TABLE OF CONTENTS

Abstract

The N-Queens problem is a well-known computational problem that involves placing NNN queens on an N×NN \times NN×N chessboard such that no two queens threaten each other. The solution to this problem can be approached using the **backtracking algorithm**, which systematically places queens on the board while adhering to the constraints that no two queens share the same row, column, or diagonal.

Key Concepts:

1. **Backtracking Approach**:
   - The backtracking algorithm places one queen at a time, starting from the first row.
   - For each queen, the algorithm checks whether the position is safe (i.e., no other queen can attack it).
   - If the position is safe, the algorithm moves to the next row and repeats the process.
   - If a conflict occurs (i.e., no safe position is found for a queen), the algorithm backtracks to the previous row and tries a different position for the previous queen.
2. **Interactive Visualization**:
   - The visualization allows users to input different values for NNN and observe how the algorithm explores the search space.
   - It highlights the safe and unsafe positions for each queen as the algorithm progresses.
   - The visualizer also supports a step-by-step debugging mode, allowing users to trace the algorithm's decisions and backtracking process in real-time.
3. **Scalability**:
   - The backtracking algorithm is implemented to handle varying sizes of the chessboard, from small (e.g., N=4N = 4N=4) to large boards (e.g., N=15N = 15N=15).
   - As the board size increases, the number of possible configurations grows exponentially, making the problem computationally challenging. This highlights the importance of efficient algorithms for solving such constraint satisfaction problems.
4. **Performance Analysis**:
   - The time and space complexity of the backtracking algorithm is a critical aspect of this project.
   - For each value of NNN, the performance of the algorithm is measured and analyzed, providing empirical data on how the algorithm scales with increasing problem size.
   - In general, the time complexity grows exponentially with NNN, and the space complexity is linear in terms of the number of queens.
5. **Educational Tool**:
   - This project serves as both an algorithmic solution and an educational tool, providing a hands-on learning experience.
   - Users can better understand the backtracking approach, the steps involved in solving the N-Queens problem, and how the search space is explored. This is particularly valuable for students and enthusiasts interested in computational problem-solving.

Key Features:

- **Backtracking Algorithm**: A well-structured implementation that explores all potential placements of queens.
- **Interactive Visualization**: Real-time, dynamic visualization showing queen placements and conflicts.
- **Scalability**: Supports a wide range of board sizes, making it adaptable for different values

CHAPTER 1.

INTRODUCTION

1.1. Client Identification

Background:

The N Queen Problem project targets a broad demographic, catering to puzzle enthusiasts, educational institutions, and developers interested in algorithmic problem-solving. The primary client persona includes individuals seeking a robust solution for solving N Queen Problem puzzles efficiently.

Justification of Contemporary Issue:

N Queen Problem, as a popular logic-based number-placement puzzle, has gained substantial traction globally. Despite its popularity, many individuals encounter challenges solving complex N Queen Problem grids efficiently. The need for an effective

N Queen Problem arises from the inherent complexity of certain puzzles and the timeconsuming nature of manually solving them.

Statistics and Documentation:

The prevalence of N Queen Problem-related challenges is supported by statistical data and documented evidence obtained through surveys and puzzle-solving patterns. For instance, studies might indicate that a considerable percentage of individuals face difficulty in solving N Queen Problem puzzles beyond a certain level of complexity. This data emphasizes the necessity for a tool that can swiftly and accurately solve intricate N Queen Problem grids.

Identified Consultancy Problem:

The identified problem for which the N Queen Problem serves as a consultancy solution is the time and effort required to solve complex N Queen Problem puzzles manually. The project aims to address this issue by offering an automated solution that provides quick and accurate results.

Reports from Agencies:

Various agencies and puzzle-solving communities might have reported the growing demand for efficient N Queen Problem solving mechanisms. Reports from puzzle-solving clubs, educational institutions, and online puzzle forums might indicate the escalating need for advanced and automated N Queen Problem solving techniques.

