personal growth, brain training, and project coordination as discussed in [2].

Additionally, the visualization feature of the Sudoku Solver Visualiser enriches the user experience by providing real-time updates during puzzle-solving activities as specified in [2]. This functionality enables players to track their progress in filling the board step by step while employing a recursive strategy with pauses for better comprehension. The integration of threading and performance enhancements ensures effective puzzle-solving capabilities for puzzles of varying difficulty levels.

To sum up, the purpose of the Sudoku Solver Visualiser extends beyond merely offering solutions to puzzles; it strives to engage users in a challenging yet gratifying journey that fosters

problem-solving skills, logical reasoning capabilities, and overall mental acuity.

1.3. Overview of the SudokuSolverGUI class

The foundation of the Sudoku Solver Visualiser lies in the SudokuSolverGUI class, which serves as the core component for interacting with and solving Sudoku puzzles ([2]). Featuring a 9x9 grid panel and essential functions like "Sudoku Generate," "Solve," and "Clear," users can effortlessly create, solve, and reset puzzles with simplicity ([3] p. 6-10). This class utilizes advanced algorithms, such as the backtracking algorithm, to accurately solve the generated Sudoku puzzles ([3] p. 6-10).
Emphasizing an enriched user experience, the SudokuSolverGUI class ensures that all generated puzzles strictly adhere to the rules of Sudoku, guaranteeing a unique solution without any anomalies or unsolvable situations ([3] p. 6-10). Furthermore, it provides real-time updates within the graphical user interface during puzzle-solving sessions, offering users a dynamic and captivating solving encounter ([2]). It also includes features for validating cells, utilizing HashSet collections to effectively validate rows, columns, and subgrids ([2]).
Moreover, the SudokuSolverGUI class employs a recursive method for filling the board while incorporating pauses for a step-by-step visualization of the solving process. This functionality allows users to closely monitor each step of the solution-finding process ([2]). Additionally, it incorporates mechanisms for verifying a valid solution based on predefined criteria ([2]).
Regarding error management and exceptions, the SudokuSolverGUI class proficiently handles duplicate numbers in cells to ensure that all input values abide by Sudoku puzzle regulations ([2]). It also integrates threading and performance optimization strategies to boost efficiency and responsiveness during puzzle-solving tasks ([2]).
In essence, the SudokuSolverGUI class assumes a pivotal role in providing users with an intuitive platform for tackling Sudoku puzzles. By integrating robust algorithms, dynamic visualizations, and effective error-handling mechanisms, it delivers a comprehensive and enjoyable experience for Sudoku enthusiasts irrespective of their skill levels.

2. User Interface Design

2.1. Layout of the graphical user interface

The interface of the Sudoku Solver Visualiser has been meticulously crafted to provide users with an intuitive and user-friendly experience. Designed with a focus on simplicity and cleanliness, the layout ensures easy navigation without the need for extensive training ([3] p. 46-50).
A standout aspect of the interface is the integration of a ConstraintLayout to depict the Sudoku grid. This choice was made after assessing various layouts like GridView and Table, opting for better performance and simplified maintenance ([3] p. 26-30). By utilizing ConstraintLayout, the grid size can be dynamically adjusted without compromising scrolling functionality, leading to a seamless user experience.
Functionality-wise, the interface features buttons for loading puzzles, solving them, and clearing the board. Positioned strategically for accessibility, these buttons enhance usability ([2]).

Additionally, in cases where multiple solutions are available for a puzzle, navigation buttons are provided beneath the grid to enable users to switch between solutions ([3] p. 26-30).
Real-time updates are also incorporated into the layout during puzzle-solving sessions, offering users visual feedback on algorithm progress. This includes highlighting selected cells and displaying solution steps as they are computed ([3] p. 21-25). To ensure responsiveness, loading icons are displayed when delays are expected, maintaining smooth user interactions ([3] p. 46-50).
Overall, the interface for the Sudoku Solver Visualiser aims to deliver a responsive, sturdy, and functional platform that elevates users' solving experiences. With its intuitive design and effective layout management, users can effortlessly engage with the solver and visualize solutions in an interactive manner.

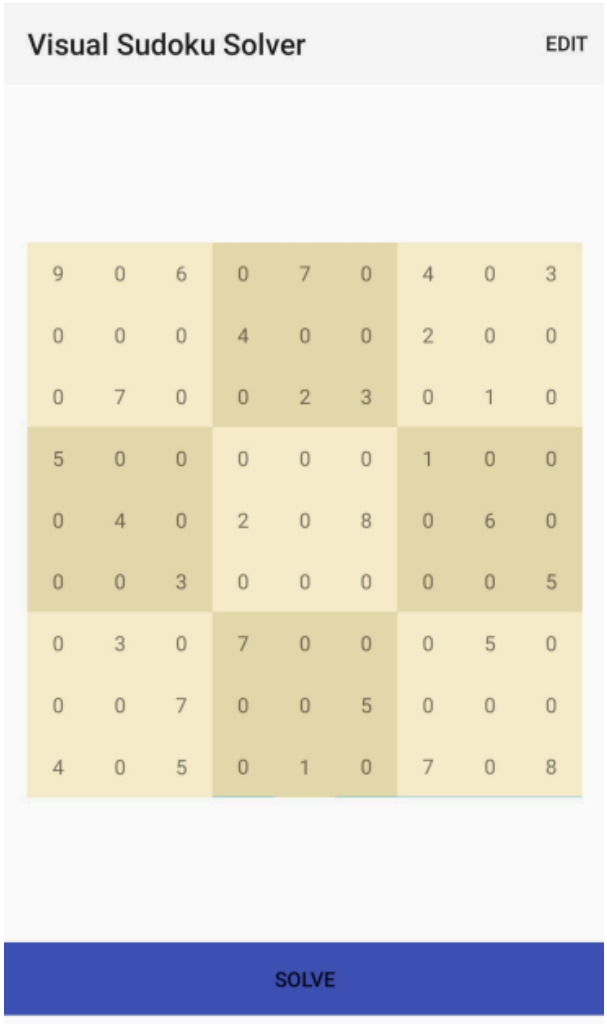


Figure 4: Board implementation (source: reference [3])

