SUDOKO

Team Members:

- Rawan Essam 7750
- Aya Salah 7361
- Nada Mostafa 7545

➢ Introduction

Sudoku, a widely popular logic-based number-placement game, has captivated players worldwide with its intriguing challenge. In this project, we delve into the realm of Constraint Satisfaction Problems (CSPs) to tackle the Sudoku puzzle using various algorithms and techniques. Our goal is to develop an intelligent agent capable of solving Sudoku puzzles efficiently while highlighting fundamental concepts such as backtracking, arc consistency, and heuristic search.

This report documents our journey through the creation of a Sudoku solver that not only provides solutions but also offers insights into the underlying algorithms and strategies employed. We explore two distinct modes: one featuring a graphical user interface (GUI) illustrating the AI agent's prowess in solving Sudoku puzzles, and another mode allowing users to input puzzle representations for the agent to solve interactively.

Throughout this endeavor, we adhere to specific requirements outlined in the project, including the implementation of backtracking algorithms for puzzle validation and generation, as well as the incorporation of arc consistency to ensure a robust solution methodology. By representing Sudoku as a CSP and defining appropriate arcs and constraints, we establish a solid foundation for tackling puzzles of varying complexities.

In addition to presenting our well-commented code, this report highlights sample runs alongside their corresponding arc consistency trees, providing a visual representation of the solving process. We also conduct comparisons across different initial board configurations, ranging from easy to hard puzzles, analyzing the time required for successful puzzle resolution.

Furthermore, the report delves into the data structures utilized, algorithmic insights, key assumptions, and any additional work undertaken to enhance the project's functionality. For those seeking an interactive experience, we explore a bonus feature that allows users to interactively input Sudoku boards, validating each step to ensure compliance with puzzle constraints.

Join us as we unravel the intricacies of Sudoku solving using CSPs, offering a comprehensive exploration of algorithms, strategies, and interactive elements in our quest to conquer the Sudoku challenge.

➢ Data Structures

1- Sudoku Grid Representation:

- **2D List:** The Sudoku grid is represented as a 9x9 2D list (self.puzzle) where each cell contains a number from 1 to 9 or is empty.

2- Domain Representation:

- **Set:** The domain of each cell is represented as a set of values (numbers 1 to 9) that can be placed in the cell. The get_domain_values function returns a set containing valid domain values for a specific cell.

3- Arcs and Constraints:

- **Queue:** The queue data structure is used to store arcs that need to be revised during arc consistency enforcement. It is used in the apply_arc_consistency function.
- **List (Steps):** The steps list is used to keep track of steps taken during arc consistency enforcement, specifically for debugging.

4- Auxiliary Data Structures:

- **List:** Various lists are used for temporary storage and processing of data, such as domain_values in solve, removed_values in revise, and old_domain in revise.

➢ Algorithms

1. Backtracking Algorithm:

- Used in solve_with_backtracking method.
- The backtracking algorithm is a brute-force search algorithm that systematically tries different possibilities, backtracking when it reaches a dead-end, until it finds a solution or exhausts all possibilities.
- It recursively explores each value for each empty cell, backtracking when a contradiction is reached until a valid solution is found.

2. Constraint Satisfaction Problem (CSP) Algorithm:

- The Sudoku puzzle is represented as a CSP.
- Variables: Each cell in the Sudoku grid is a variable.
- Domains: The possible numbers (1 to 9) that can be placed in each cell form the domain of each variable.
- Constraints: The Sudoku rules (no repetition in rows, columns, or subgrids) are represented as constraints between variables.

3. Arc Consistency Algorithm:

- Used in apply_arc_consistency method.
- Arc consistency is applied to ensure that each variable's domain is consistent with the constraints.
- The algorithm enforces arc consistency by iteratively revising the domains of variables based on constraints until no further changes can be made.
- It creates arcs between connected variables (cells) and revises domains by removing inconsistent values, maintaining consistency throughout the Sudoku grid.

4. Forward Checking:

- Used in forward_checking method.
- Forward checking is applied to check the consistency of domains for empty cells after each assignment.
- It ensures that no domain becomes empty, indicating an invalid move, during the solving process.

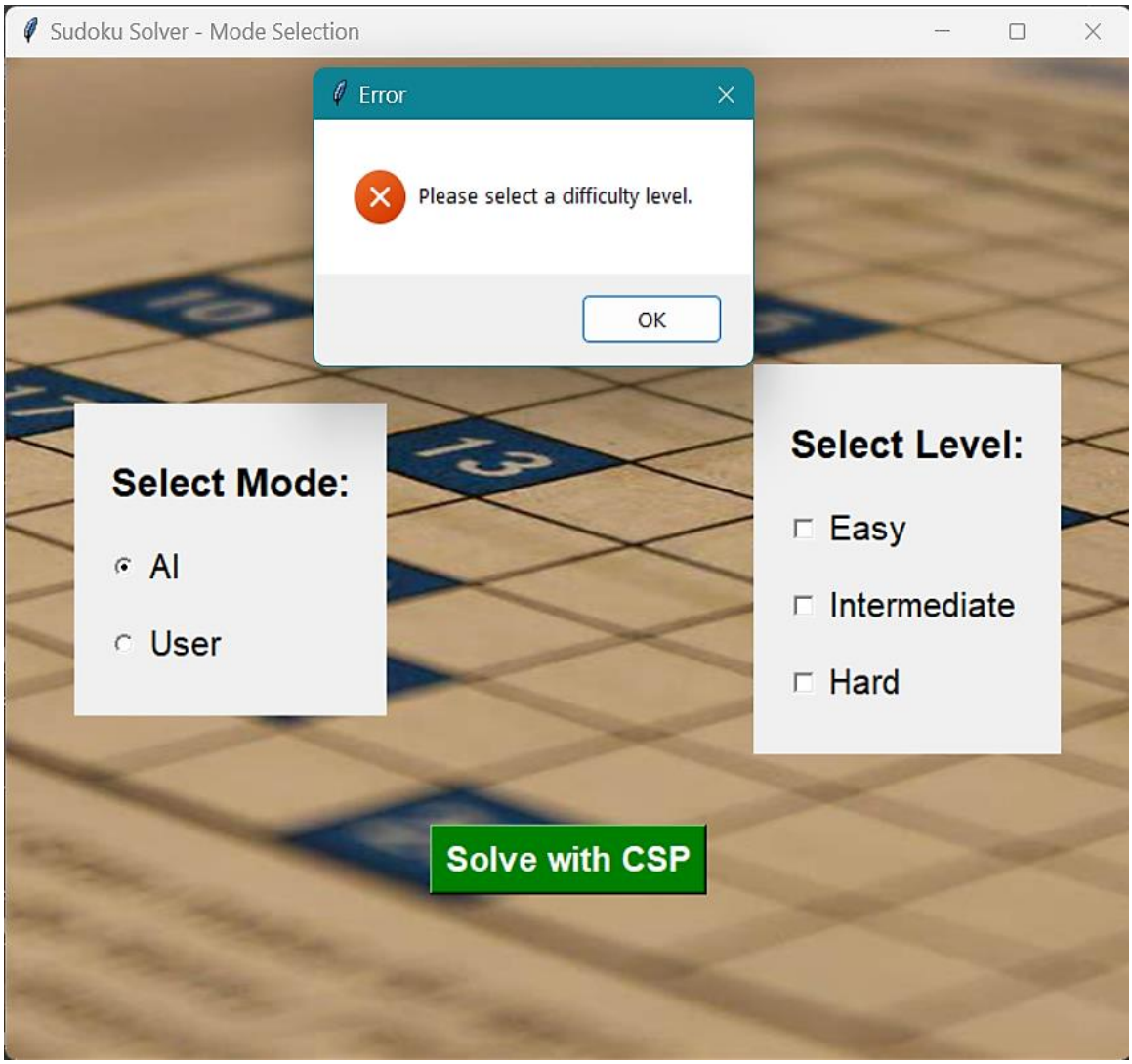
5. Minimum Remaining Values (MRV) Heuristic:

- Used in mrv method.
- The MRV heuristic selects the variable (cell) with the fewest remaining values to choose from, prioritizing variables likely to lead to a solution.
- It helps in reducing the search space and potentially finding a solution faster by exploring more promising variables early.

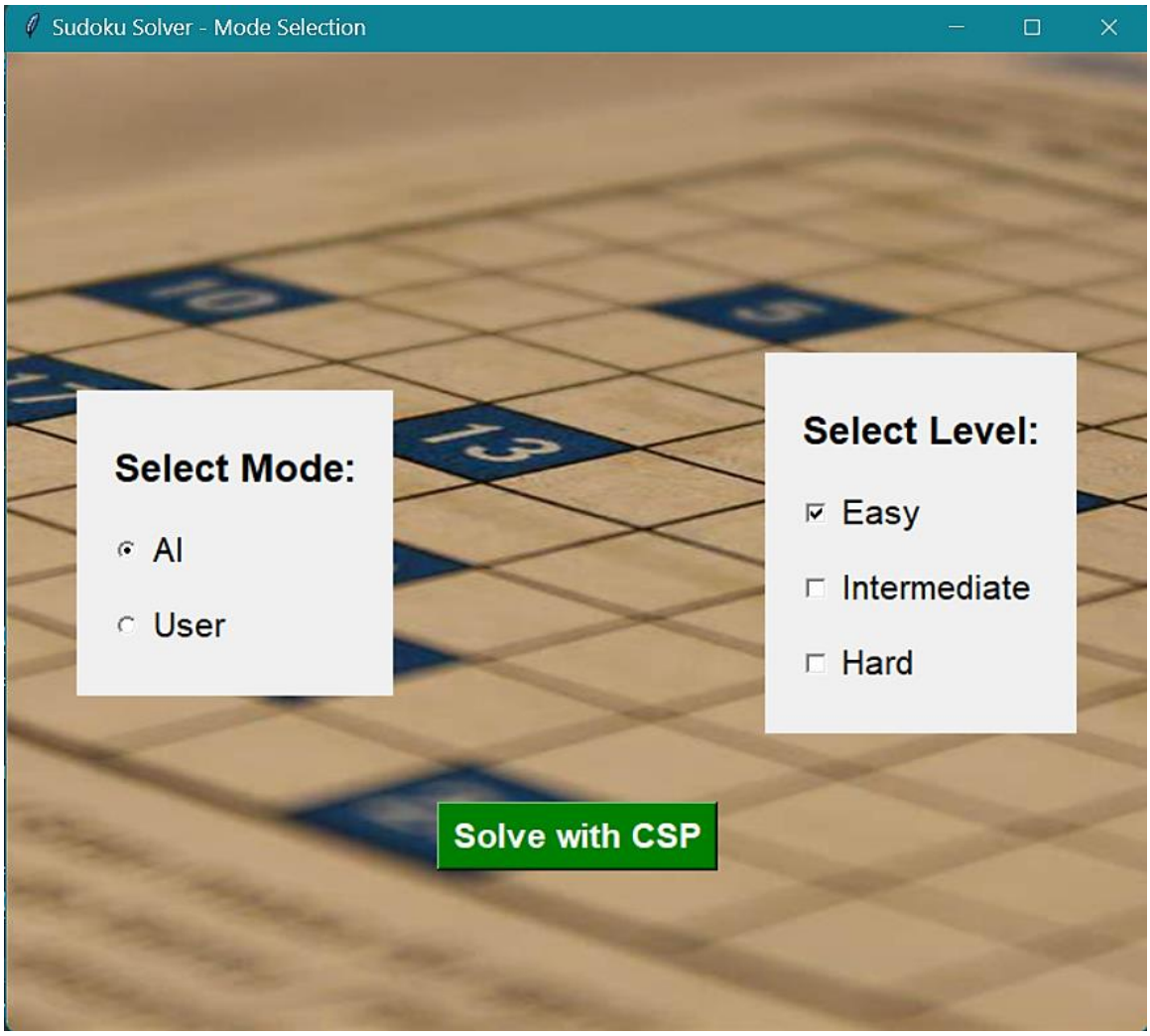
6. Least Constraining Value (LCV) Heuristic:

- Used in lcv method.
- The LCV heuristic prioritizes values for assignment that impose the fewest constraints on other variables, facilitating constraint satisfaction and reducing backtracking.