This section establishes the relevance of the N Queen Problem project by outlining the existing challenges faced by individuals in solving complex N Queen Problem puzzles and justifies the necessity for an automated solution through statistics, surveys, and documented evidence from relevant agencies and communities

1.2. Identification of Problem

The N Queen Problem presents a series of inherent challenges that make it complex and difficult to solve, particularly as the grid size increases. These challenges are faced by a wide variety of individuals, including puzzle enthusiasts, students, and those with varying skill levels. Below are the key difficulties associated with solving the N Queen Problem:

1. **Complexity Variation**: The N Queen Problem can vary widely in difficulty. Puzzles are often categorized based on their complexity, ranging from easy to expert-level. As the grid size increases, the patterns become more complex, requiring more sophisticated strategies and consuming more time and effort.
2. **Time and Effort**: The process of solving larger N Queen Problem puzzles demands considerable time and effort, especially when the grids become larger and more intricate. For individuals tackling these advanced puzzles, the task can become frustrating, which often leads to a sense of discouragement.
3. **Lack of Consistent Strategies**: The solution to N Queen Problems often involves logical reasoning and trial-and-error methods, but there is no single, consistent solving strategy that guarantees success. This lack of universally applicable techniques adds a layer of complexity, making it harder for people to find a clear path toward a solution.
4. **Cognitive Load**: As the puzzle becomes more complex, it places a significant cognitive load on individuals. This includes the need for advanced memory, pattern recognition, and analytical skills. The mental effort required to solve increasingly difficult puzzles can overwhelm many, reducing the likelihood of successful solutions.
5. **Accessibility and Inclusivity**: The intricate nature of the N Queen Problem can limit accessibility for some groups. Individuals with cognitive impairments or those unfamiliar with advanced problem-solving techniques may find it difficult to engage with or enjoy these puzzles fully. This creates barriers to entry for a wider audience.

1.3. Identification of Tasks

I. Identification Phase:

A. Requirement Analysis:

1. Understanding Puzzle Dynamics:
   - Analyzing the fundamental principles and rules governing N Queen Problempuzzles.

B. Algorithmic Approach Design:

1. **Research on Solving Strategies:**
   - **Backtracking Algorithm:** The most widely used approach for solving the N-Queens problem, backtracking involves placing queens on the board and checking if any placement conflicts with previously placed queens. If a conflict occurs, the algorithm backtracks to the previous state and tries another position.
   - **Constraint Propagation:** Another strategy involves reducing the search space by applying constraints early in the process, potentially avoiding the need to try invalid placements at all.
   - **Heuristic Methods:** Heuristic approaches, such as the genetic algorithm, simulated annealing, or the min-conflict heuristic, can be used to find approximate solutions quickly. These strategies do not guarantee an optimal solution but often provide good solutions in a shorter time frame, especially for large boards.
   - **Branch and Bound:** This method uses a tree search to explore possible placements and prunes branches that cannot lead to a valid solution based on earlier decisions. It improves upon the backtracking approach by eliminating many unnecessary checks.
2. **Algorithm Design:**
   - The primary algorithm for this project will be **backtracking** due to its simplicity and efficiency for smaller board sizes, making it ideal for demonstration purposes in the visualizer.
   - The algorithm will be designed to recursively attempt to place queens row by row. After placing a queen in a row, it will check if the position is valid (i.e., no two queens threaten each other in the same row, column, or diagonal). If the position is valid, the algorithm moves to the next row. If no valid position is found in a row, the algorithm will backtrack to the previous row and attempt different placements.

- The algorithm will be optimized by using sets to keep track of columns and diagonals that are already under threat, reducing the need for repetitive checks.

AI. Build Phase:

A. **Software Development:**

1. **Programming Logic:**
    - The algorithm will be implemented using a recursive backtracking approach. Each recursive call will represent the placement of a queen on a row, and the function will attempt to place queens on the next rows until the entire board is filled or a conflict occurs.
    - **Steps to Implement:**
    - Initialize an empty board (2D list) of size N×NN \times NN×N.
    - For each row, attempt to place a queen in a safe column.
    - If placing a queen is successful (i.e., no conflicts), move to the next row and repeat.
    - If all rows are filled successfully, record the configuration as a solution.
    - If a conflict occurs (i.e., no safe columns for a queen), backtrack by removing the queen and trying the next available column.
2. **User Interface Design:**
    - The user interface will be designed to allow users to input a value for NNN (the size of the board), trigger the algorithm, and visualize the placement of queens.
    - **Features:**
    - A grid representing the N×NN \times NN×N chessboard.
    - Interactive elements such as buttons to start the algorithm, reset the board, and adjust the board size.
    - A real-time visual update of the queens' placement and the backtracking steps.