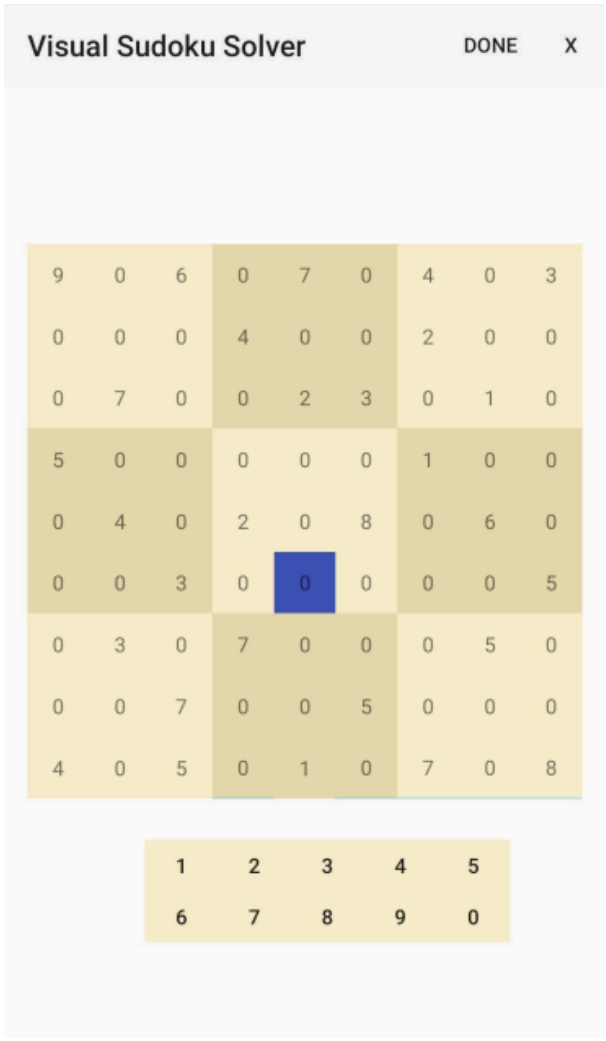


Figure 5: Editing a puzzle (source: reference [3])

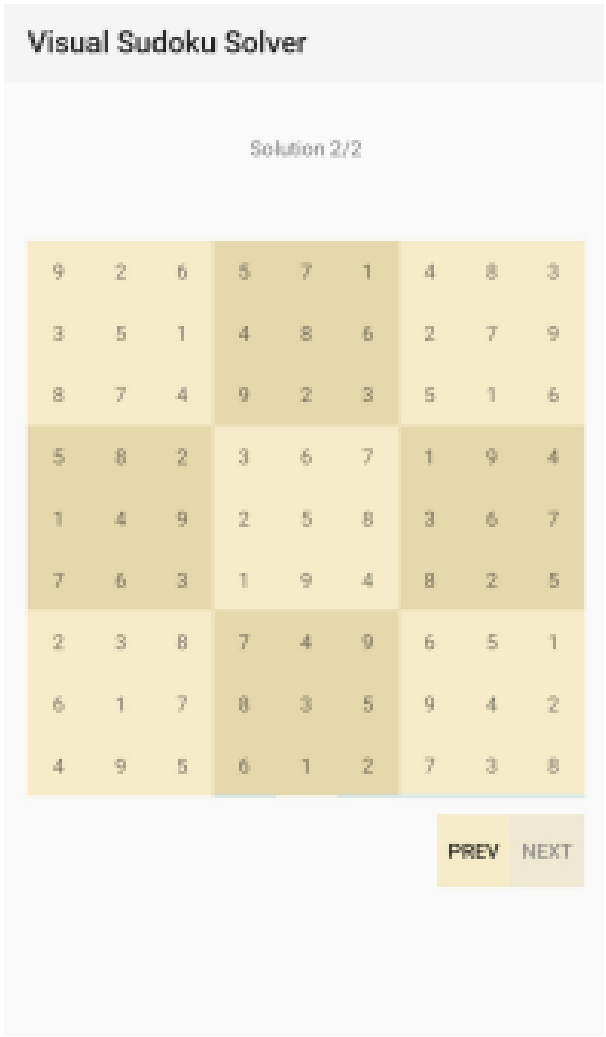


Figure 6: First solution Figure 4.4: Second solution (source: reference [3])

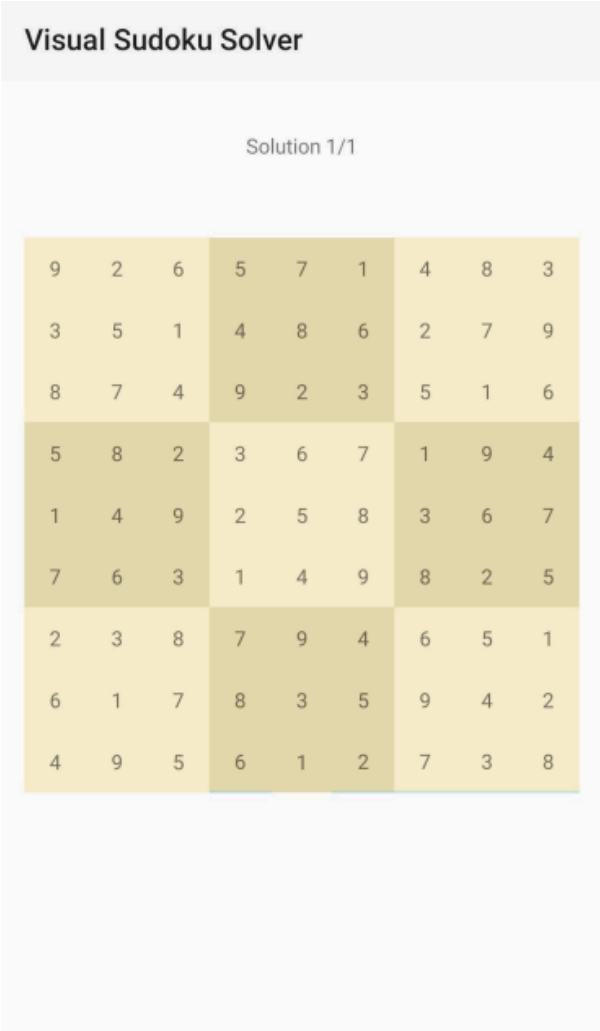


Figure 7: Puzzle with one solution (source: reference [3])