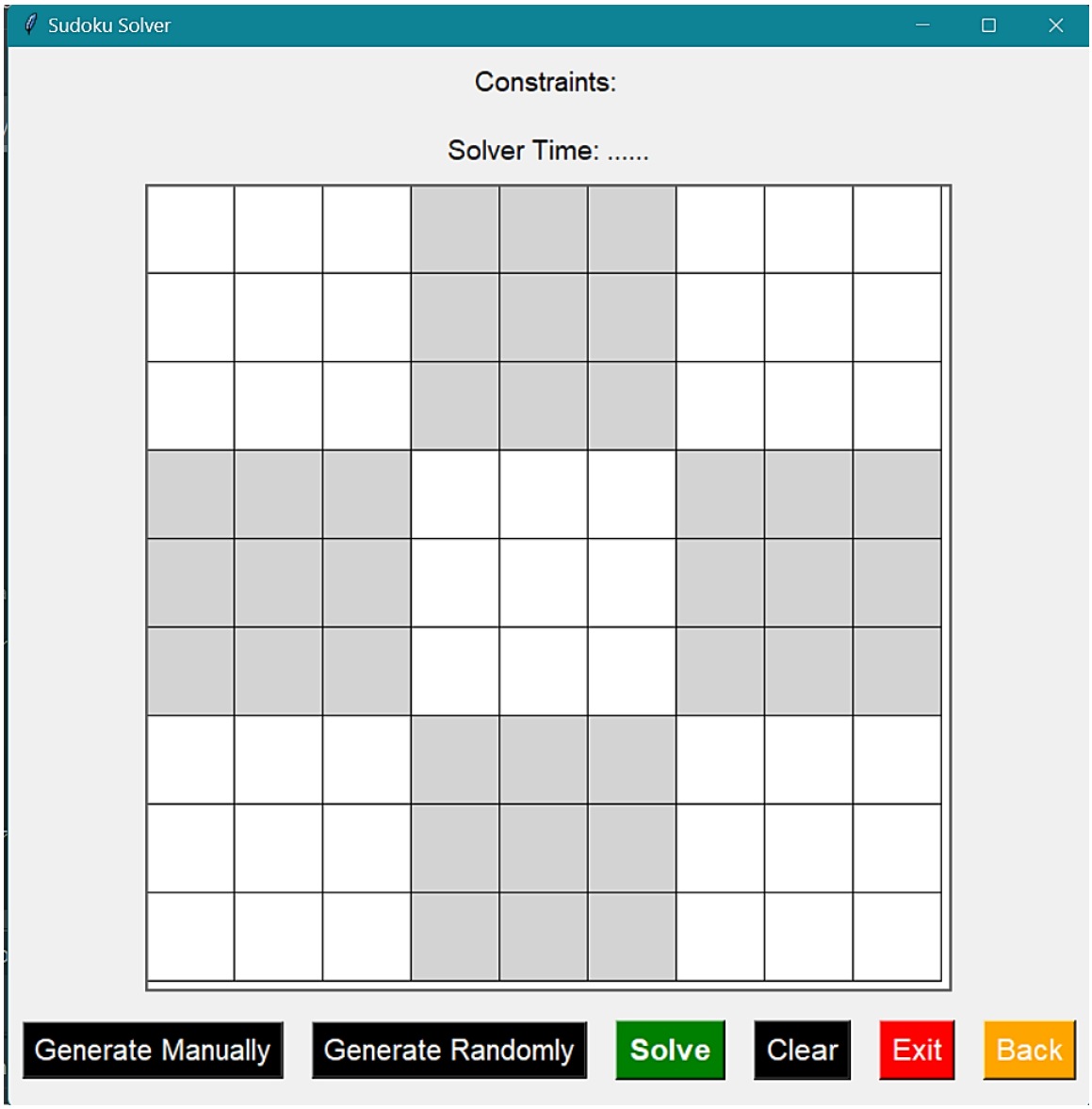
➢ Project



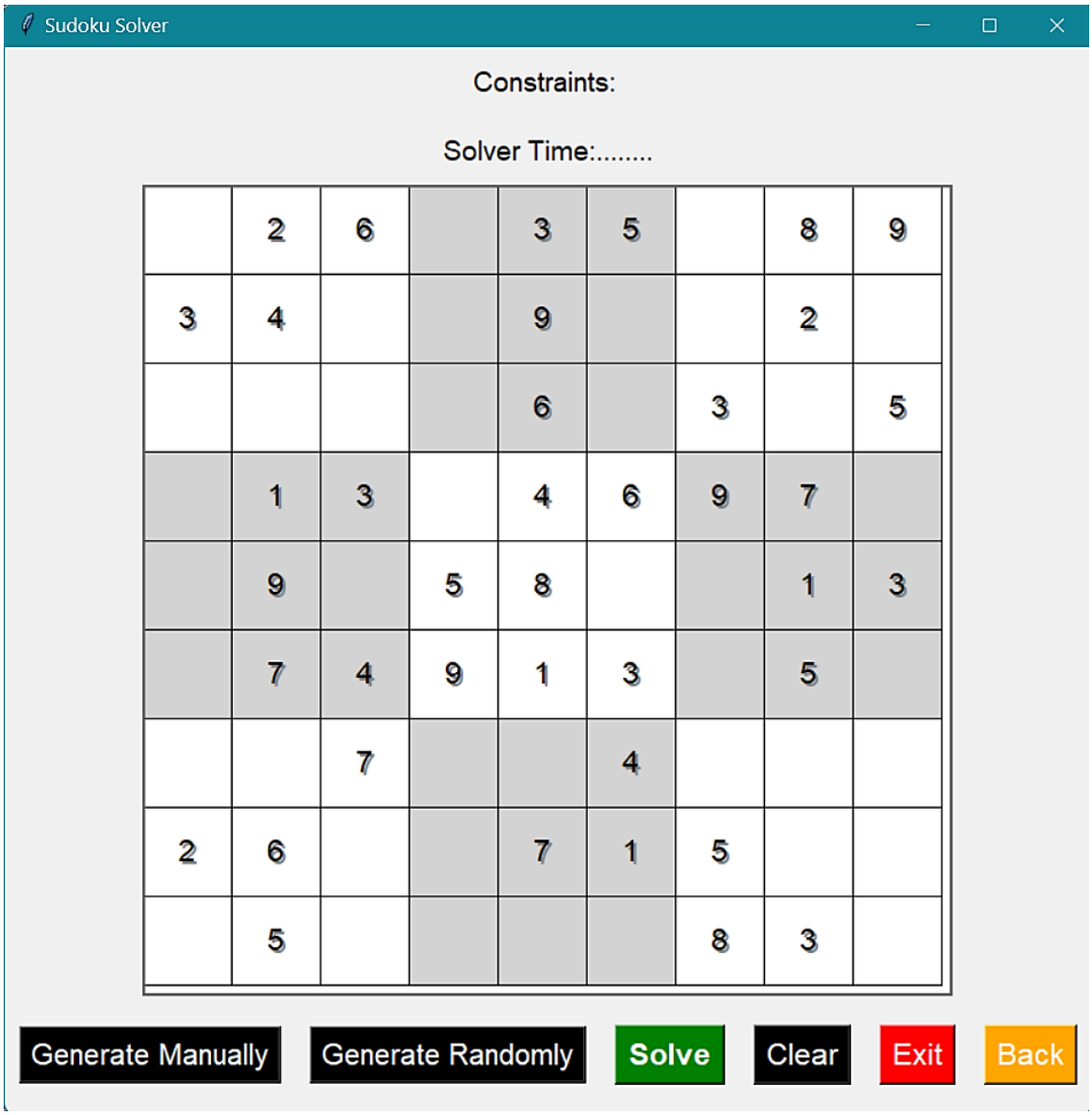
Mode Selection GUI: Select Level Constraint



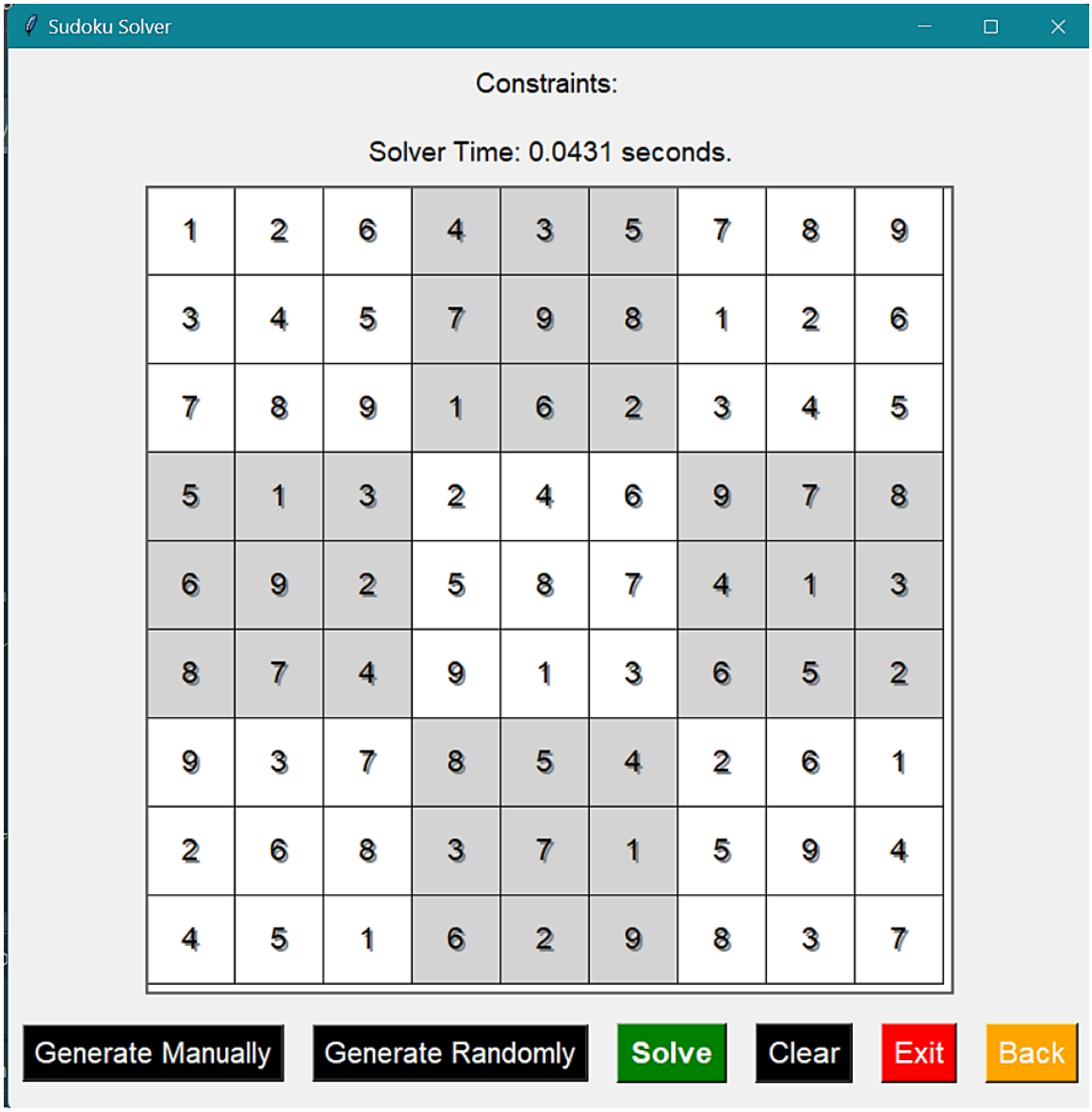
Mode Selection GUI: Select Easy Level in AI Mode


Sudoku Solver
Constraints:
Solver Time: .....
Generate Manually
Generate Randomly
Solve
Clear
Exit
Back