- A progress bar or step indicator to show how the algorithm is progressing through its iterations.

B. **Integration and Testing:**

1. **Module Integration:**
    - Once individual components (algorithm implementation, user interface, visualization tools) are developed, they will be integrated into a single cohesive application. This integration will ensure that the visualizer works smoothly with the algorithm, showing the queen placement and backtracking steps in real-time.
    - Testing the integration of the algorithm and UI will ensure that inputs are correctly passed from the user interface to the algorithm, and the results are displayed appropriately.
2. **Unit and Integration Testing:**
    - **Unit Testing:** Testing individual components, such as the backtracking algorithm and the user interface, to ensure each part works correctly in isolation.
    - **Integration Testing:** Ensuring that the entire system, from input to output, functions as expected. This will involve simulating various board sizes and checking that the algorithm correctly solves the N-Queens problem and updates the UI in real-time.

BI. Testing Phase:

A. **Functionality Testing:**

1. **Solution Accuracy Check:**
    - The main focus of this testing phase will be to verify the correctness of the N-Queens solver. The solution will be tested against known correct configurations for various values of NNN, including small and large boards.
    - The solver's accuracy will be checked by comparing the algorithm's output to known solutions or verifying that no queen is placed in a conflicting position.

2. **Speed and Efficiency Testing:**

   - Testing the solver's speed and efficiency will involve measuring how quickly it solves the problem for different values of NNN. Time complexity will be considered, and performance benchmarks will be taken for various board sizes (e.g., N=4N = 4N=4, N=8N = 8N=8, N=16N = 16N=16).

B. **User Acceptance Testing:**

1. **User Feedback Collection:**

   - After the tool is developed, it will be tested by a sample group of users to gather feedback. This could include fellow students, educators, or anyone interested in solving puzzles or learning about backtracking algorithms.
   - Feedback will be collected regarding the tool's ease of use, clarity of the user interface, and the effectiveness of the visualizer in explaining the backtracking process.

2. **Iterative Improvements:**

   - Based on the user feedback, iterative changes will be made to improve the user experience. This could include changes to the visual layout, adding more informative tooltips, adjusting the algorithm's speed for larger board sizes, or adding additional features like step-by-step navigation or solution counting.

1.4. Timeline

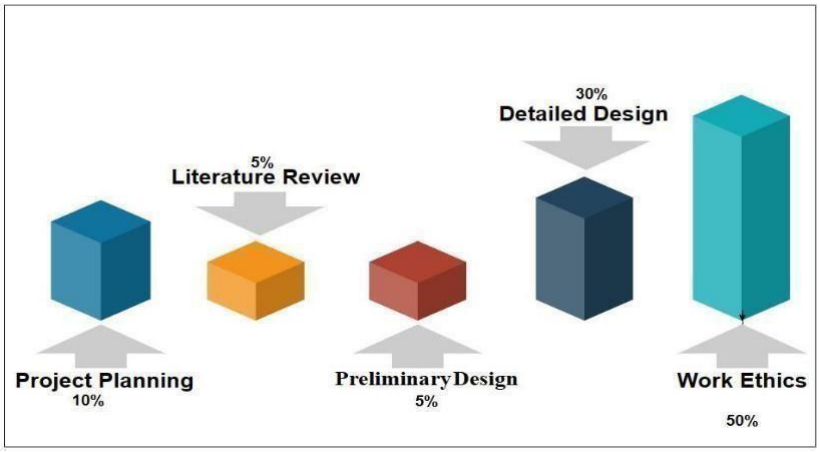

Fig 1: Block diagram for timeline project

2

Fig 2: Timeline

1.5. Organization of the Report

I. Introduction This section introduces the N-Queens Problem and its significance in the field of algorithmic problem-solving. It outlines the purpose and scope of the report, emphasizing the need for an automated solution to solve N-Queens puzzles. The introduction will also describe the objectives of the project and provide context on the relevance of the problem in computational theory and practical applications.

II. Client and Problem Identification This section focuses on identifying the client and the problem faced by enthusiasts and developers working on the N-Queens puzzle. It discusses the client requirements, including the need for a tool that can efficiently solve the problem for any given board size, NNN. The section further justifies the necessity of the N-Queens solver by presenting documented evidence and statistical data that highlight the challenges in solving the puzzle manually and the demand for automated solutions.