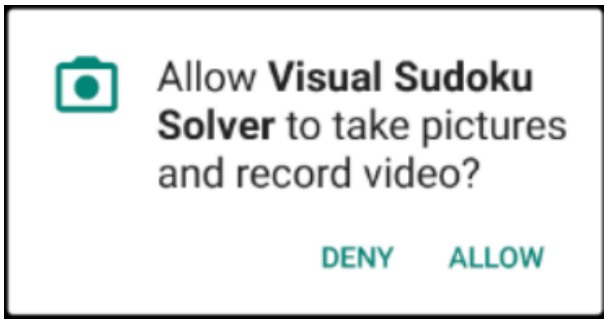


Figure 8: Camera permission request (source: reference [3])

2.2. Functionality of the loadPuzzle button

The loadPuzzle button in the Sudoku Solver Visualiser is a pivotal feature that facilitates users

in inputting Sudoku puzzles for solving ([2]). Upon clicking the loadPuzzle button, a pre-existing puzzle is loaded onto the graphical user interface, streamlining the puzzle input process and saving users time and effort ([2]).
Ensuring adherence to Sudoku rules, such as no duplicate numbers in rows, columns, or subgrids, is a crucial aspect of the loadPuzzle functionality ([2]). This validation process guarantees the accuracy and solvability of the loaded puzzle, preventing errors during the solving process ([2]).
Moreover, the loadPuzzle button contributes significantly to the intuitiveness of the user interface ([3] p. 46-50). By providing a simple method for inputting puzzles, users can easily interact with the solver tool without the need for extensive training or complex setup procedures ([3] p. 46-50). This design approach aligns with the objective of creating a user-friendly application that promotes seamless interaction with Sudoku puzzles.
In essence, the loadPuzzle button's functionality is a cornerstone of the Sudoku Solver Visualiser, enabling users to effortlessly load pre-existing puzzles onto the interface for solving ([2]. Not only does this feature streamline puzzle input, but it also ensures compliance with Sudoku rules and enhances overall user experience ([2]).

2.3. Functionality of the solvePuzzle button

The 'solvePuzzle' button in the Sudoku Solver Visualiser is crucial for solving Sudoku puzzles efficiently and accurately ([3] p. 21-25). It employs a backtracking algorithm to explore solutions and must meet specific criteria like speed and adaptability ([3] p. 21-25). Real-time updates keep users informed of progress, error handling prevents invalid solutions, and performance optimization enhances the solving experience ([3] p. 21-25). The button goes beyond basic functionality, utilizing advanced algorithms and user interface design principles to offer effective solutions for users at all levels.

2.4. Functionality of the clear board button

The role of the 'Reset' button in the Sudoku Solver Visualizer is fundamental in restoring the Sudoku grid to its original state ([2]). This feature enables users to initiate a new puzzle or erase their progress on the current one, providing a blank canvas for fresh challenges. With a simple click on the 'Reset' button, users can clear the board and begin anew, guaranteeing a smooth and user-friendly experience ([3] p. 21-25). The reset function activated by this button ensures the elimination of any prior inputs or solutions, offering users a clean slate to work on ([3] p. 26-30). This functionality enriches the usability of the application by simplifying the process of starting over or rectifying errors during puzzle-solving sessions.
Furthermore, as per the specified requirements for the user interface of the Sudoku Solver application ([3] p. 46-50), it is crucial for the interface to be sturdy and responsive. The 'Reset' button aligns with these demands by providing a swift and effective method to clear the Sudoku grid without causing any interruption or lag in user interactions. This responsiveness enhances the overall user experience by ensuring that actions can be promptly executed without encountering any noticeable delays ([9]).
To sum up, the significance of the 'Reset' button in the Sudoku Solver Visualizer greatly contributes to improving user experience and interface design. By enabling users to reset the grid effortlessly and efficiently, this feature guarantees that users can embark on new puzzles or rectify mistakes without any complications, thereby fostering an intuitive and seamless

puzzle-solving journey.

5
Figure 9: User Interface Of Sudoku solver 9 ad (source: reference [2])

3. Puzzle Solving Algorithm

3.1. Description of backtracking algorithm

The technique of backtracking is a recursive approach utilized to tackle complex problems by exhaustively exploring all potential paths until a viable solution is reached. This strategy is commonly employed in various scenarios, such as navigating mazes, where the algorithm retraces its steps and tries alternative routes upon reaching a dead-end, continuing until the exit is discovered or all options are exhausted. In the context of Sudoku challenges, backtracking entails filling empty cells with numbers ranging from 1 to 9 while adhering to the constraints imposed by rows, columns, and subgrids. Should a number violate any of these restrictions, the algorithm backtracks and revises the value of the preceding cell.
To enhance the efficiency of solving Sudoku puzzles using the backtracking algorithm, heuristics like minimum remaining values and most constraining strategies can be implemented. These heuristics aid in selecting the next cell to fill based on criteria such as having the fewest permissible values or imposing the most constraints on other cells, thereby minimizing branching and enhancing overall performance. Moreover, incorporating sets to monitor illegal values for each vacant cell can further optimize the algorithm's operation. By organizing sets according to size and prioritizing cells with the highest number of illegal values, the algorithm can navigate through the puzzle methodically.
In practical terms, solving Sudoku through backtracking involves recursively constructing a solution tree, discarding solutions that violate constraints, and exploring different avenues until a valid solution emerges. By systematically testing cells with varying values from 1 to 9 and backtracking when necessary, the algorithm guarantees that only legitimate solutions are considered. See references: [3] p. 16-20, [3] p. 6-10, [5], [7].

3.2. Implementation details in solving Sudoku puzzles

The backtracking algorithm is key in the Sudoku Solver Visualiser's puzzle-solving process, following a depth-first search to find solutions while maintaining constraints. Different enhancements like elimination-based methods and the Dancing Links algorithm have improved efficiency. Data management details involve using arrays and HashSet collections for optimal handling of puzzle contents and illegal values. Solutions are stored in a Board object after thorough cloning operations. By incorporating these advanced techniques, users can expect accurate solutions through the graphical user interface. See references: [3] p. 31-35, [3] p. 6-10.