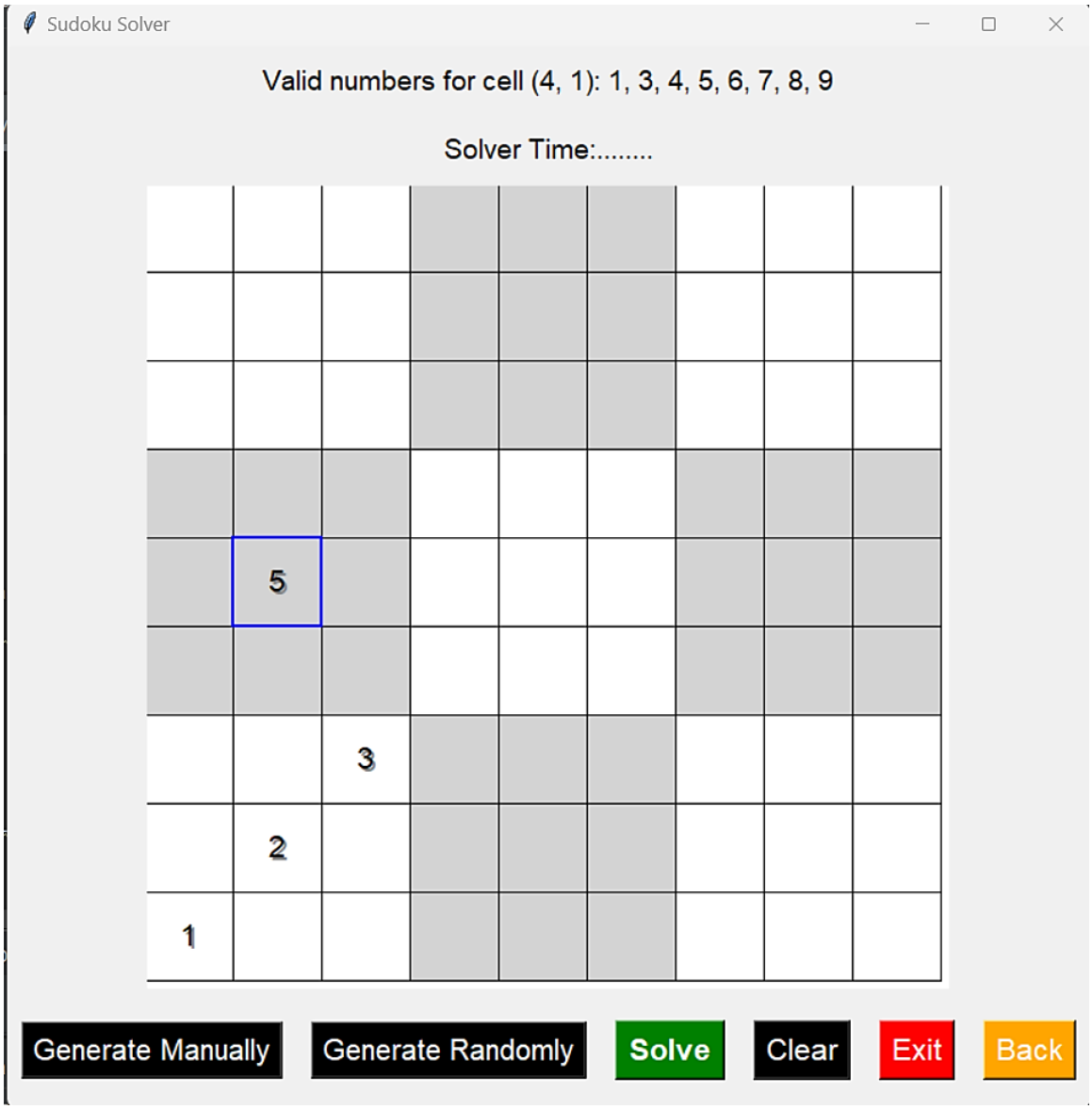
AI Mode GUI



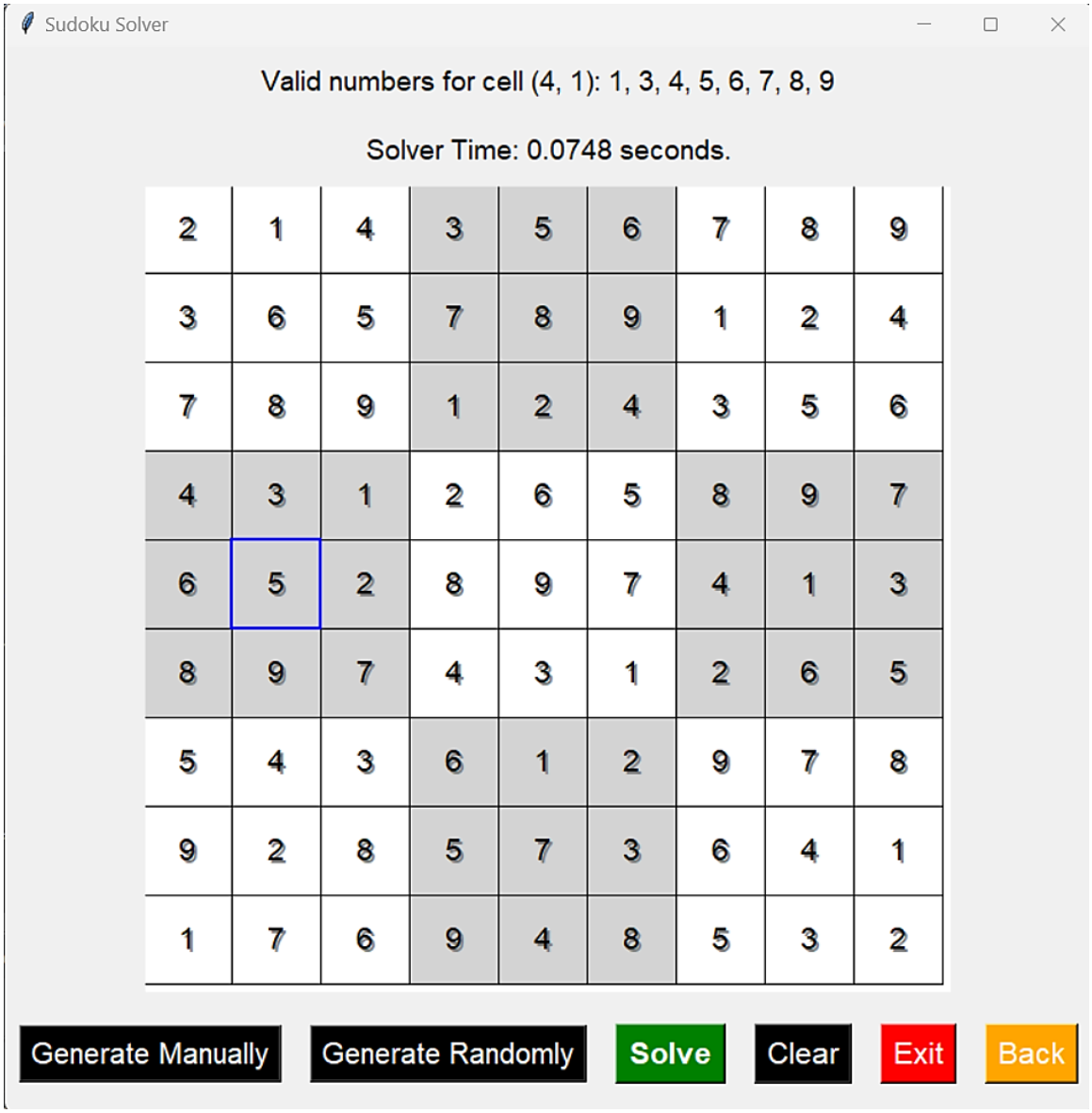
AI Mode GUI: Select Generate Randomly



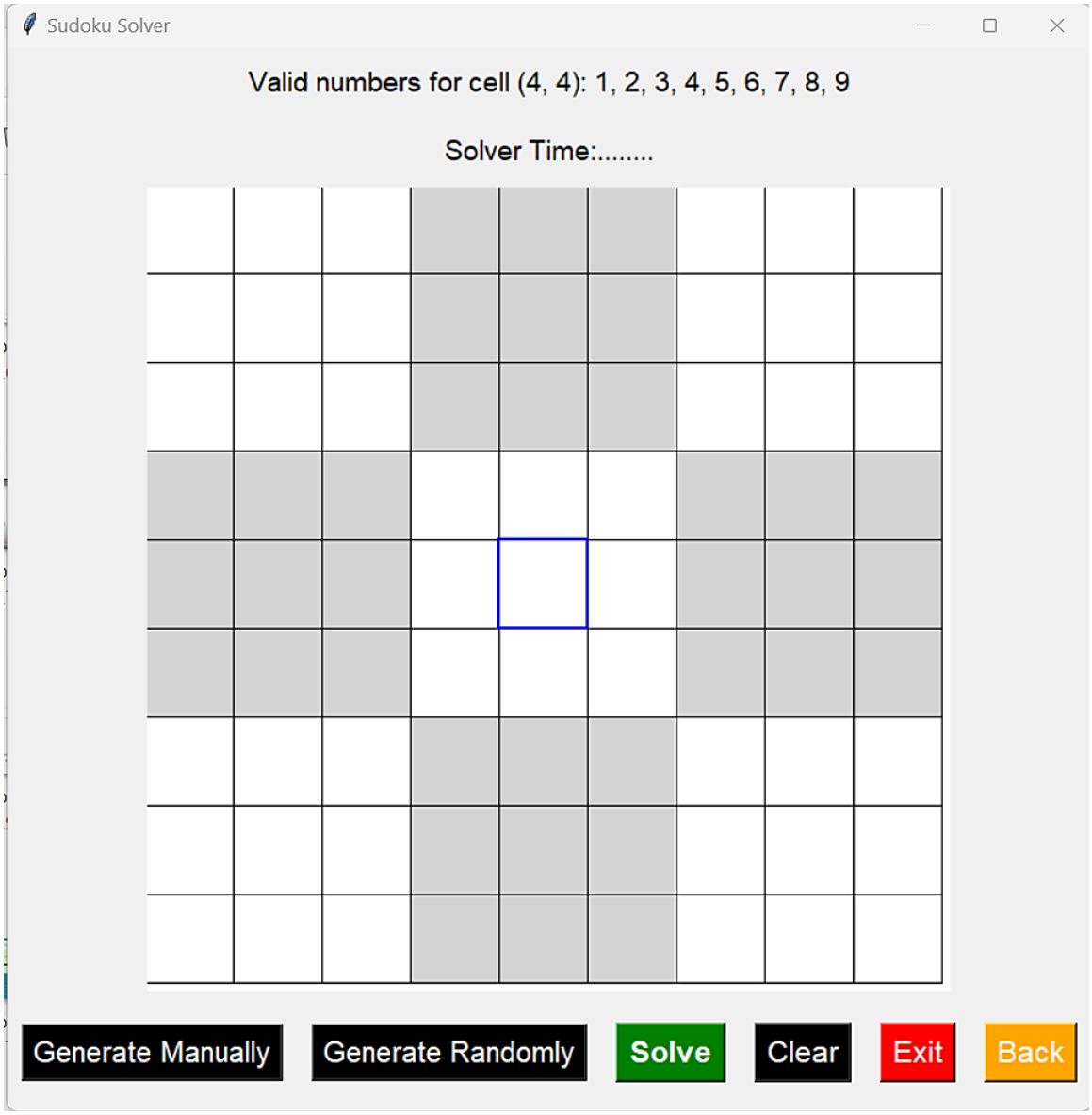
AI Mode GUI: Select Solve



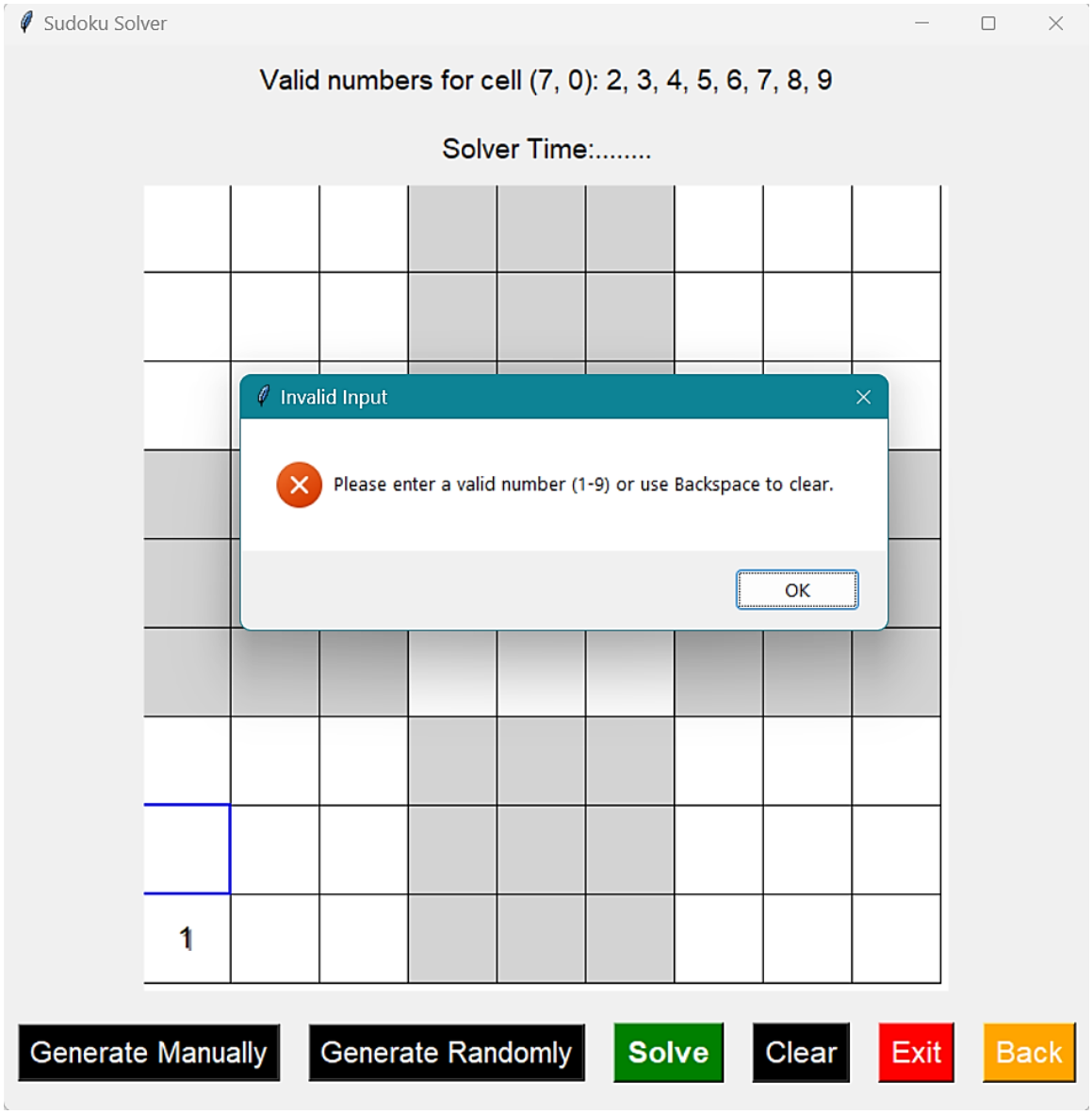
AI Mode GUI: Select Generate Manually



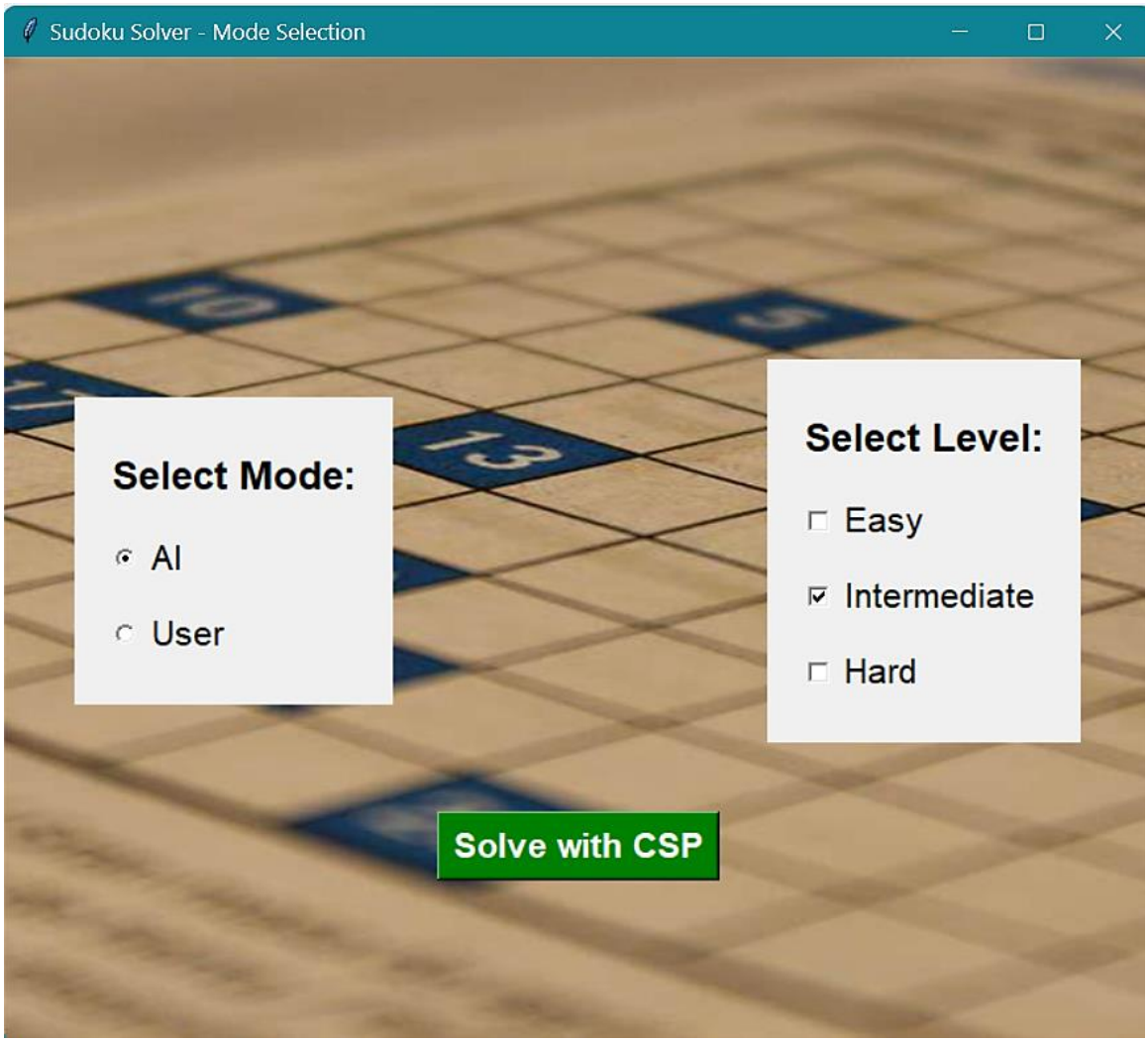