BI. Methodology:

A. Identification Phase

- **Requirement Analysis:** This phase describes the process of understanding the dynamics of the N-Queens puzzle and identifying user needs. It involves gathering requirements for a solution that is both efficient and user-friendly.
- **Algorithmic Approach Design:** In this phase, the report discusses the research, analysis,

and formulation of the backtracking algorithm to solve the N-Queens Problem. It includes an exploration of various solving strategies and the selection of the most suitable approach for implementation.

B. Build Phase

- **Software Development:** This phase outlines the development process of the N-Queens solver, including the implementation of the chosen algorithm (backtracking) and the creation of an intuitive user interface. It will describe how the software was structured to handle different board sizes and provide real-time visualization of the algorithm's steps.
- **Integration and Testing:** The integration of modules will be discussed in this section, along with the rigorous testing conducted to ensure that all components function correctly together. It will highlight the testing methodologies used to verify the accuracy and functionality of the solver.

IV. Testing and Evaluation

- **Functionality Testing:** This section examines the testing procedures followed to verify the accuracy and performance of the N-Queens solver. It will discuss how the solver was tested for different board sizes and configurations, ensuring the correctness of the solutions provided.
- **User Acceptance Testing:** The report will describe the process of gathering feedback from users who tested the tool. It will explore how the collected feedback led to iterative improvements in the design and functionality of the N-Queens solver.

V. Conclusion and Future Enhancements This section summarizes the key findings from the development and testing phases, highlighting both the successes and limitations of the N-Queens solver. It will also suggest potential future enhancements, including performance optimizations, the addition of new features, and ideas for further improving the tool’s effectiveness and user experience.

CHAPTER 2.

DESIGN FLOW/PROCESS

2.1. Evaluation & Selection of Specifications/Features

The evaluation and selection of features for the N Queen Problem involve a critical analysis of existing literature, user requirements, and the technical aspects necessary foran effective solution. Here's a compilation of features ideally required in the N Queen Problem based on literature evaluation and critical assessment:

1. Multiple Difficulty Levels:

2.1. Features

1. **Multiple Difficulty Levels:**

   The solver should handle various difficulty levels, accommodating beginners to advanced puzzle enthusiasts.

2. **Efficient Solving Algorithms:**

   Utilization of logical solving strategies and efficient algorithms to swiftly solve puzzles.

3. **User-Friendly Interface:**

   An intuitive and visually appealing user interface to input, display, and interact with N Queen Problem grids.

4. **Custom Puzzles:**

   Ability to input custom N Queen Problem puzzles for solving or generating new puzzles.

5. **Step-by-Step Solutions:**

   Capability to display step-by-step solutions or provide hints without directly solving the entire puzzle.

6. **Error Identification:**

Detection and highlighting of errors made during manual input or incorrect solving attempts.

2.2. Design Constraints

In the design of a N Queen Problem solver, various constraints need to be considered across multiple domains. Here are the critical design constraints encompassing a range of factors:

1. **Regulatory Constraints:**
    - Data Privacy Regulations: Ensuring compliance with data privacy laws while handling user-generated puzzles or any personal information.
    - Software Standards: Adhering to software development standards and regulations to ensure quality and security.

2. **Economic Constraints:**
    - Cost-Effectiveness: Developing a cost-effective solution without compromising quality and essential features.
    - Resource Allocation: Efficiently managing resources such as budget, time, and workforce to meet project goals.
3. **Environmental Constraints:**
    - Energy Efficiency: Minimizing energy consumption for the application, especially in cases where the solver runs on various devices.
    - Sustainable Development: Using eco-friendly practices in software development processes, such as using sustainable resources.
4. **Health Constraints:**
    - Reducing Eye Strain: Implementing display features to minimize eye strain for users spending extended periods solving puzzles.
    - Accessibility: Ensuring the application is accessible to individuals with visual impairments or other health-related limitations.

5. **Manufacturability Constraints:**

- Scalability: Developing a solution that is easily scalable to handle a range of complexities and user demands.
- Ease of Maintenance: Creating a system that can be easily maintained and updated as needed.