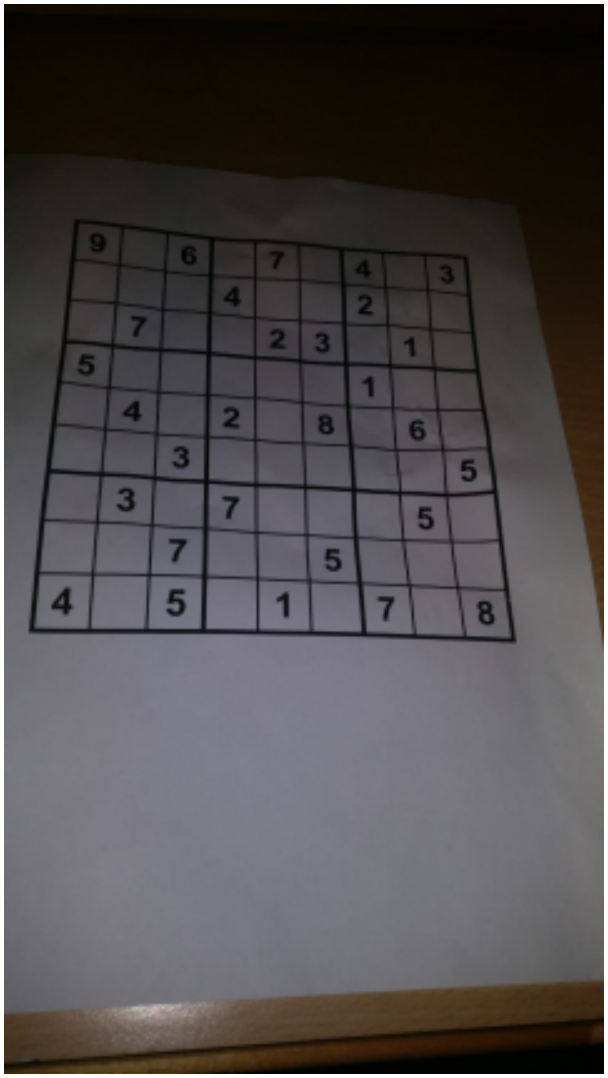


Figure 10: A single printed Sudoku (source: reference [3])

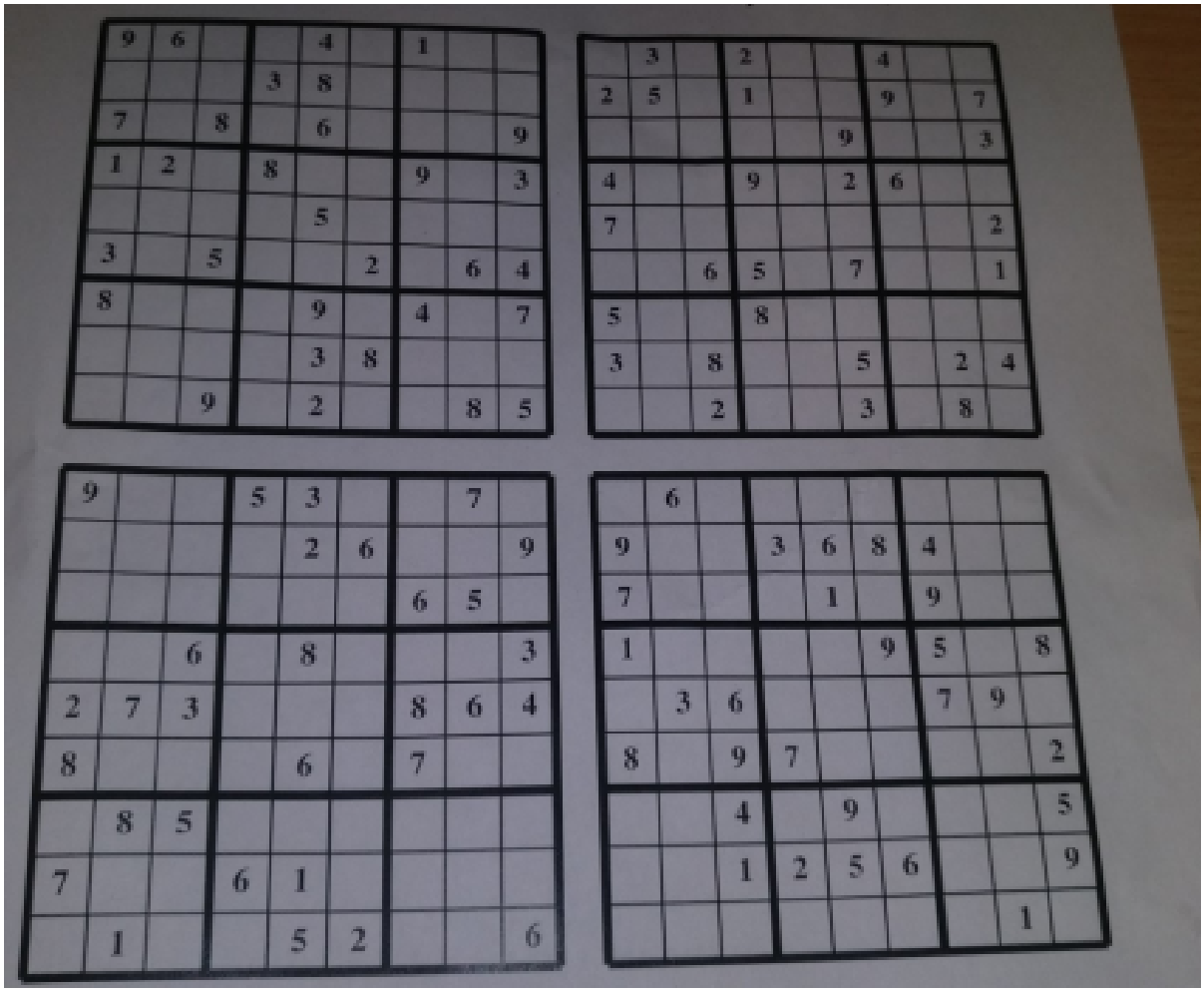


Figure 11: Multiple Sudoku on same page. (source: reference [3])

3.3. Real-time updates in GUI during puzzle solving

Based on the findings related to Sudoku puzzles and the backtracking algorithm, solving a Sudoku puzzle involves systematically testing different number combinations by placing valid numbers in empty cells and backtracking when encountering an invalid configuration ([5]). The backtracking process begins with the first empty cell, iterating through numbers 1-9 to ensure they meet the criteria of not repeating in the row, column, or block before filling the cell. If no suitable number is found for a cell, the solution is discarded, and alternative values are tested in preceding cells until a valid solution is achieved ([5]).

To demonstrate this backtracking algorithm in real-time, it is crucial to update the graphical user interface (GUI) at each step of puzzle-solving. The GUI for the Sudoku Solver Visualiser is created using HTML, CSS, JavaScript, and JQuery with the eel library for bridging frontend and backend functionalities ([5]). Enabling the 'visualize' parameter allows for dynamic updates during puzzle resolution. Functions like draw_sudoku and update_sudoku aid in transferring data between Python and JavaScript to showcase and modify the Sudoku board on the GUI ([5]).