AI Mode GUI: Select Solve



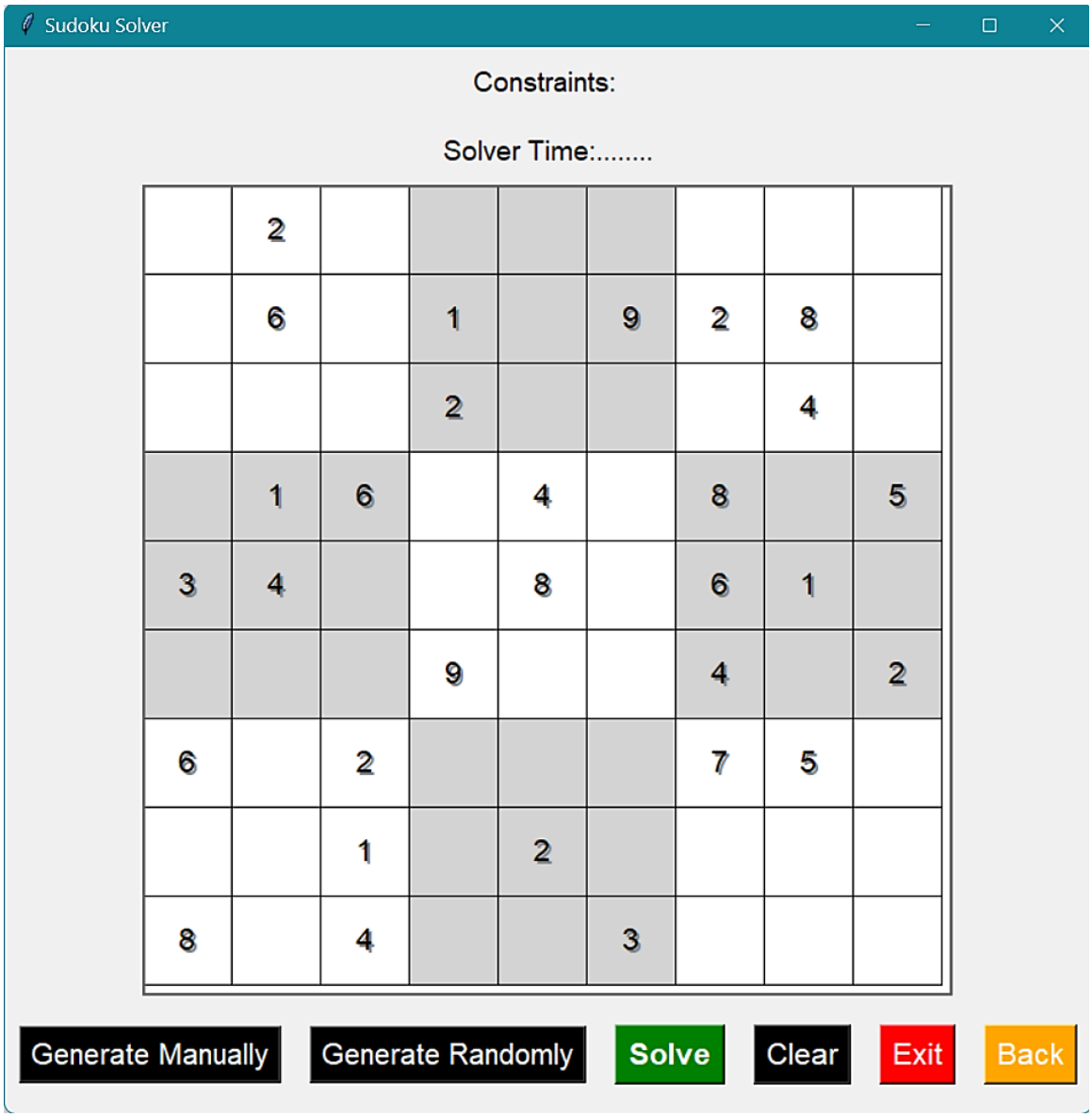
AI Mode GUI: Domain Constraints on Selecting Cell 4,4



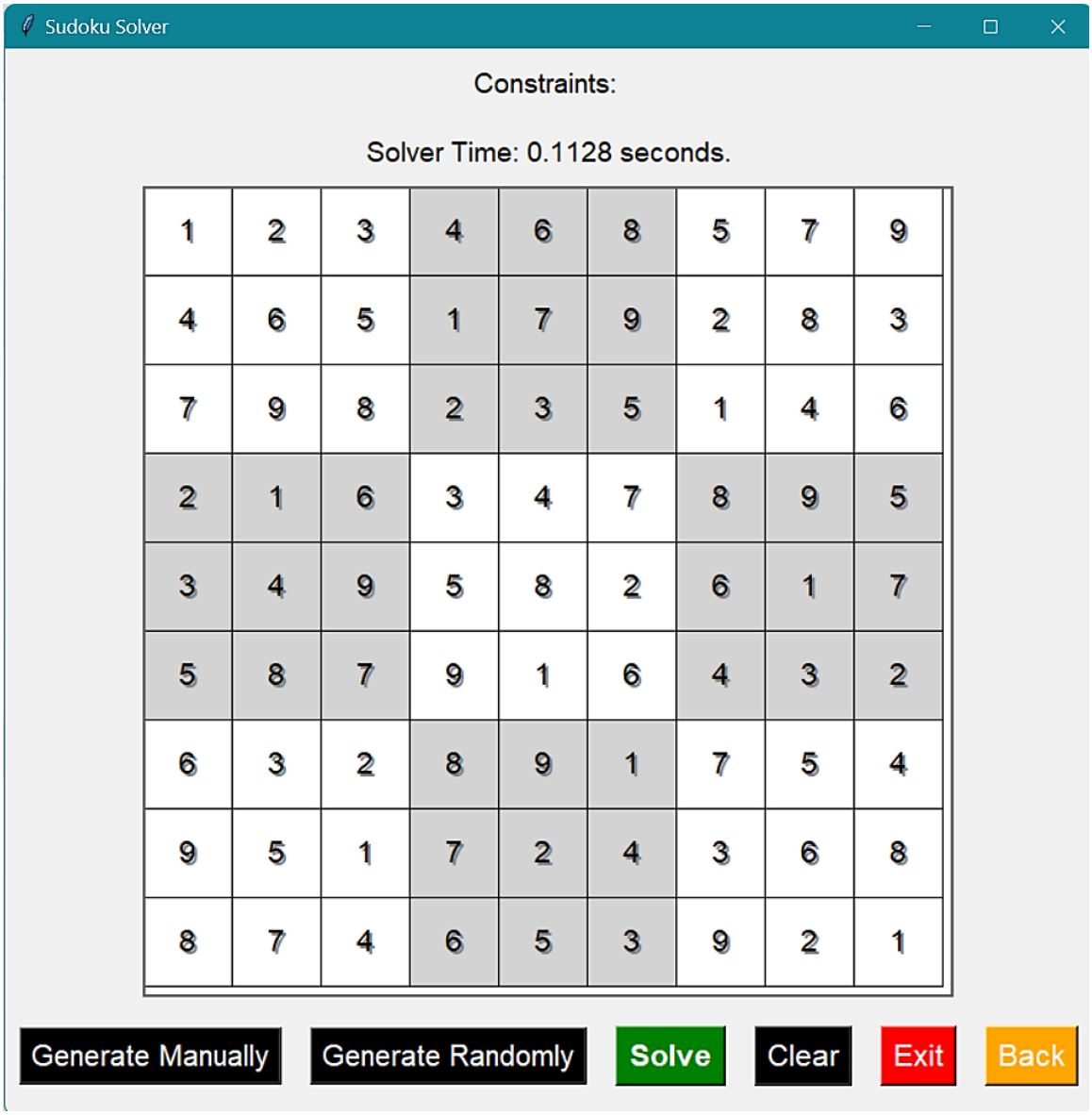
AI Mode GUI: Select Generate Manually And Enter An Invalid Value Not In Cell Domain



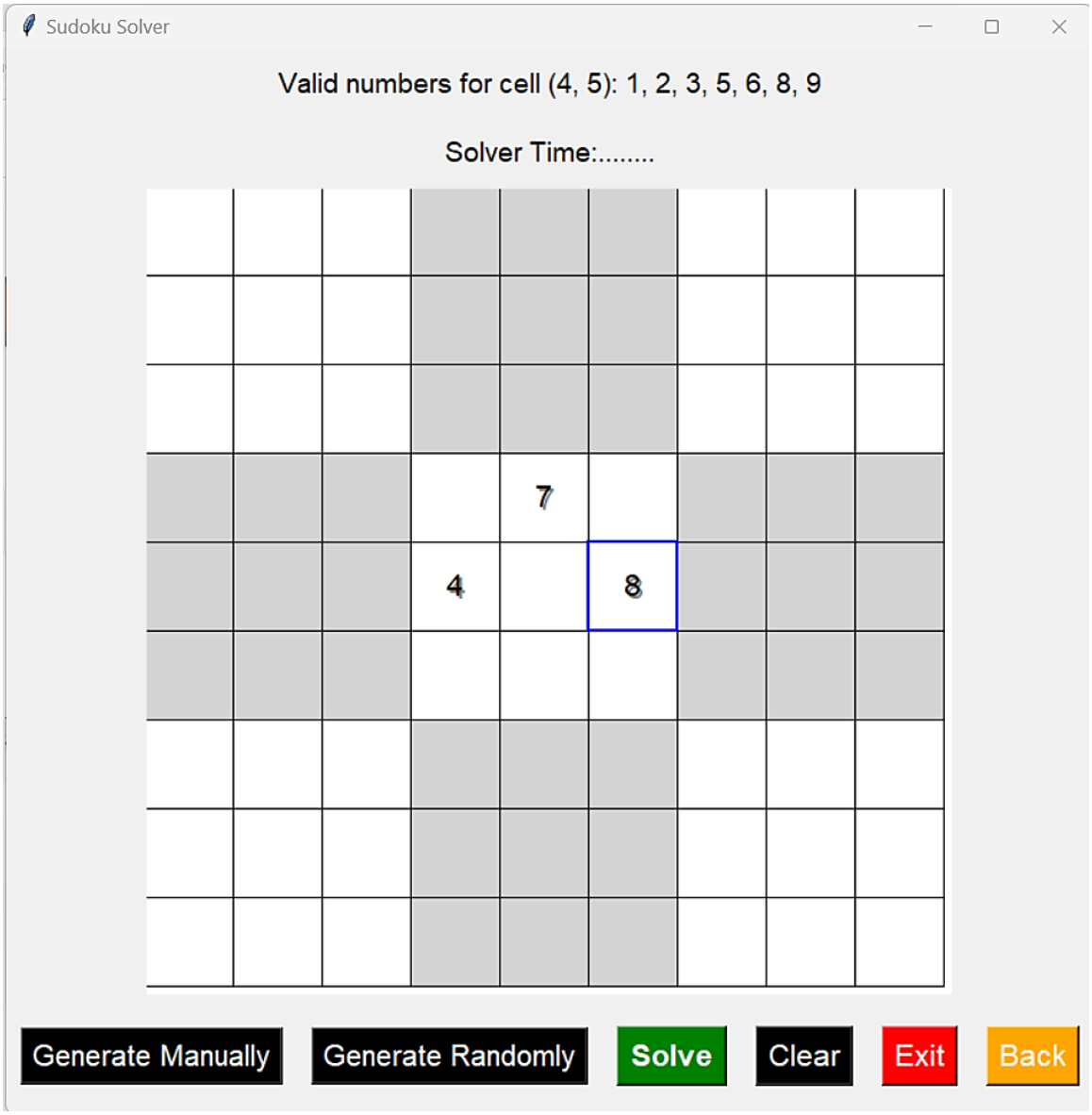
Mode Selection GUI: Select Intermediate Level in AI Mode



