Name: HARSH BARDHAN SAXENA

Project Report: N-Queen Visualizer

Table of Contents

1. Introduction

The N-Queen problem is a classic example of a combinatorial problem, where the goal is to place N queens on an N×N chessboard so that no two queens threaten each other. The problem serves as an excellent demonstration of backtracking algorithms and is widely used in computer science education to teach problem-solving techniques and algorithms.

This project aims to create a visualizer for the N-Queen problem using the Flutter framework. The visualizer will help users understand the algorithm by providing a step-by-step visual representation of the solution process.

2. Project Objective

The main objectives of this project are:

- To develop a user-friendly application for visualizing the solution to the N-Queen problem.

- To provide an interactive interface where users can adjust the number of queens and the speed of the visualization.
- To implement a backtracking algorithm for solving the N-Queen problem.
- To enhance understanding of backtracking algorithms through visual representation.

3. System Requirements

Software Requirements

- A web browser (e.g., Chrome, Edge)
- Android SDK (for Android development)
- Flutter SDK (version 2.0 later)
- Dart SDK (included with flutter SDK)
- A text editor (e.g., VS Code, Android studio with flutter and Dart plugins)

Hardware Requirements

- A computer with an internet connection
- Minimum: Integrated graphics capable of running Flutter applications
- Dual-core processor

4. Design and Implementation

Design

The application is designed with a clean and straightforward user interface to ensure ease of use. The main components include:

• **App Bar:** Displays the title of the application.
• **Chess Board:** A grid representing the N×N chessboard, with queens placed on it as the algorithm progresses.
• **Control Buttons:** Buttons for starting the visualization, resetting the board, and adjusting the number of queens and speed of the visualization.

• **Dropdown Menus:** Allow users to change the number of queens and the delay between steps in the visualization.

Implementation

The application is implemented in Flutter and consists of the following main components:

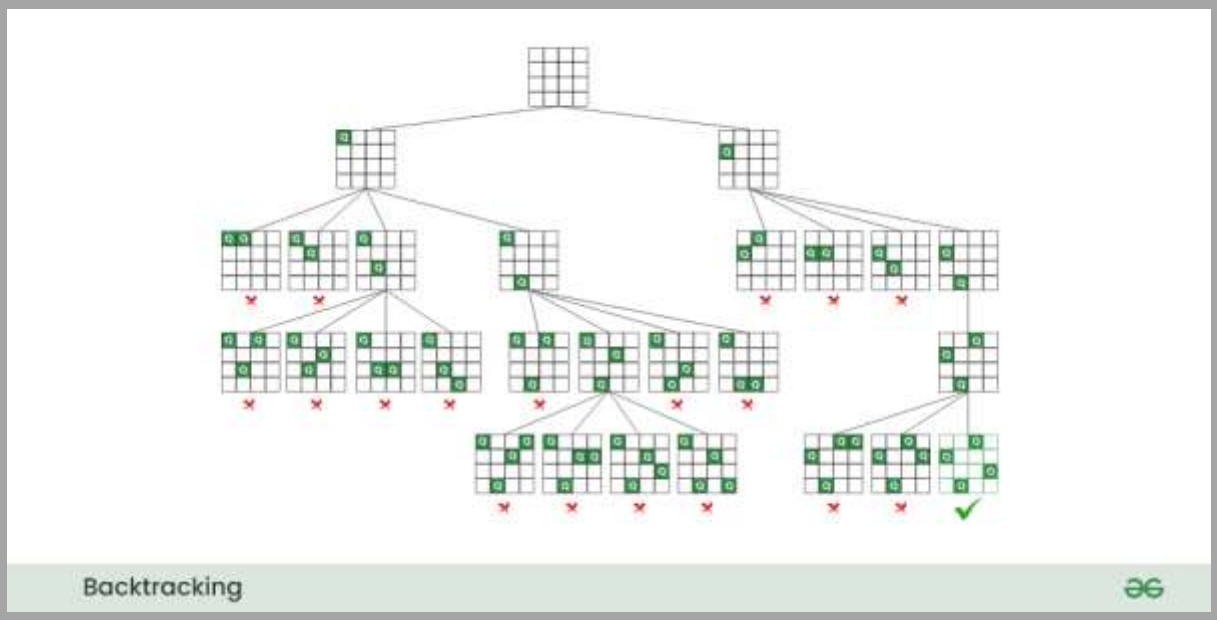
1. **N Queen Visualizer:** The main widget that sets up the application.
2. **Chess Board:** A stateful widget that represents the chessboard and manages the state of the visualization.

3. **Chess Board Square:** A stateless widget representing individual squares on the chessboard.