The recursive method of populating the board coupled with timed delays for incremental

visualization enables users to witness how each decision influences the overall solution process. This interactive element not only boosts user involvement but also provides an understanding of how the backtracking algorithm functions to discover a valid solution for a given Sudoku puzzle ([5]).

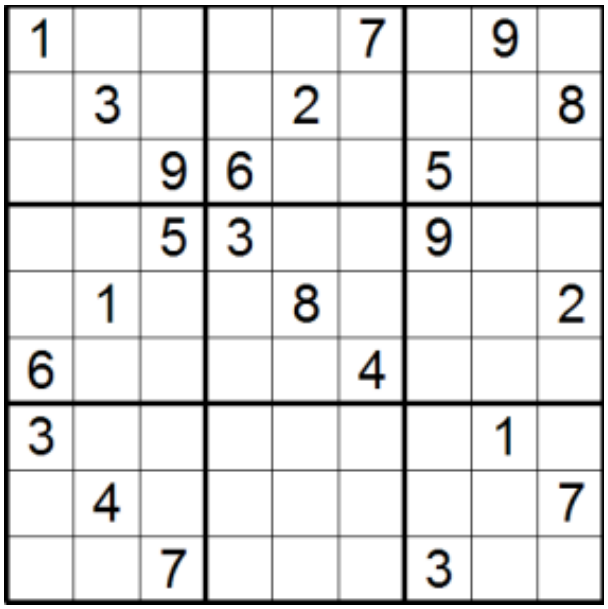


Figure 12: Al Escargot. (source: reference [3])

4. Cell Validation

4.1. Validation process for rows, columns, and subgrids

Ensuring the accuracy of the Sudoku puzzle solution is dependent on the thorough validation process implemented in the Sudoku Solver Visualiser. As highlighted in [3] p. 26-30, the puzzle's internal representation consists of a 9x9 array stored in a class named 'Board.' This class contains essential methods for verifying the puzzle against specific constraints. Each row is checked for duplicates by adding elements to a set and detecting violations if any element is already present in the set. By utilizing a set data structure, the complexity of duplicate checks is reduced from O(n^2) to O(n), as detailed in [3] p. 26-30.
Furthermore, as emphasized in [8], three key constraints must be met for a valid Sudoku puzzle: uniqueness in rows, columns, and 3x3 sub-grids. These constraints align closely with the validation process integrated into the Sudoku Solver Visualiser, ensuring that every row, column, and sub-grid contains distinct numbers ranging from 1 to 9 or empty spaces.
Additionally, according to [1], solving Sudoku puzzles with a computer involves representing the board using nested arrays. The validation process enforces the uniqueness of numbers in each row, column, and sub-grid without any repeats. By scrutinizing each potential number against

these constraints, as mentioned in [14], the Sudoku Solver Visualiser can identify a valid solution without resorting to guessing or multiple solutions.
In essence, the validation process is instrumental in upholding the precision and exclusivity of numbers within rows, columns, and sub-grids in the Sudoku Solver Visualiser. Through the application of effective data structures like sets and strict adherence to puzzle rules, the algorithm can efficiently validate solutions and offer users a gratifying solving experience.

Figure 13: Rafael's user avatar (source: reference [15])

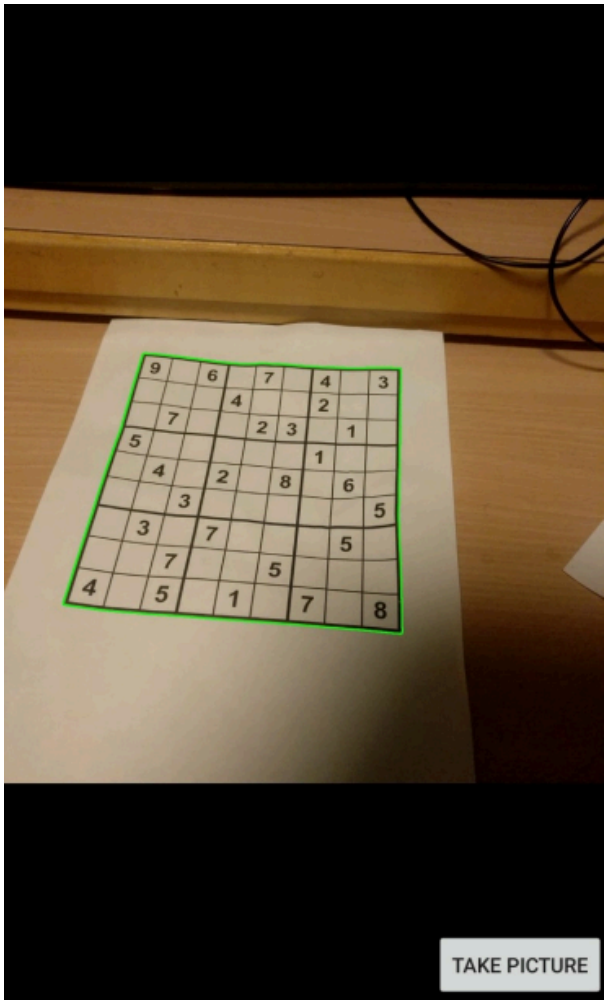


Figure 14: Detection and highlighting of Sudoku (source: reference [3])

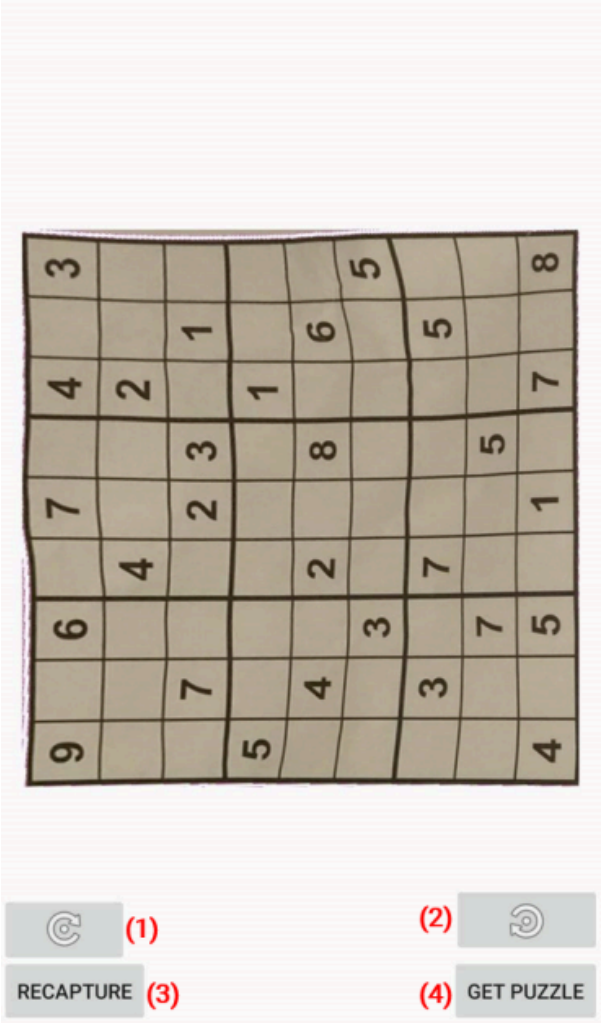


Figure 15: Incorrectly oriented grid (source: reference [3])