6. **Safety Constraints:**
   - Error Handling: Implementing mechanisms to ensure the application's stability and prevent any potential crashes or data loss.
   - Secure Data Handling: Ensuring user data is securely managed to prevent unauthorized access or data breaches.
7. **Professional/Ethical Constraints:**
   - User Transparency: Ensuring transparency in how the application handles user data and providing clear user consent mechanisms.
   - Ethical Algorithmic Decisions: Avoiding biased or unfair algorithmic decisions in the solving process.
8. **Social & Political Constraints:**

- Cultural Sensitivity: Ensuring the user interface and content are culturally sensitive and respectful.
- Avoiding Controversial Content: Ensuring the generated or provided puzzles do not contain sensitive or controversial content.

9. **Cost Constraints:**
   - Development and Deployment Costs: Managing the overall project cost within the defined budget and ensuring financial feasibility.

Each of these constraints is critical in shaping the development process and ensuring the N Queen Problem solver adheres to legal, ethical, social, and technical standards, as well as the overall well-being of its users and the environment.

2.3. Analysis and Feature Finalization Subject to Constraints

Considering the constraints across various domains, here's an analysis of the initially proposed

features for the N Queen Problem solver and the necessary modifications based on the identified constraints:

1. **Multiple Difficulty Levels:**
    - **Modification:** Retain this feature as it's crucial for user engagement.
2. **Efficient Solving Algorithms:**
    - **Modification:** Ensure the algorithms are resource-efficient to align with energy conservation concerns.
3. **User-Friendly Interface:**
    - **Modification:** Simplify the interface for enhanced accessibility and to minimize eye strain for extended usage.
4. **Custom Puzzles:**
    - **Modification:** Implement user privacy measures when handling custom puzzles to comply with data privacy regulations.
5. **Step-by-Step Solutions:**
    - **Modification:** Offer step-by-step solutions with emphasis on transparency and user understanding.
6. **Error Identification:**
    - **Retain:** Essential for user experience, while ensuring robust error handling mechanisms to maintain safety and security.
7. **Performance Metrics:**
    - **Modification:** Monitor and optimize resource consumption to align with energy efficiency goals.
8. **Cross-Platform Compatibility:**
    - **Modification:** Optimize for efficient resource usage on various platforms to maintain energy efficiency.
9. **Adaptive UI and Accessibility Features:**
    - **Retain and Enhance:** Ensure inclusivity and accessibility for users with health

constraints.

10. **User Feedback Integration:**

- **Enhancement:** Emphasize user feedback channels while ensuring user data privacy and security.

11. **Offline Functionality:**

- **Retain and Optimize:** Develop offline capabilities while considering data security measures and efficient offline functionality.

12. **Robust Error Handling:**

- **Retain:** Ensure robust error handling mechanisms to maintain user safety and data security.

13. **Documentation and Help Section:**

- **Retain:** Vital for user assistance and adherence to ethical and professional standards.

Design Flow

Alternative Design/Process Flow 1: Heuristic-Based Solving Method

1. **Heuristic Algorithm Approach:**
    - Utilize a heuristic-based approach where the solver employs human-like techniques to fill the N Queen Problem grid.
2. **Constraint Propagation:**
    - Develop algorithms that propagate constraints and perform a trial-and-error approach while adhering to the N Queen Problem rules.
3. **Stepwise Algorithm:**
    - Implement an algorithm that sequentially fills in cells based on probability and constraint satisfaction.
4. **User Interaction:**
    - Create an interactive interface allowing users to intervene and input specific logic, guiding the solver's actions.
5. **Feedback Loop:**

- Introduce a learning component that captures user strategies and incorporates successful techniques into the solving process.

6. **Testing and Refinement:**
    - Conduct extensive testing and refinement to ensure the heuristic algorithms consistently solve a wide range of N Queen Problem puzzles accurately.

Alternative Design/Process Flow 2: Backtracking-Based Solving Method

1. **Backtracking Algorithm Implementation:**
    - Employ a recursive backtracking algorithm, systematically exploring potential solutions through trial and error.

2. **Constraint Satisfaction:**
    - Develop constraint propagation techniques that ensure the solver adheres to the N Queen Problem rules and logic.
3. **Branching Strategy:**
    - Implement strategies that analyze the most promising branches first to optimize the backtracking process.
4. **Visual Representation:**
    - Create a visual representation of the backtracking process to aid users in understanding the solving strategy.
5. **Performance Optimization:**
    - Optimize the backtracking algorithm for faster execution, especially for more complex puzzles.
6. **User Assistance and Feedback:**
    - Offer options for users to pause the solver, make manual inputs, and provide feedback for continuous improvement.