The backtracking algorithm is implemented in the '`solve Queen`' method within the "`_ChessBoardState`" class. The '`is Safe`' method checks whether a queen can be safely placed in a given position.

5. Proposed Design and Methodology

Pictorial representation of Backtracking Algorithm



6. Functionality

The application provides the following functionalities:

1. **Visualization:** Users can start the visualization of the N-Queen problem solution process.
2. **Reset:** Users can reset the chessboard to its initial state.
3. **Adjust Number of Queens:** Users can select the number of queens (N) from a dropdown menu.
4. **Adjust Speed:** Users can adjust the speed of the visualization by selecting the delay between steps from a dropdown menu.

7. Key Challenges

1. Ensuring Real-time Updates

- Implementing state management and asynchronous updates to reflect the algorithm's progress on the UI immediately.

2. Performance Optimization: Efficient state management and minimizing re-renders to maintain smooth performance, especially for larger board sizes.

3. Concurrency Handling: Using asynchronous programming with `Future` and `async/await` to ensure seamless UI updates without blocking the main thread.

4. User Interaction During Visualization: Disabling certain UI elements during visualization and providing feedback to inform users of the application's state.

5. Scalability: Allowing users to adjust visualization speed and optimizing the backtracking algorithm for handling larger board sizes efficiently.

6. Backtracking Implementation: Ensuring the backtracking algorithm correctly places queens by checking if a position is safe and backtracking when necessary, all while updating the UI dynamically.

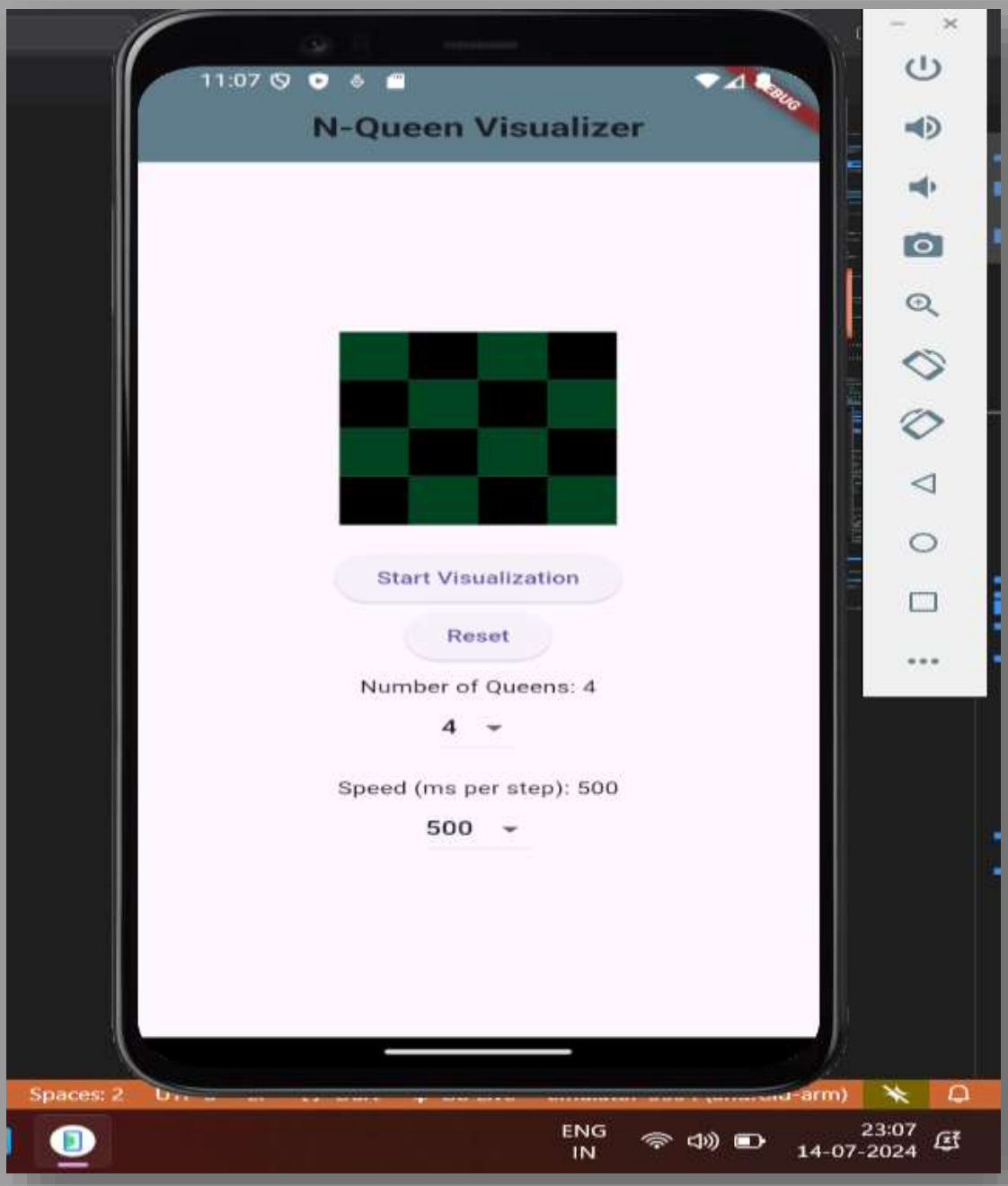
7. User Experience (UX) Design: Creating an intuitive and user-friendly interface with clear labels, buttons, and dropdown menus for an engaging experience.