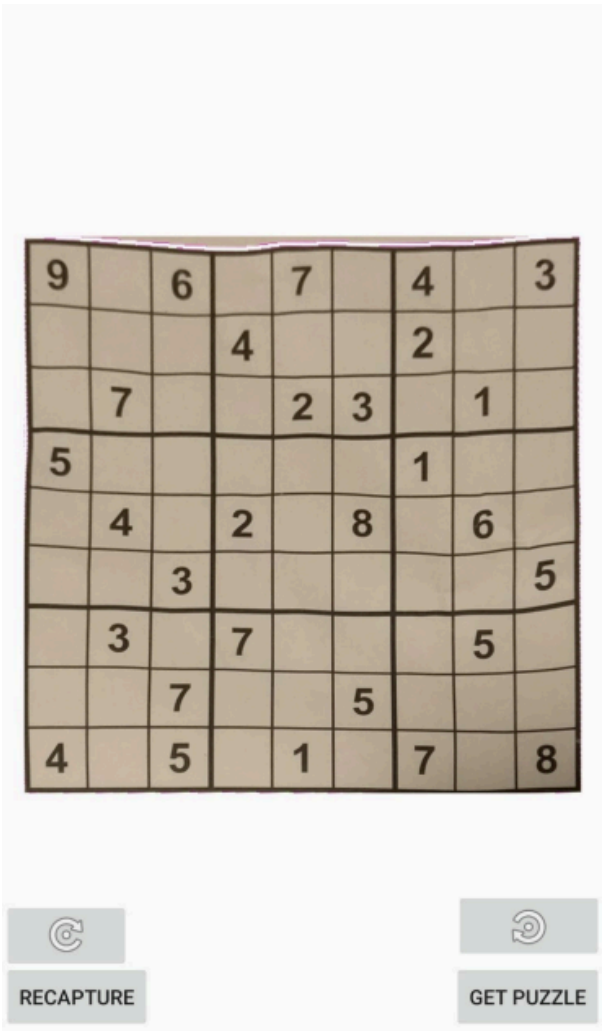


Figure 16: Correctly oriented puzzle. (source: reference [3])

Figure 17: demongolem's user avatar (source: reference [6])

4.2. Use of HashSet collections for validation

In line with [1], the Sudoku Solver Visualiser makes use of a HashSet collection to validate entries within the SudokuSolverGUI class. This feature helps filter potential values for each cell on the Sudoku grid, guaranteeing that every cell contains a distinct number from 1 to 9 or remains empty. The implementation of HashSet enables swift detection of duplicates in rows, columns, and subgrids by storing unique values and offering constant time complexity for various operations such as element checking and addition.
Furthermore, as pointed out in [3] p. 31-35, the HashSet is also employed to track illegal values for each empty cell in the context of puzzle-solving algorithms. By storing forbidden values in a HashSet, the algorithm can efficiently manage and update constraints for every cell while ensuring optimal performance.
Additionally, [3] p. 26-30 underscores the significance of utilizing a Set data structure, particularly HashSet, for validating rows based on puzzle restrictions. By employing a Set to retain numbers already encountered in rows, the process of identifying duplicates is simplified to constant time complexity (O(1), significantly reducing computational load compared to pairwise comparisons.
To sum up, the integration of HashSet collections for validation purposes in the Sudoku Solver Visualiser plays a pivotal role in maintaining adherence to Sudoku puzzle constraints by effectively handling unique values within rows, columns, and subgrids. This approach not only boosts the algorithm's efficiency but also streamlines the validation process while solving puzzles.

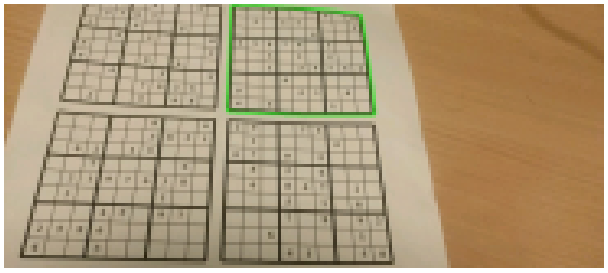


Figure 18: Left puzzle detected Figure 5.2: Right puzzle detected (source: reference [3])

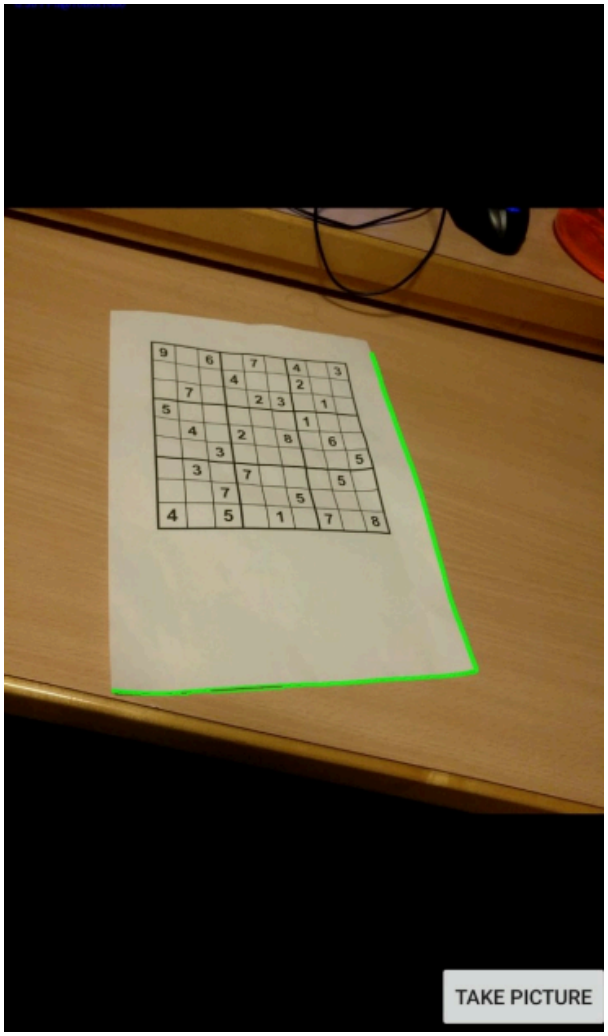


Figure 19: Page detected instead of Sudoku (source: reference [3])