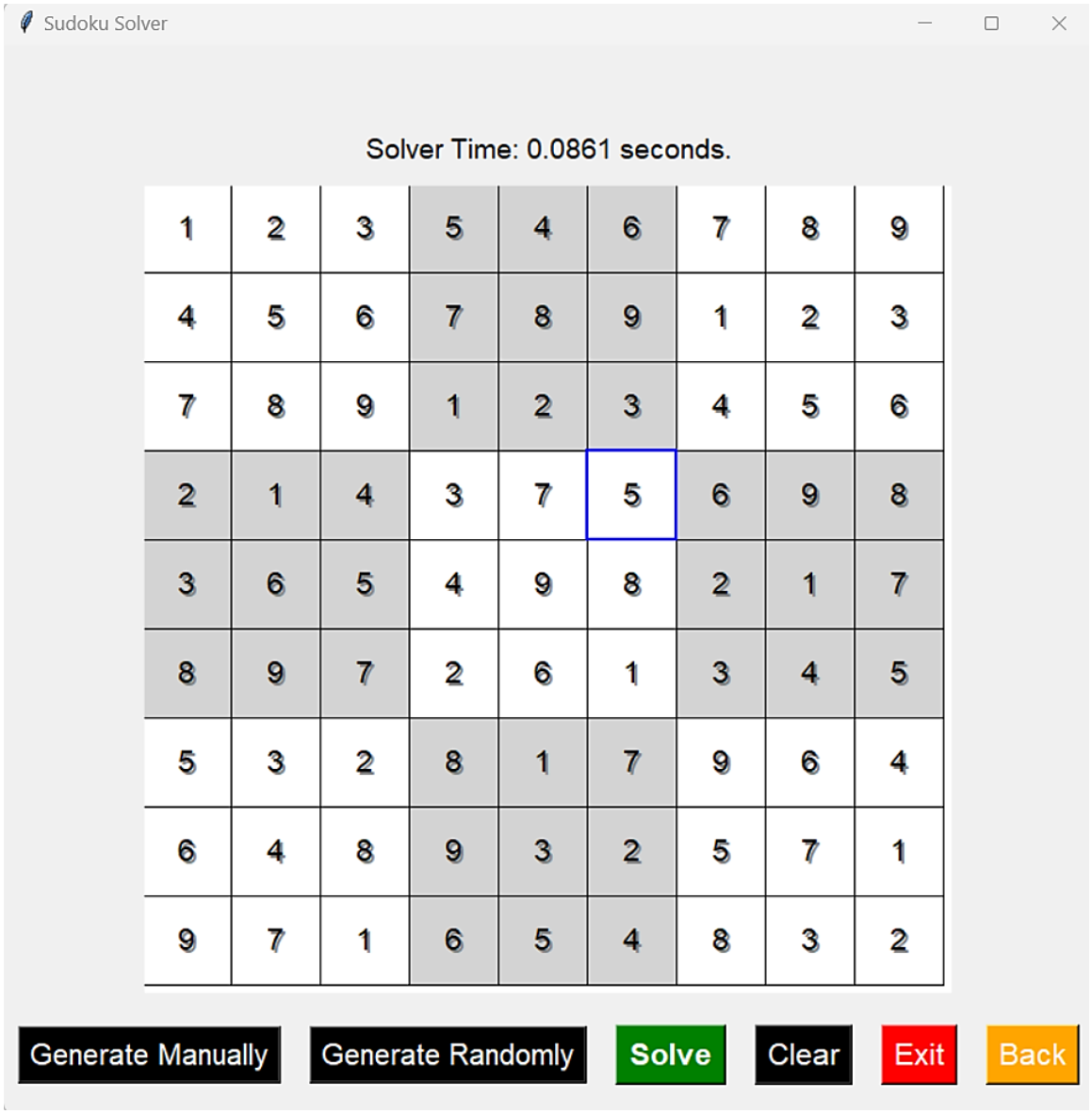
AI Mode GUI: Select Generate Randomly



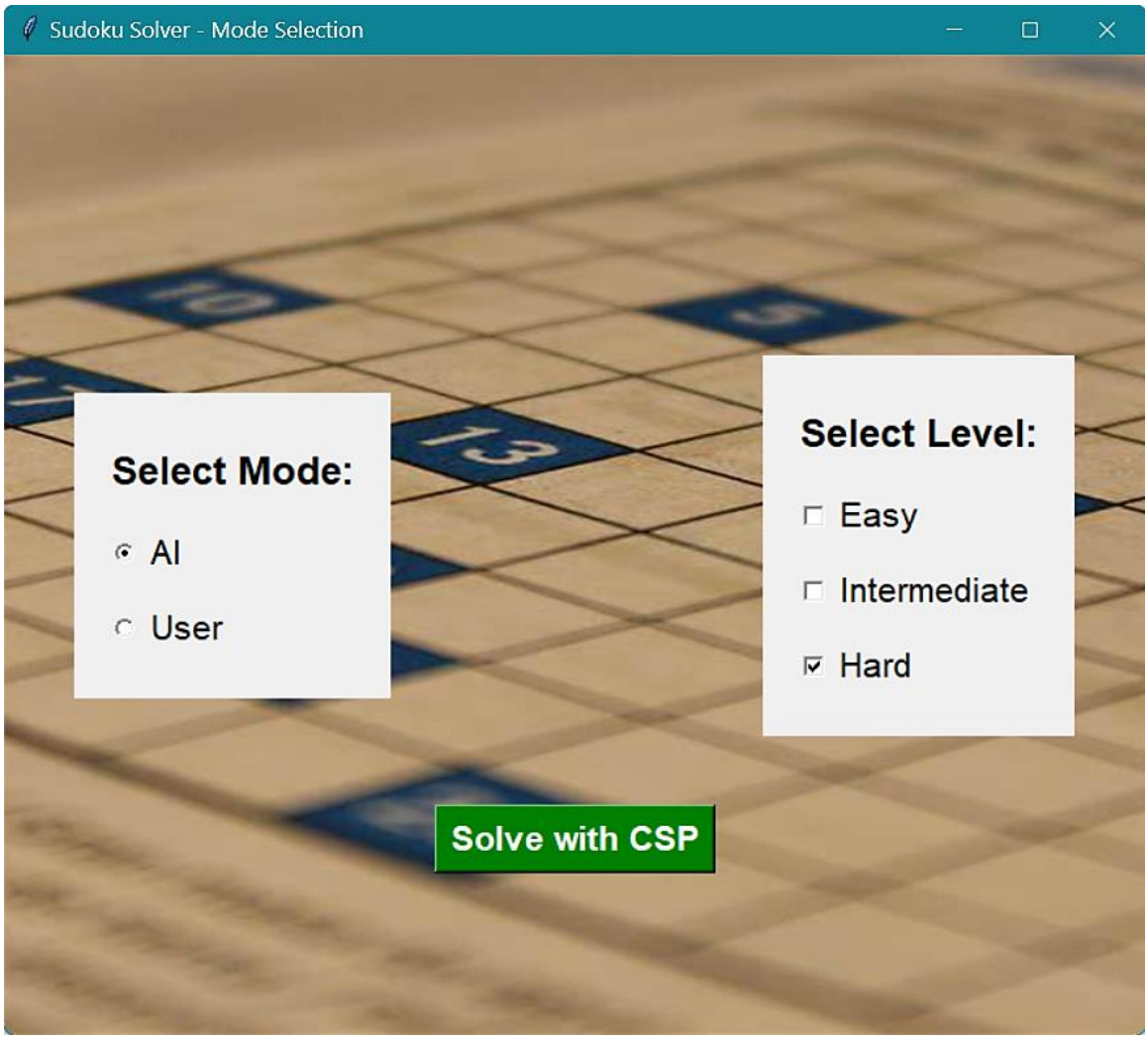
AI Mode GUI: Select Solve



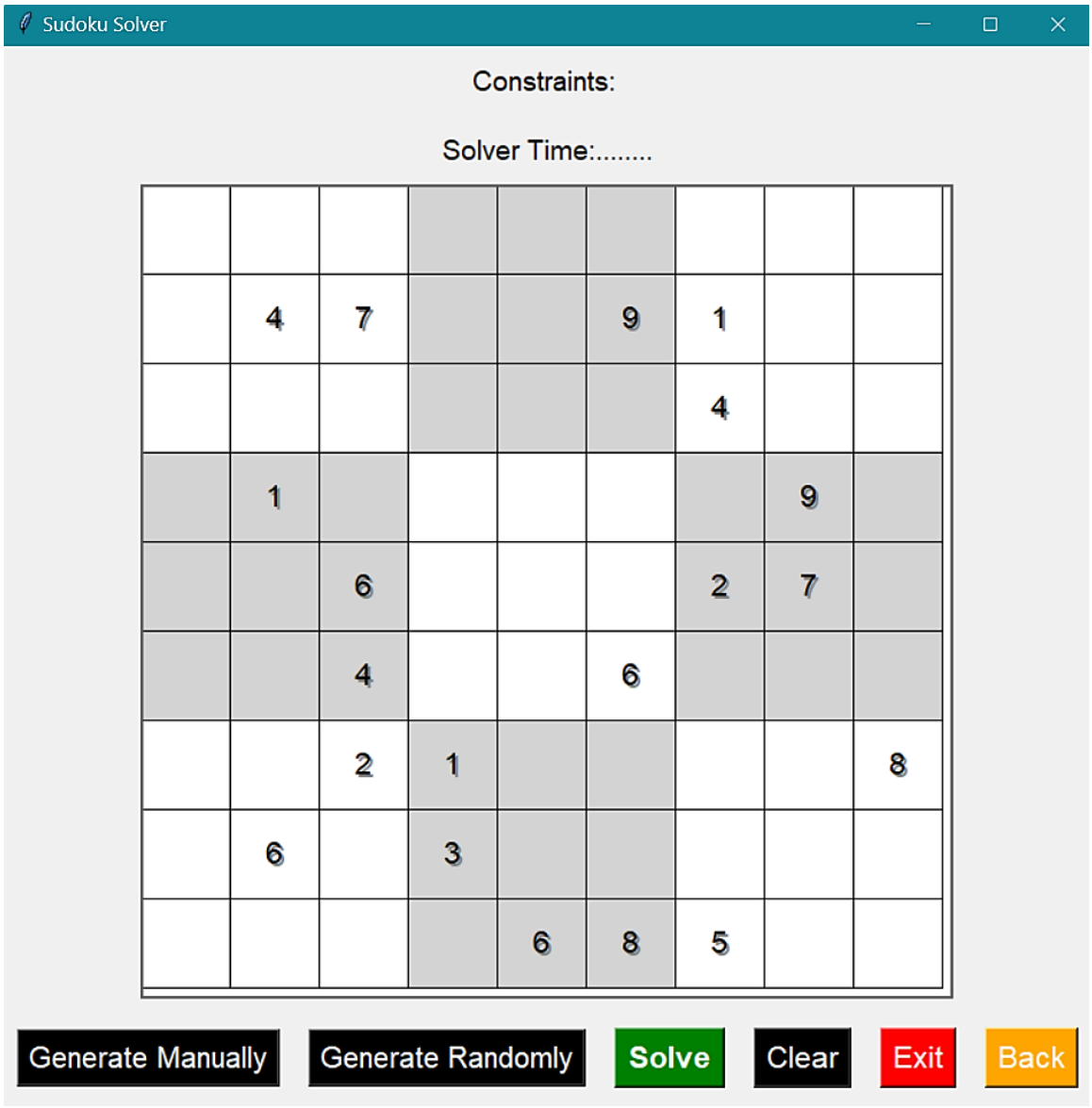
AI Mode GUI: Select Generate Manually



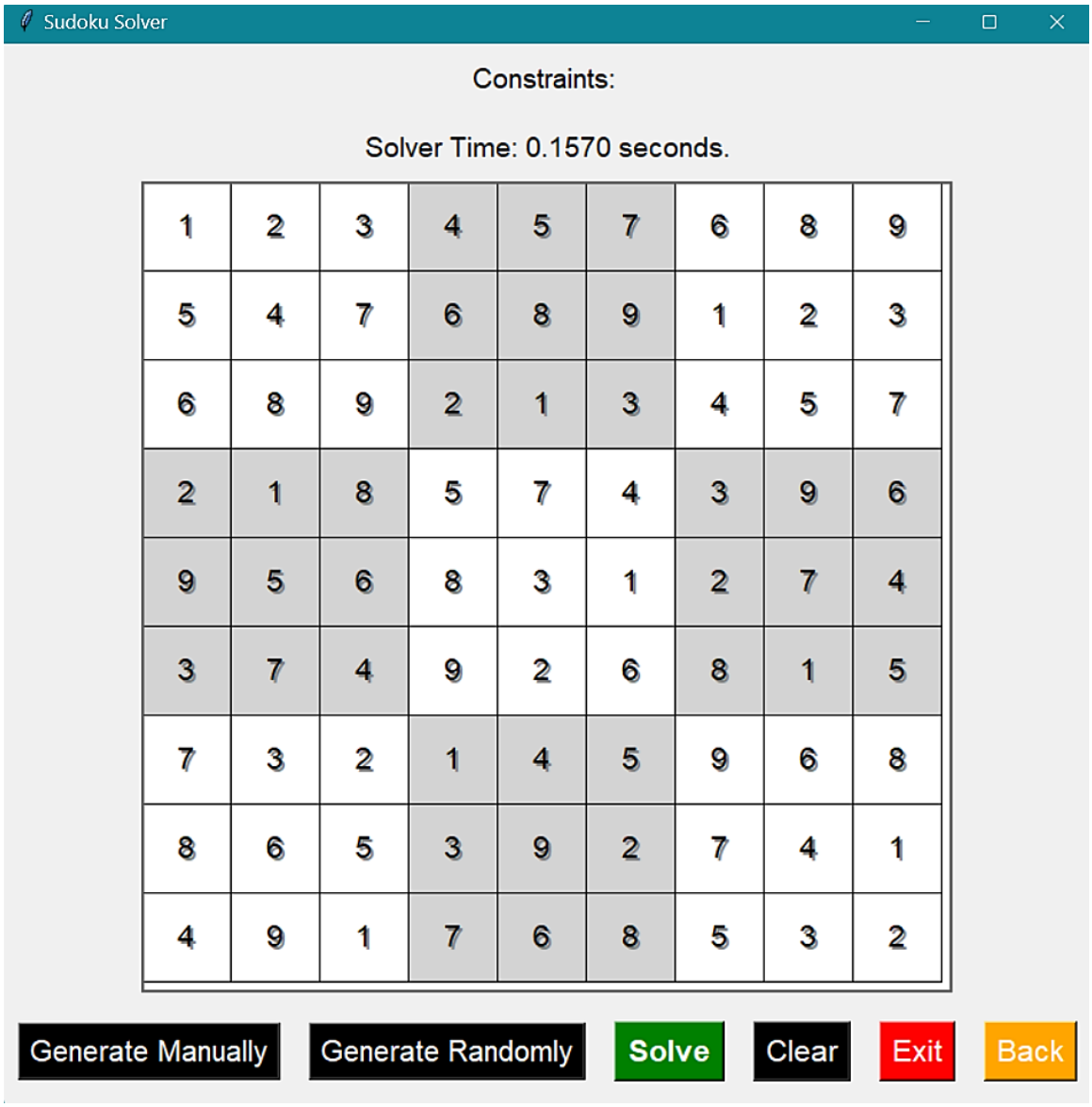
AI Mode GUI: Select Solve



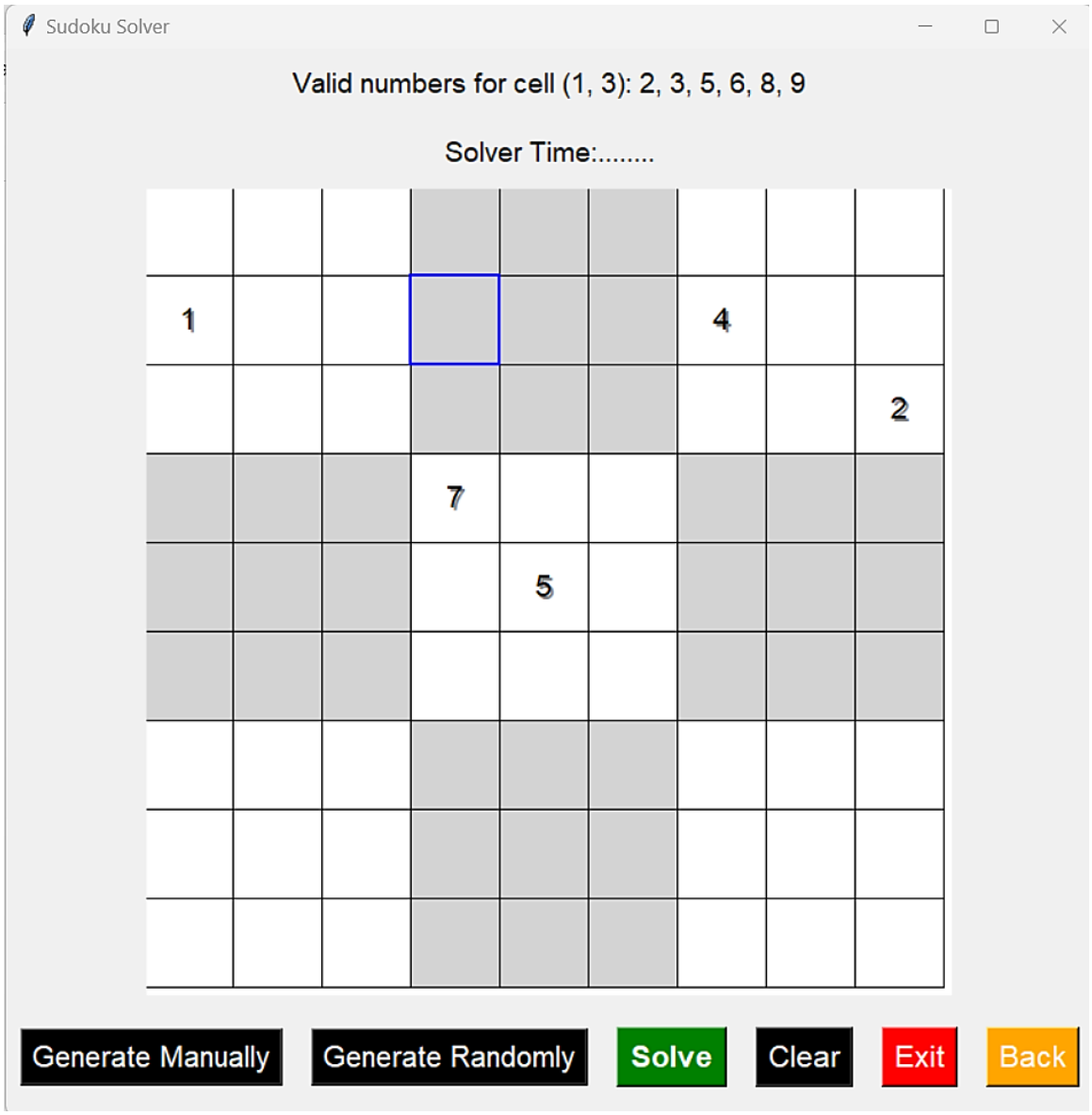
Mode Selection GUI: Select Hard Level in AI Mode