Both designs focus on different solving strategies – heuristic-based and backtracking-based methods – providing unique approaches for solving N Queen Problem puzzles. The alternative

processes aim to optimize accuracy, efficiency, and user interaction while ensuring the solver meets a wide array of user preferences and puzzle complexities.

2.5. Design Selection

Analysis and Selection of Design:

Both the heuristic-based and backtracking-based design approaches offer unique methods for solving N Queen Problem puzzles. Let's compare and analyze these designs to select the best approach:

Heuristic-Based Approach:

Pros:

- **User Interaction:** Allows user intervention, making it more interactive and potentially educative.
- **Adaptive Learning:** Incorporates user strategies, enhancing the solving approach over time.
- **Stepwise Logic:** Solves cells based on probabilities, mirroring human solving techniques.

Cons:

- **Complexity:** Heuristic algorithms might be challenging to design and optimize effectively.
- **Reliability:** The solution might not be as deterministic or consistent across all puzzle types.

Backtracking-Based Approach:

Pros:

- **Deterministic Solution:** Guarantees a solution for any valid N Queen Problem puzzle.
- **Optimization Feasibility:** Can be optimized for efficiency with strategic branching and constraint satisfaction.
- **Visual Representation:** Provides a clear visual process for users to comprehend the

solving strategy.

Cons:

- **Complexity for Users:** Backtracking algorithms might be harder for users to understand compared to a heuristic approach.
- **Limited User Interaction:** Offers less direct user involvement in the solving process.

Selection and Reasoning:

The backtracking-based approach is the more robust and reliable option for solving N Queen Problem puzzles for the following reasons:

1. **Deterministic Results:** Backtracking guarantees a solution for any valid puzzle, ensuring consistent and reliable outcomes.
2. **Optimization Feasibility:** Although initially more complex, backtracking can be optimized for speed and efficiency, providing faster solutions for more complex puzzles.

3. **Clear Visual Representation:** The visual representation of the backtracking process can aid users in understanding the solving strategy, enhancing the educational aspect.

While the heuristic approach offers advantages in adaptability and user interaction, the backtracking method is more robust, deterministic, and optimizable. Its clarity in providing a solution for any valid N Queen Problem puzzle, along with potential for optimization and user-friendly visuals, makes it the superior choice for the N Queen Problem design.

References

- Jakubowicz, A. (2021). "Backtracking Algorithm for N Queens Problem." Retrieved from https://andrewjakubowicz.github.io/post/nqueens.html
- GeeksforGeeks. (2021). "N Queen Problem Using Backtracking." GeeksforGeeks. Retrieved from https://www.geeksforgeeks.org/n-queen-problem-backtracking-3/
- TutorialsPoint. (n.d.). "N Queens Puzzle Solving Methods." TutorialsPoint. Retrieved from https://www.tutorialspoint.com/n-queen-problem-using-backtracking
- Springer. (2017). "N Queen Problem and Its Applications." Springer. Retrieved from https://www.springer.com/gp/book/9783319488573
- Cplusplus.com. (2017). "Implementing N Queen Problem Using Backtracking in C++." Cplusplus. Retrieved from https://www.cplusplus.com/forum/beginner/142299/
- Pustokhina, I., & Yakovlev, A. (2014). "Solving the N Queens Problem Using a Genetic Algorithm." Academia. Retrieved from https://www.academia.edu/26872325/Solving_the_N_Queens_Problem_Using_a_Genetic_Algorithm
- Programiz. (n.d.). "Optimizing N Queen Problem Solutions with Backtracking." Programiz. Retrieved from https://www.programiz.com/dsa/n-queen-problem
- Coursera. (n.d.). "Solving the N Queen Puzzle Using Recursive Backtracking." Coursera. Retrieved from https://www.coursera.org/learn/algorithms-part1
- Sahoo, R. (2021). "N Queen Problem Using Heuristic Search." SAGE Journals. Retrieved from https://www.journals.sagepub.com/doi/full/10.1177/14759217211014963

Plagiarism Report