5. Visualizing Steps

5.1. Recursive approach in filling the board

The methodical process of populating the board adopts a recursive strategy, commencing from an empty board and haphazardly filling the initial cell. Subsequently, when moving to the subsequent vacant cell, a random number within the range of 1-9 is generated and scrutinized against Sudoku regulations prior to insertion. If all prerequisites are satisfied, the digit is added, and the sequence advances to the succeeding cell. Nevertheless, in scenarios where no suitable values can be accommodated in a cell, a backtracking maneuver ensues by revisiting preceding cells and exploring alternative combinations until a viable solution is attained.
This recursive filling technique emulates the resolution of a Sudoku puzzle by creating fresh instances with minor deviations at each phase, analogous to decision trees in recursion. The procedure exemplifies how a puzzle can be disintegrated into subassignments that are methodically resolved until a comprehensive solution is reached. This approach aligns with the essence of Sudoku puzzles as undertakings that can be delineated by their analogous subtasks.
The visual representation of this recursive strategy can be witnessed in operation through platforms like trinket.io, where backtracking occurs when potential values become impracticable during the resolution process. This graphical depiction bolsters comprehension of how backtracking functions in real-time while unraveling Sudoku conundrums.
In essence, the recursive methodology for populating the board in the Sudoku Solver Visualiser adheres to a structured approach of generating and validating values for each cell until a feasible solution is attained. This tactic demonstrates the intricacy and nuances entailed in efficiently solving Sudoku puzzles while upholding adherence to stipulated constraints and regulations. See references: [14], [5], [8].

5.2. Delays for step-by-step visualization

"In the Sudoku Solver Visualizer, deliberate pauses aid in understanding the backtracking algorithm's execution ([14]). The ConstraintLayout ensures optimal performance for the User Interface ([3] p. 26-30). Prioritizing speed sustains user engagement, with pauses enabling smooth progress monitoring ([3] p. 21-25). Intentional delays allow for immediate updates during puzzle-solving ([5]).
Proposed enhancements like grid visualization improvements align with the deliberate pauses for step-by-step visualization ([3] p. 51-55). These pauses enhance user satisfaction and lay the groundwork for future advancements like manual solving capabilities. Overall, intentional pauses in the visualizer enhance user comprehension and engagement by providing a clear portrayal of the puzzle-solving process.

6. Solution Found Mechanism

6.1. Criteria for determining a valid solution

In order to ascertain a valid solution using the Sudoku Solver Visualizer, specific criteria must be satisfied. As highlighted in [3] p. 46-50, the solver adheres to the predefined requirements

established during the design phase, ensuring the identification of a correct solution for any given puzzle, if such a solution exists. Employing a backtracking methodology, as outlined in [10], involves sequentially assigning numbers to empty cells while confirming their safety by cross-referencing with the current row, column, and 3x3 subgrid to avoid repetitions.
Moreover, as indicated in [14], there exist two distinct termination conditions for the backtracking algorithm utilized in Sudoku resolution. The first condition arises when the entire board is filled with no remaining blank spaces for evaluation. The second condition arises when the ongoing candidate solution fails to meet the ultimate objective of solving the puzzle. These conditions effectively prevent the algorithm from endlessly cycling in pursuit of a solution and enable it to efficiently ascertain when a valid solution has been successfully reached.
The Sudoku Solver Visualizer offers real-time updates within its graphical user interface throughout the puzzle-solving process, as detailed in [13]. This feature allows users to monitor the solver's progress and witness how each individual step contributes towards achieving a valid solution. By visually representing each stage of the solving process with strategically placed delays for clarity, users can grasp the underlying mechanics of the algorithm and develop an appreciation for the intricacies involved in solving Sudoku puzzles.
To sum up, a valid solution in the Sudoku Solver Visualizer hinges on meeting predetermined design requirements, following a systematic backtracking approach with defined termination conditions, and providing real-time updates for enhanced visualization purposes. Users can confidently rely on this tool to effectively tackle Sudoku puzzles while gaining insights into the algorithmic processes driving its functionality.

7. Error Handling and Exceptions

7.1. Handling duplicate numbers in cells

Addressing the presence of duplicate numbers within cells is a fundamental aspect when utilizing the Sudoku Solver Visualizer. As emphasized in [6], the key to successfully solving Sudoku puzzles lies in employing specific strategies rather than resorting to random guesses. However, as noted by Donald.McLean in [6], some of the most challenging puzzles may necessitate unconventional techniques, leading enthusiasts to make educated guesses instead of fully understanding these methods. Nevertheless, it is important to acknowledge that puzzles with a unique solution should not require guesswork, as asserted by Kendall Frey in [6].
Within the implementation of the SudokuSolverGUI class, outlined in [1], a series of functions are in place to ensure accurate validation of each cell on the board. The backtracking algorithm detailed in [11] guarantees that every number placed on the board adheres to the rules of its respective row, column, and subgrid. This process entails verifying the validity of potential moves based on the available options for each cell.
Moreover, real-time updates during puzzle-solving, highlighted in [3] p. 41-45, play a critical role in effectively illustrating each step of the process. By incrementing values while maintaining their validity, as demonstrated in [11], the solver can explore various possibilities without resorting to guesswork. Additionally, error-handling mechanisms specified in [3] p. 41-45 ensure that users receive detailed messages when inconsistencies are identified, guiding them on how to address these issues.

In essence, managing duplicate numbers within cells using the Sudoku Solver Visualizer necessitates a methodical approach to validate each move without relying on arbitrary guesses. By incorporating robust validation procedures and providing real-time updates during puzzle-solving, users can navigate through different scenarios efficiently while upholding the integrity of their solutions.