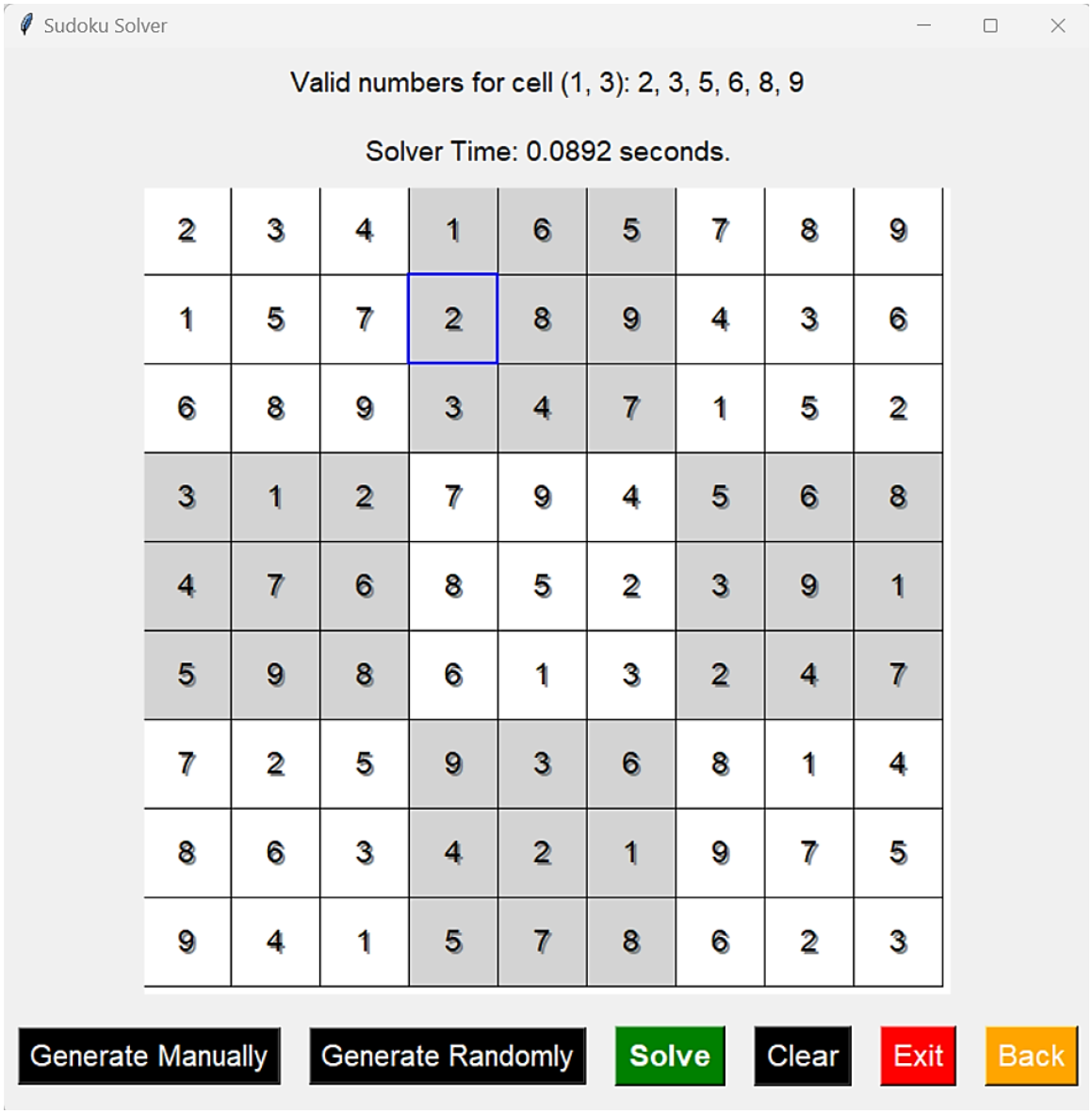
AI Mode GUI: Select Generate Randomly



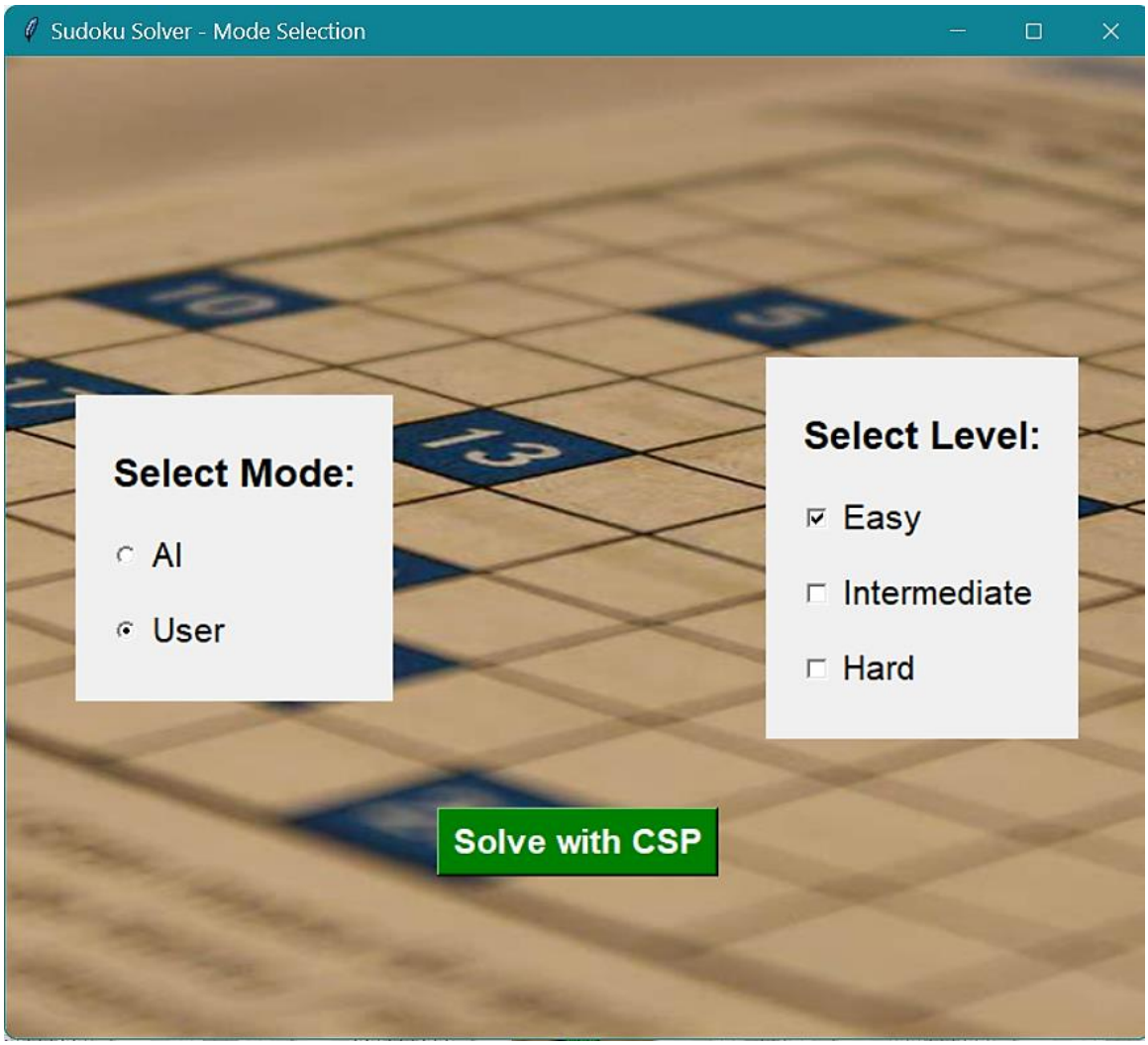
AI Mode GUI: Select Solve



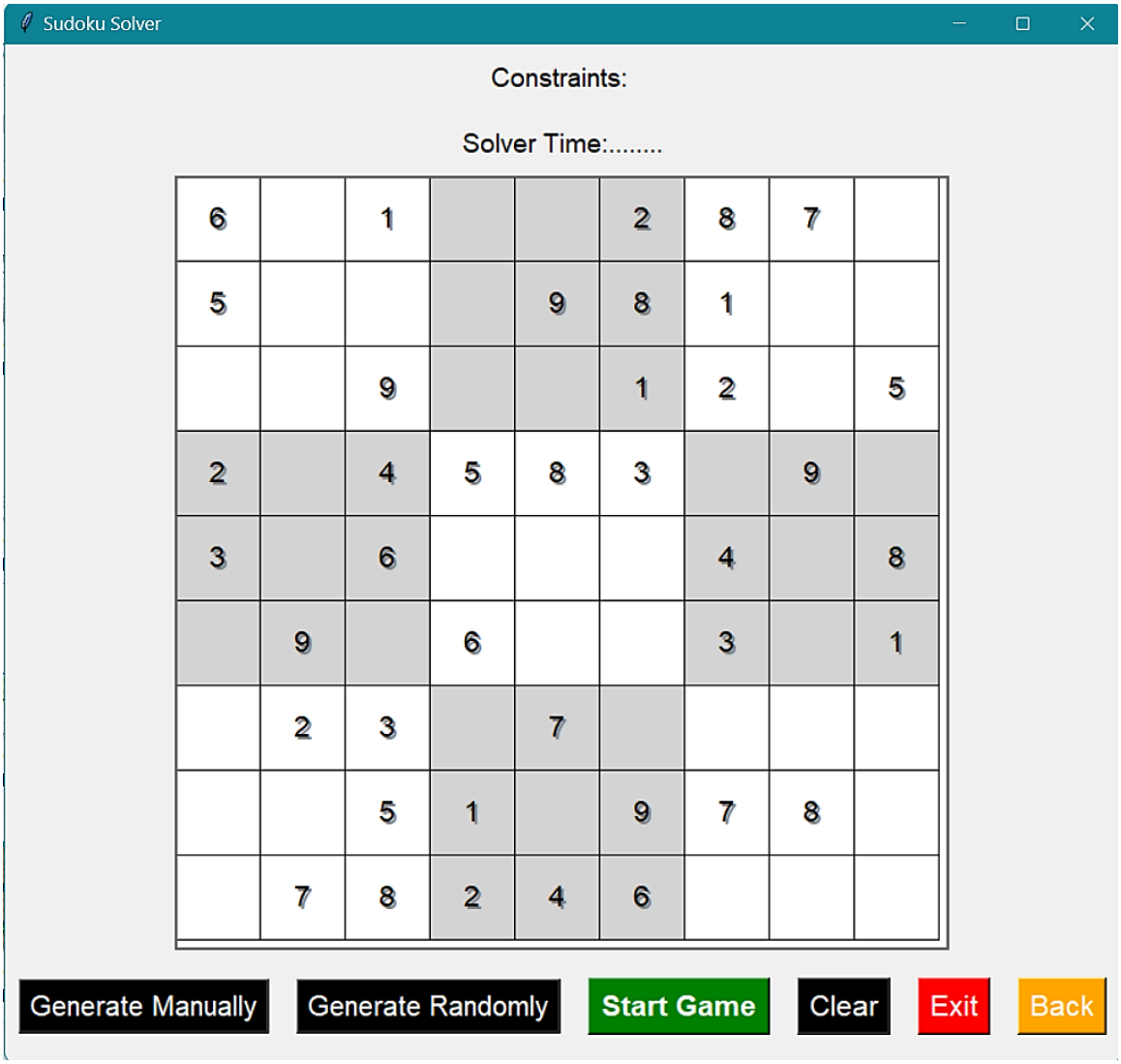
AI Mode GUI: Select Generate Manually



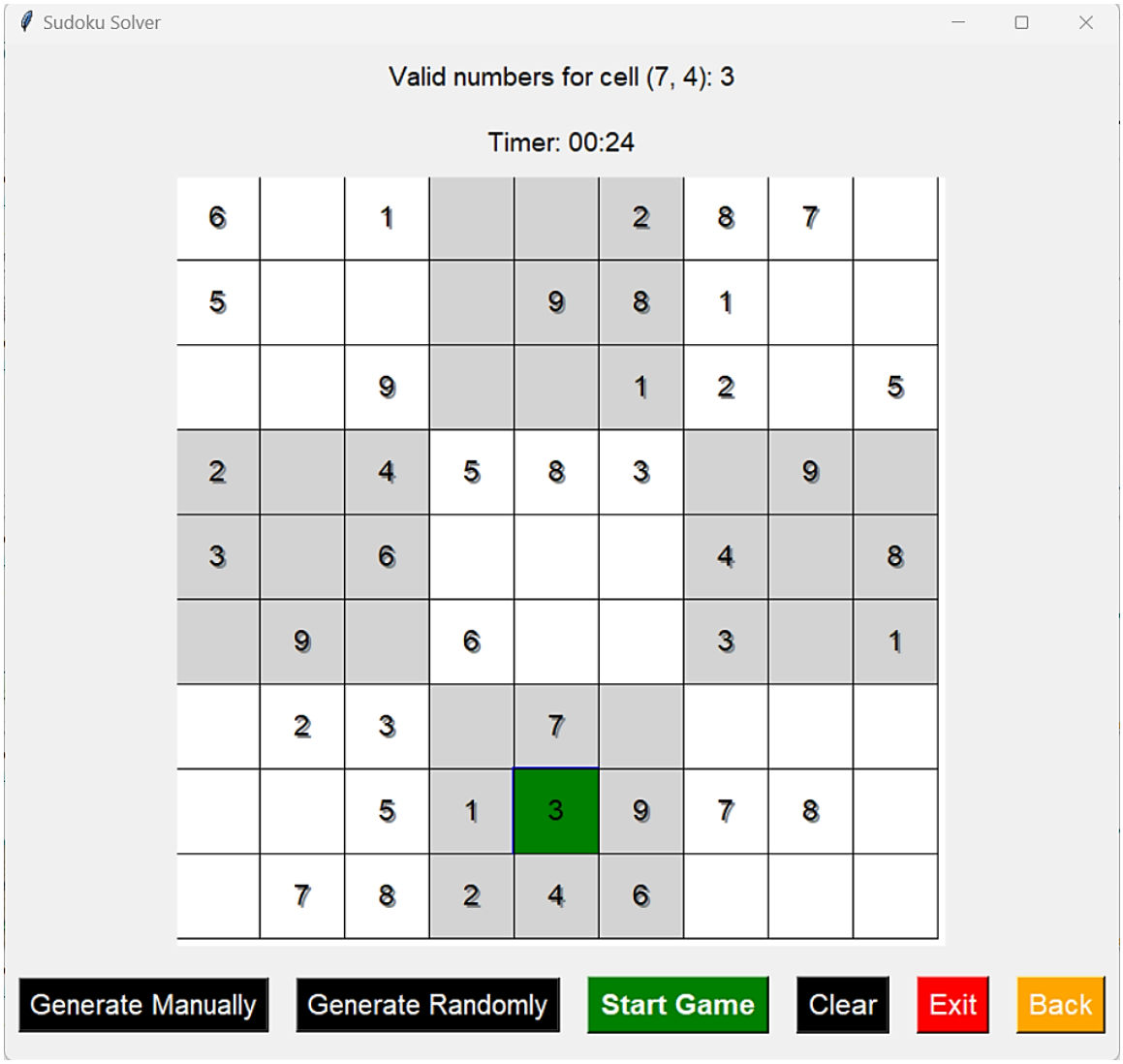
AI Mode GUI: Select Solve



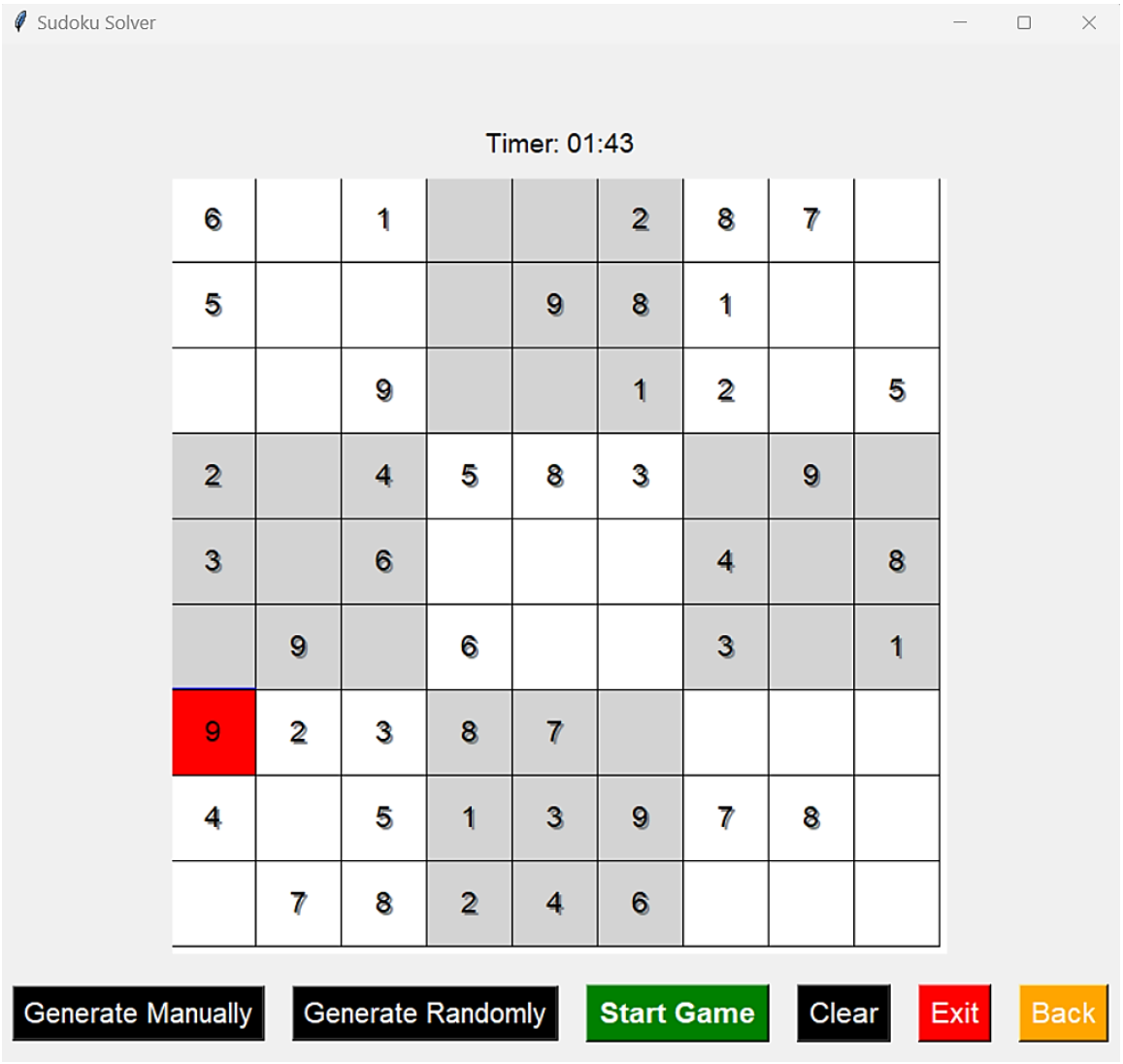
Mode Selection GUI: Select User Mode



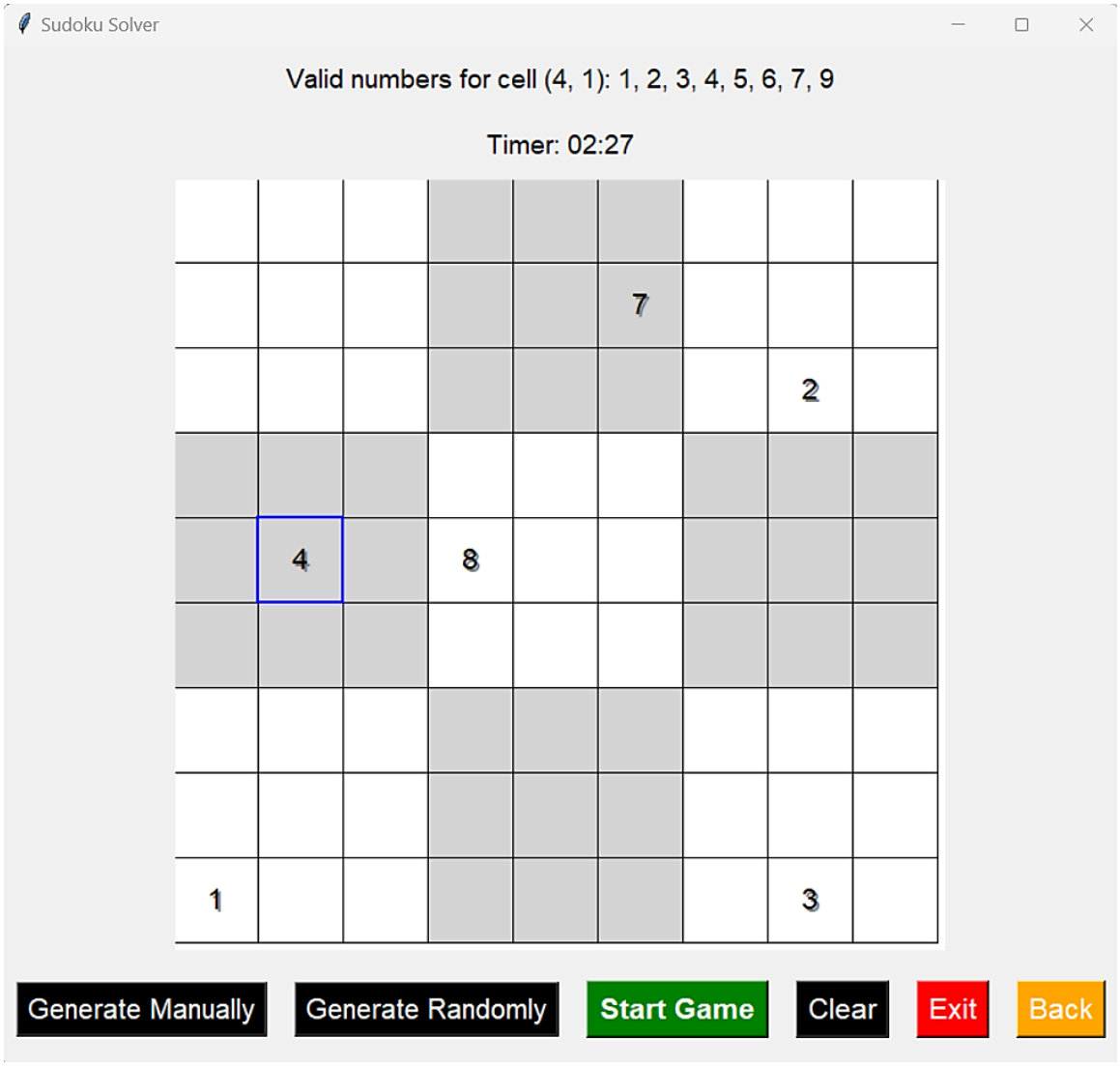
User Mode GUI: Select Generate Randomly



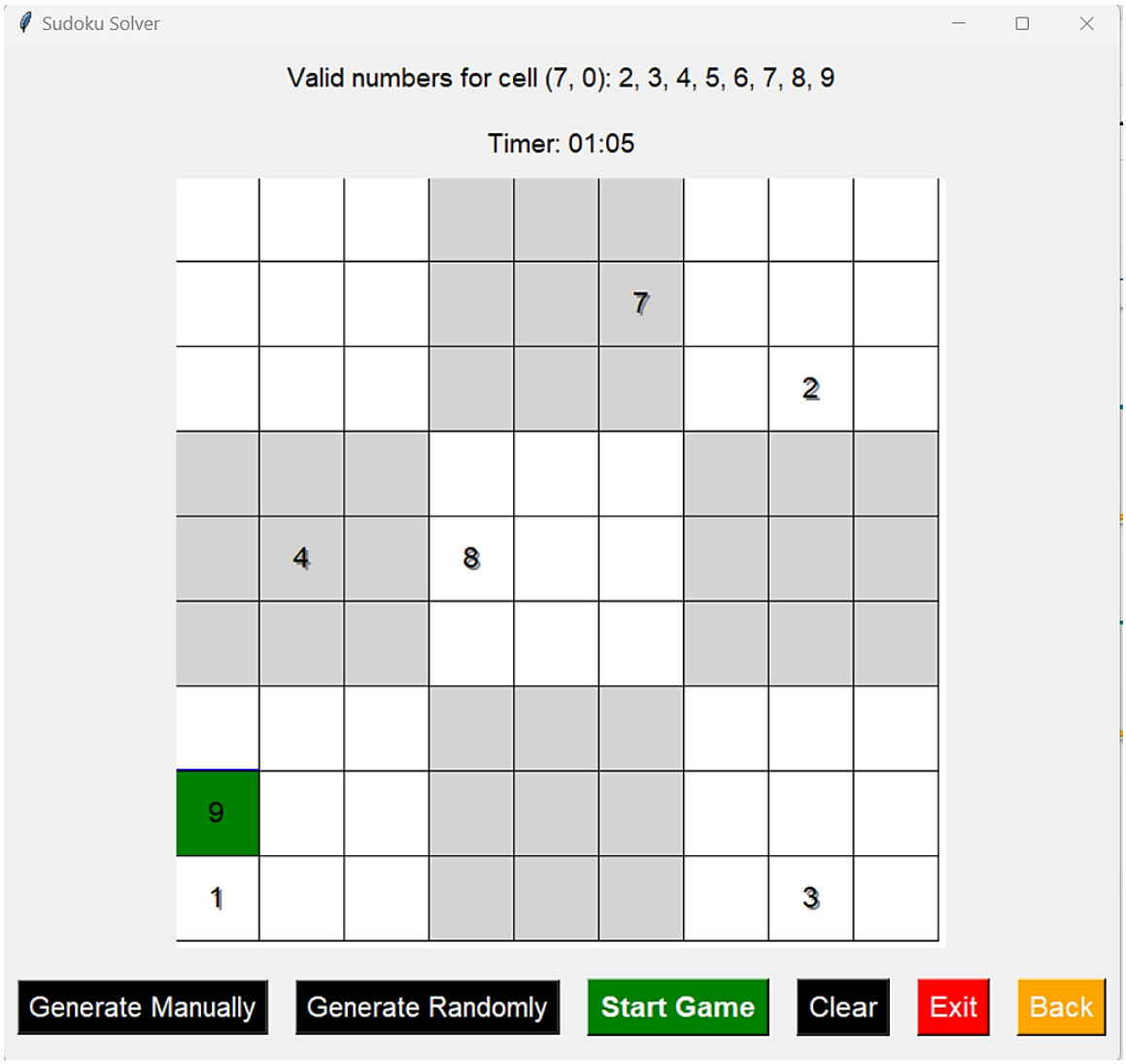
User Mode GUI: Select Start Game and Enter a Correct Value



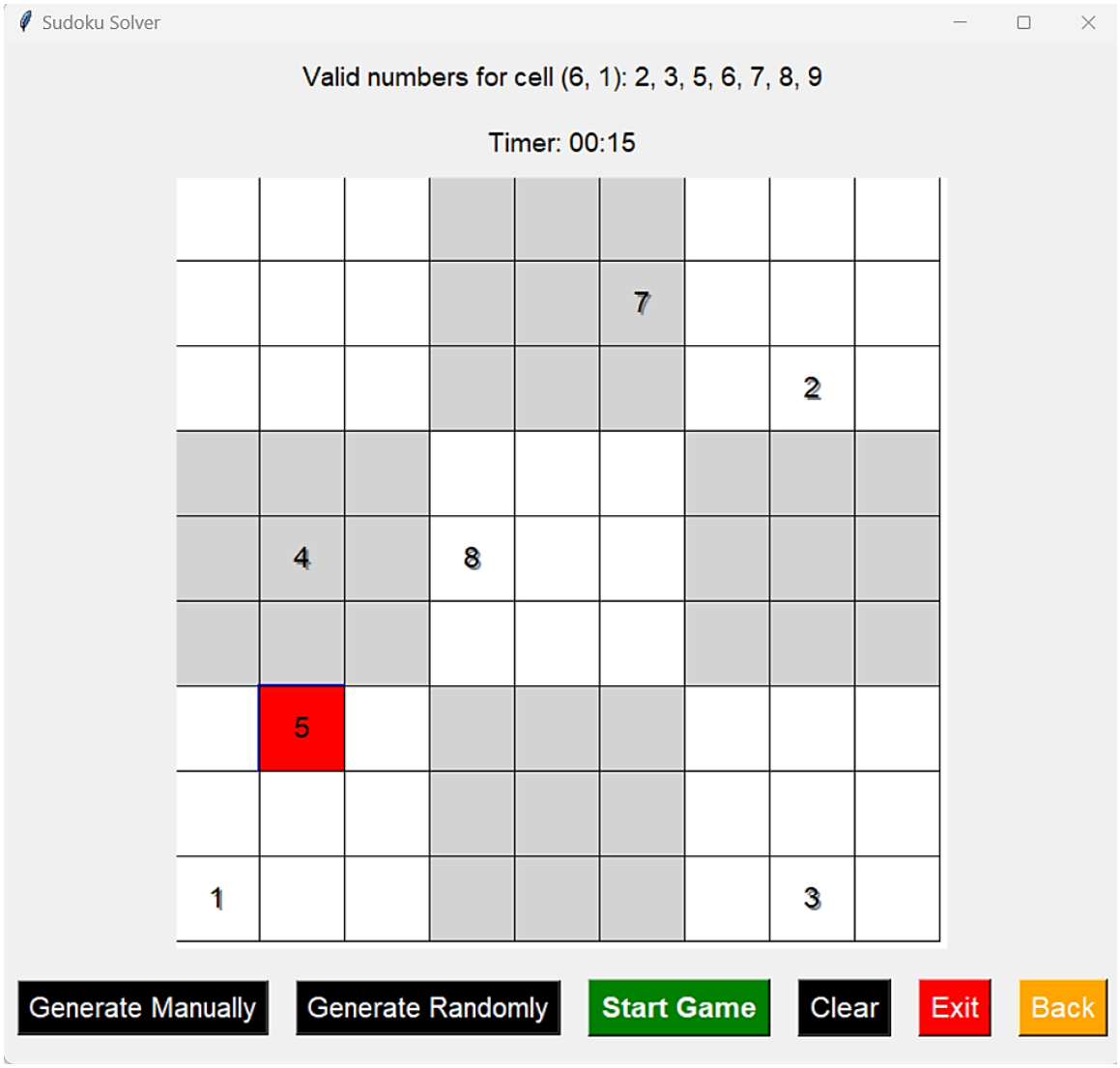
User Mode GUI: Enter a Wrong Value



User Mode GUI: Select Generate Manually



User Mode GUI: Select Start Game and Enter a Correct Value



User Mode GUI: Enter a Wrong Value

```
step 1 :
    Queue size: 40
    Arc: ( 1  ,  0 ) -> ( 0  ,  0 )
    Domains:  [1, 2, 3, 4, 5, 6, 7, 8, 9]  ->  [2]
    Remove [2] From ( 1  ,  0 )
    New Domain: [1, 3, 4, 5, 6, 7, 8, 9]
```

Terminal Printing Shape

➢ Assumptions:

- **Valid Input:** The project assumes that input Sudoku puzzles are valid and adhere to Sudoku rules (no duplicate numbers in rows, columns, or subgrids).
- **Solvable Puzzles:** It assumes that all generated puzzles are solvable, either by the AI agent or by user input (for Mode 2).
- **Consistent Constraints:** The algorithms rely on consistent constraints and assume that the initial puzzle setup does not violate Sudoku rules.
- **Arc Consistency Success:** The success of arc consistency assumes that the initial domain reduction and subsequent revisions lead to a consistent and solvable puzzle.

➢ Necessary Clarifications:

- **Initial Domain Reduction:** Clarify the process of reducing domain values for pre-filled cells and initializing domains for empty cells based on the initial puzzle.
- **Arc Consistency Process:** Provide detailed explanations of how arc consistency is applied, including revisions, queue management, and domain updates.
