SUDOKO SOLVER

Group Members:

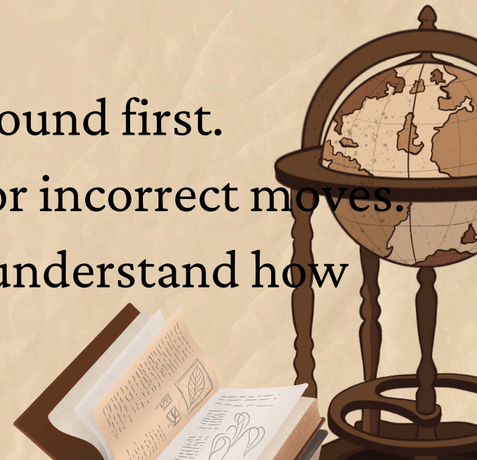
Vadlamudi Jyothsna(B23CS1076)

Shruti Sunil Vibhute(B23CS1070)

Yaddanapudi Gnani Prakash(B23CS1080)

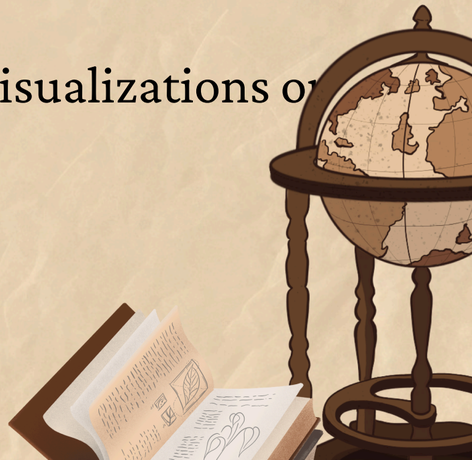
Objective:

- The primary goal of the Sudoku Solver project is to create a comprehensive application that caters to both Sudoku enthusiasts and learners by offering a versatile tool for:
- Generating Puzzles: Automatically create Sudoku puzzles of varying difficulties, providing an endless source of challenges for users.
- Solving Puzzles: Implement four distinct algorithms to solve Sudoku puzzles, showcasing the different computational strategies involved:
- Backtracking: A classic brute-force approach using depth-first search.
- Minimum Remaining Values (MRV): An enhancement of backtracking using heuristic methods to prioritize cells with fewer options.
- A Search Algorithm*: Combines the concepts of cost and heuristic search to find the most optimal path to the solution.
- Breadth-First Search (BFS): Explores solutions level by level, ensuring the shortest path is found first.
- Playing Mode: Allows users to manually solve puzzles, with real-time feedback for correct or incorrect moves.
- Visualization: Offers a dynamic visual representation of the solving process, helping users understand how different algorithms operate and compare their efficiency.



Problem Statement:

- Manual Sudoku Solving: For complex Sudoku puzzles, manual solving can be time-consuming and prone to errors, especially for beginners.
- Lack of Educational Tools: Many Sudoku applications can solve puzzles but do not provide insights into how the solution is reached. Users miss the opportunity to learn and understand the logic behind various solving techniques.
- Need for Algorithm Comparison: With multiple approaches to solving Sudoku, it is often unclear which algorithm is the most efficient for different puzzle complexities. A tool that can visualize and compare these algorithms can offer valuable learning experiences.
- User Engagement: Existing Sudoku solvers lack interactive elements like real-time solving visualizations or a play mode, reducing engagement and educational value.



Methodology:

- User Interface Development:
- The application is built using RAD Studio, providing a visually appealing and user-friendly interface.
- Users can easily navigate between puzzle generation, solving options, and play mode using a simple menu structure.
- Responsive design elements ensure smooth interactions and user engagement.
- Algorithm Implementation:
- Backtracking:
  - Uses a recursive, depth-first search approach to try all possible values until the correct one is found.
  - Efficient for smaller or less complex puzzles but can be slow for more challenging grids.
- Minimum Remaining Values (MRV):
  - An advanced version of backtracking that selects the cell with the fewest possible values first, reducing the overall search space and improving efficiency.
- A Search Algorithm*:
  - Utilizes a heuristic function (e.g., the number of filled cells) combined with a cost function to determine the most promising path towards the solution.
  - A more sophisticated approach, optimal for finding solutions with fewer steps.
- Breadth-First Search (BFS):
  - Explores all potential solutions level by level, ensuring the first found solution is the shortest.
  - Suitable for simpler or medium-complexity puzzles, but may struggle with time efficiency for larger grids.
- Data Structures Used:
- Vectors: Dynamic arrays that store the Sudoku grid and allow flexible data manipulation.
- Hash Maps: Keep track of potential values for each cell, speeding up constraint checking and reducing computation time.
- Priority Queues: Used in A* to prioritize nodes with the lowest combined cost and heuristic score.
- Queues: Implement BFS, maintaining nodes in a FIFO order to explore the puzzle level by level.
- Visualization and Play Mode:
- The solving process is visualized step-by-step on the screen, allowing users to follow the algorithm's decision-making process.
- In play mode, users can solve puzzles manually, receiving instant feedback on the correctness of their inputs.

Results:

- Application Performance:
- The Sudoku Solver successfully generates, solves, and allows users to play Sudoku puzzles across various difficulty levels.
- The visualization feature provides a clear and engaging display of each algorithm's decision-making process, making it easier for users to follow and learn from.
- Algorithm Efficiency:
- Backtracking:
  - Solves straightforward puzzles effectively but can be inefficient for complex grids due to its exhaustive search approach.
- MRV:
  - Significantly reduces the solving time for complex puzzles by narrowing down the search space, proving to be more efficient than plain backtracking.
- A Search*:
  - Offers the most optimal solutions, especially in cases with multiple possibilities, but requires more computational resources.
- BFS:
  - Performs well for simpler grids but may take longer for more complex puzzles due to its exhaustive, level-wise exploration.
- User Engagement and Feedback:
- Users appreciated the play mode, which provides a hands-on experience, as well as the dynamic visualizations that enhance understanding of the algorithms.
- Screenshots:
- Include visuals of the main menu, solving process, and play mode to illustrate the application's features.

Conclusion:

- The Sudoku Solver application successfully meets its objective of providing a versatile tool for solving, generating, and playing Sudoku puzzles. It stands out due to its algorithm visualization feature, offering an educational perspective by showcasing the step-by-step solving process.
- By incorporating multiple algorithms, the application provides a unique platform for comparing the efficiency of different solving techniques, making it valuable for both learners and enthusiasts.
- The project demonstrates how various data structures and algorithmic approaches can be leveraged to tackle a common logic puzzle effectively, offering insights into practical problem-solving strategies.

Challenges:

- Performance Optimization: Balancing speed and accuracy across various solving algorithms, especially for complex puzzles.
- Smooth Visualization: Ensuring real-time updates without lag, particularly for computation-heavy algorithms.
- User Interface Design: Creating an intuitive and engaging interface while accommodating multiple features.
- Edge Case Handling: Managing scenarios like multiple solutions, no solution, and incorrect inputs effectively.