8. Threading and Performance Optimization

Effective threading and performance optimization are paramount in maximizing the efficiency of the Sudoku Solver Visualizer. As highlighted in [12], the integration of WebAssembly modules developed in C++ and Rust can significantly enhance processing speed. Employing multiple threads for tasks demanding substantial computational power enables the solver algorithm to operate more effectively. However, as noted in [3] p. 31-35, it is crucial to maintain consistent solving times when implementing multithreading to ensure precise and dependable outcomes. Furthermore, exploring various solving algorithms, as recommended in [3] p. 51-55, can further elevate the performance of the Sudoku Solver Visualizer. By evaluating the effectiveness of different algorithms, developers can pinpoint the most efficient approach to swiftly and accurately solve Sudoku puzzles. Additionally, as indicated in [8], refining the backtracking algorithm by prioritizing empty cells based on their number of potential solutions can streamline the process, leading to quicker puzzle resolutions.
Moreover, as emphasized in [12], techniques such as downsizing image quality and leveraging a pool of Web Workers for concurrent processing can also contribute to performance improvement. These methods aim to simplify puzzle-solving procedures and decrease computational load, ultimately enhancing user experience.
In summary, by incorporating multithreading, exploring diverse solving algorithms, optimizing image processing methods, and implementing efficient backtracking strategies, the Sudoku Solver Visualizer can achieve heightened levels of performance enhancement. These endeavors align with the project's objective of delivering users with a rapid and effective automated solution for effortlessly and accurately solving Sudoku puzzles.

9. Conclusion

In final analysis, the Sudoku Solver Visualizer holds immense potential in enhancing the solving experience of Sudoku puzzles [3] p. 51-55. The app's capability to identify, isolate, and resolve Sudoku puzzles through a smartphone camera demonstrates the innovation and convenience it brings to puzzle fans [3] p. 6-10. Despite certain drawbacks related to image quality and recognition performance, particularly with unusual fonts or extreme lighting conditions [3] p. 51-55, the overall effectiveness of the app shows promise when used in ideal conditions [3] p. 51-55.
Suggestions for future improvements, such as introducing manual solving features, hint functionalities, and grid visualization enhancements, signal an ongoing commitment to improving user experience and functionality [3] p. 51-55. Providing a tutorial that explains the capabilities of the app could further assist users in maximizing its potential [3] p. 51-55. Furthermore, exploring alternative algorithms like the dancing links algorithm for faster solving

could be advantageous in enhancing performance [3] p. 51-55.
The real-time updates in GUI during puzzle-solving and the recursive approach to filling the board make significant contributions to the interactive nature of the Sudoku Solver Visualizer [3] p. 6-10. The validation process for rows, columns, and subgrids using HashSet collections ensures precise solutions are produced [3] p. 6-10. Additionally, error-handling mechanisms for identifying duplicate numbers in cells enhance the reliability of the solver [1].
Overall, with a focus on user interface design, puzzle-solving algorithms, cell validation techniques, and solution accuracy criteria, the Sudoku Solver Visualizer emerges as a valuable tool for Sudoku enthusiasts seeking efficient and visually engaging puzzle-solving experiences. By addressing current limitations and implementing future work suggestions [3] p. 51-55, this app has the potential to become even more robust in its functionality and user appeal.

References

- [1] A. Spittel. "Coming Back to Old Problems: How I Finally Wrote a Sudoku Solving Algorithm". (accessed Jul 10, 2024). [Online]. Available: https://dev.to/aspittel/how-i-finally-wrote-a-sudoku-solver-177g
- [2] "Sudoku Solver". Apr 2024. [Online]. Available: https://www.scribd.com/document/700019852/Project-report-for-Sudoku-Solver
- [3] N. Pilavakis. "Visual Sudoku Solver". Apr 2020. [Online]. Available: https://project-archive.inf.ed.ac.uk/ug4/20201867/ug4_proj.pdf
- [4] D. Panchal. "Sudoku Solver - A Visualizer made using Backtracking Algorithm". Apr 2021. [Online]. Available: https://community.codenewbie.org/dhhruv/sudoku-solver-a-visualizer-made-using-backtracking-algorithm-4a33
- [5] T. Kumar. "Sudoku-Backtracking algorithm and visualization". May 2020. [Online]. Available: https://medium.com/analytics-vidhya/sudoku-backtracking-algorithm-and-visualization-75adec8e860c
- [6] "Are there published Sudoku puzzles that require guessing?". (accessed Jul 10, 2024). [Online]. Available: https://puzzling.stackexchange.com/questions/12/are-there-published-sudoku-puzzles-that-require-guessing
- [7] "Backtracking Algorithm - Sudoku Solver". Mar 2023. [Online]. Available: https://www.101computing.net/backtracking-algorithm-sudoku-solver/
- [8] J. C. K. Lung. "Golang Newbie: Solving Sudoku using backtracking algorithm". Jun 2022. [Online]. Available: https://medium.com/@gmu1233/golang-newbie-solving-sudoku-using-backtracking-algorithm-8bd54cb9ee35
- [9] "How to Use Progress Bars". (accessed Jul 10, 2024). [Online]. Available: https://docs.oracle.com/javase%2Ftutorial%2Fuiswing%2F%2Fcomponents/progress.html
- [10] "Sudoku Visualizer". (accessed Jul 10, 2024). [Online]. Available: https://sudoku-gg.vercel.app/
- [11] A. Spittel. "How I came back to an old problem and finally wrote a Sudoku-solving algorithm". Jun 2018. [Online]. Available: https://medium.com/free-code-camp/coming-back-to-old-problems-how-i-finally-wrote-a-s

udoku-solving-algorithm-3b371e6c63bd
- [12] C. Eberhardt. "A WebAssembly Powered Augmented Reality Sudoku Solver". Jan 2020. [Online]. Available: https://blog.scottlogic.com/2020/01/03/webassembly-sudoku-solver.html
- [13] "Sudoku Solver". (accessed Jul 10, 2024). [Online]. Available: https://leetcode.com/problems/sudoku-solver/discuss/15835/java-generate-validate-and-solve-nxn-sudoku-puzzle-with-visualization-tracking-and-100-readable-code
- [14] B. Follow. "Simple Sudoku with Backtracking". Mar 2020. [Online]. Available: https://medium.com/swlh/simple-sudoku-with-backtracking-bb4813ddabb1
- [15] "Sudoku solver in JS". (accessed Jul 10, 2024). [Online]. Available: https://stackoverflow.com/questions/42736648/sudoku-solver-in-js