- **Heuristic Choices:** Explain why specific heuristics like MRV (Minimum Remaining Values) and LCV (Least Constraining Value) are chosen and how they contribute to solution efficiency.
- **Performance Metrics:** Clarify the metrics used for performance evaluation, such as solving time, number of backtracks, and comparison across different puzzle complexities.

➢ Extra Work:

- **Interactive Features:** If implemented, describe any interactive features that enhance user engagement, such as real-time feedback on input validity or step-by-step puzzle solving visualization.
- **GUI Enhancements:** Detail any GUI enhancements beyond basic functionality, such as user-friendly interfaces, error handling, and visual representations of solving processes.
- **Optimization Strategies:** Discuss any optimization strategies or algorithmic improvements implemented to enhance solving speed or reduce resource usage.
- **Error Handling:** Explain how errors or invalid inputs are managed within the program, ensuring robustness and user-friendly behavior.

➢ Conclusion

In conclusion, the implementation of a Constraint Satisfaction Problem (CSP) solver for Sudoku has been successfully achieved. Through the utilization of various algorithms and data structures, including Backtracking, Arc Consistency, and heuristics like Minimum Remaining Values (MRV) and Least Constraining Value (LCV), we have developed a robust Sudoku solver capable of tackling puzzles of varying complexities.

The project's primary objectives were met, including the development of a graphical user interface (GUI) that demonstrates the AI agent's ability to solve Sudoku puzzles in both interactive and automated modes. The interactive mode allows users to input custom Sudoku puzzles and verify their validity against the game rules, providing immediate feedback. Meanwhile, the automated mode highlights the AI agent's proficiency in solving Sudoku puzzles using advanced CSP techniques.

The key algorithms implemented, such as Backtracking for puzzle validation and generation, and Arc Consistency for constraint propagation and domain reduction, significantly contribute to the solver's efficiency and accuracy. These algorithms,

combined with appropriate data structures like matrices to represent the Sudoku grid and lists to manage domains and constraints, form the backbone of the solver's functionality.

Furthermore, the report delves into the details of the implemented algorithms, providing insights into how each algorithm contributes to the overall solving process. Sample runs and Arc Consistency trees are presented, demonstrating the solver's step-by-step approach to solving Sudoku puzzles and maintaining arc consistency throughout.

Additionally, the report includes a comparison of different initial boards' complexities (easy, intermediate, hard) and the corresponding time required to solve each puzzle. This analysis sheds light on the solver's performance under varying puzzle difficulties and highlights its adaptability and efficiency in handling different scenarios.

Assumptions made during the development process, such as the representation of Sudoku as a CSP with defined variables, domains, and constraints, are clarified within the report. Any extra work beyond the specified requirements, such as additional optimizations or user interface enhancements, is also outlined, highlighting the project's comprehensive approach and potential for further development.

Overall, the Sudoku CSP solver project has been a significant endeavor, combining theoretical concepts of constraint satisfaction with practical implementation to create a functional and versatile tool for solving Sudoku puzzles. The project's success opens avenues for future enhancements, algorithm optimizations, and potential applications in broader problem-solving domains beyond Sudoku.