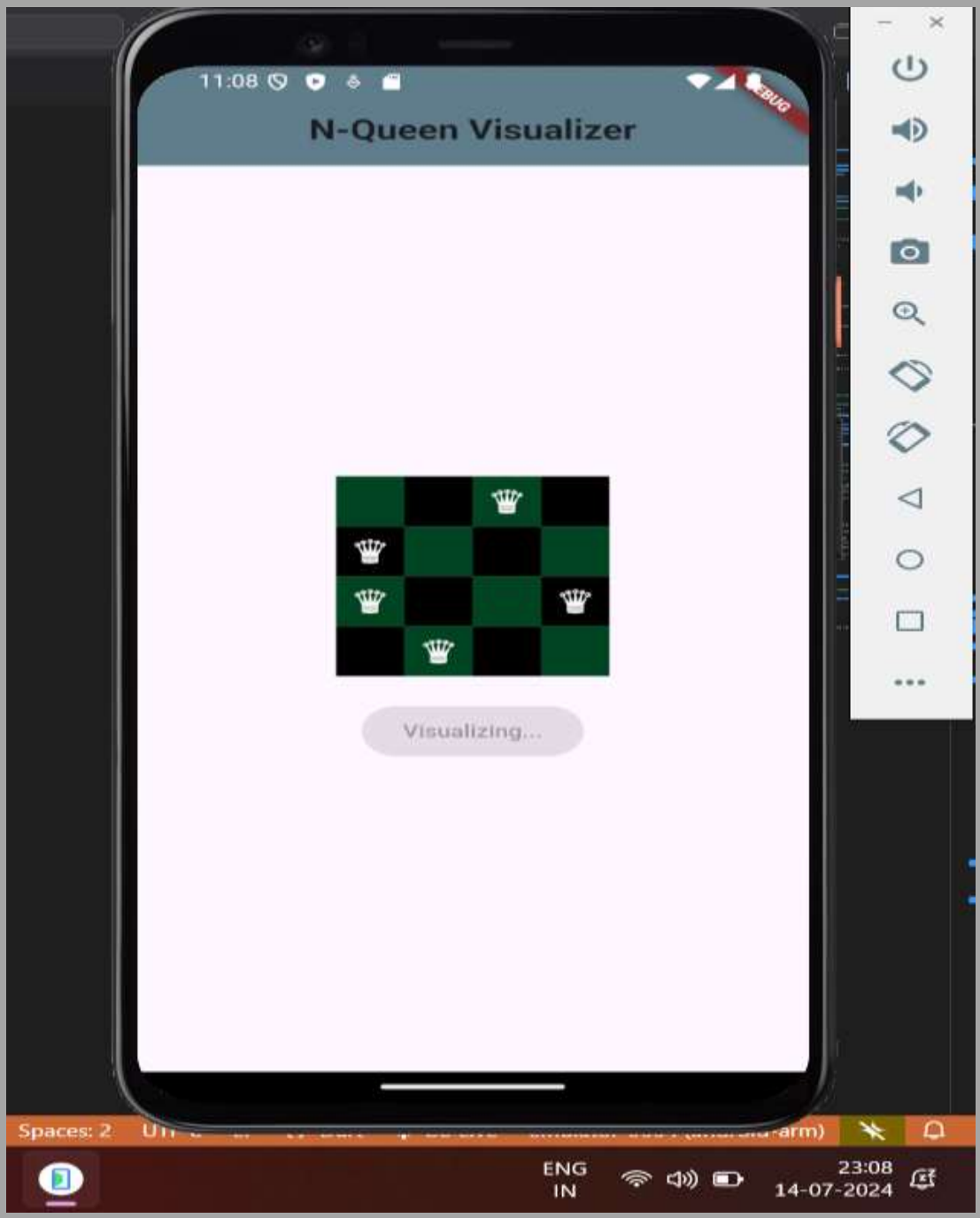
8. Debugging and Testing:

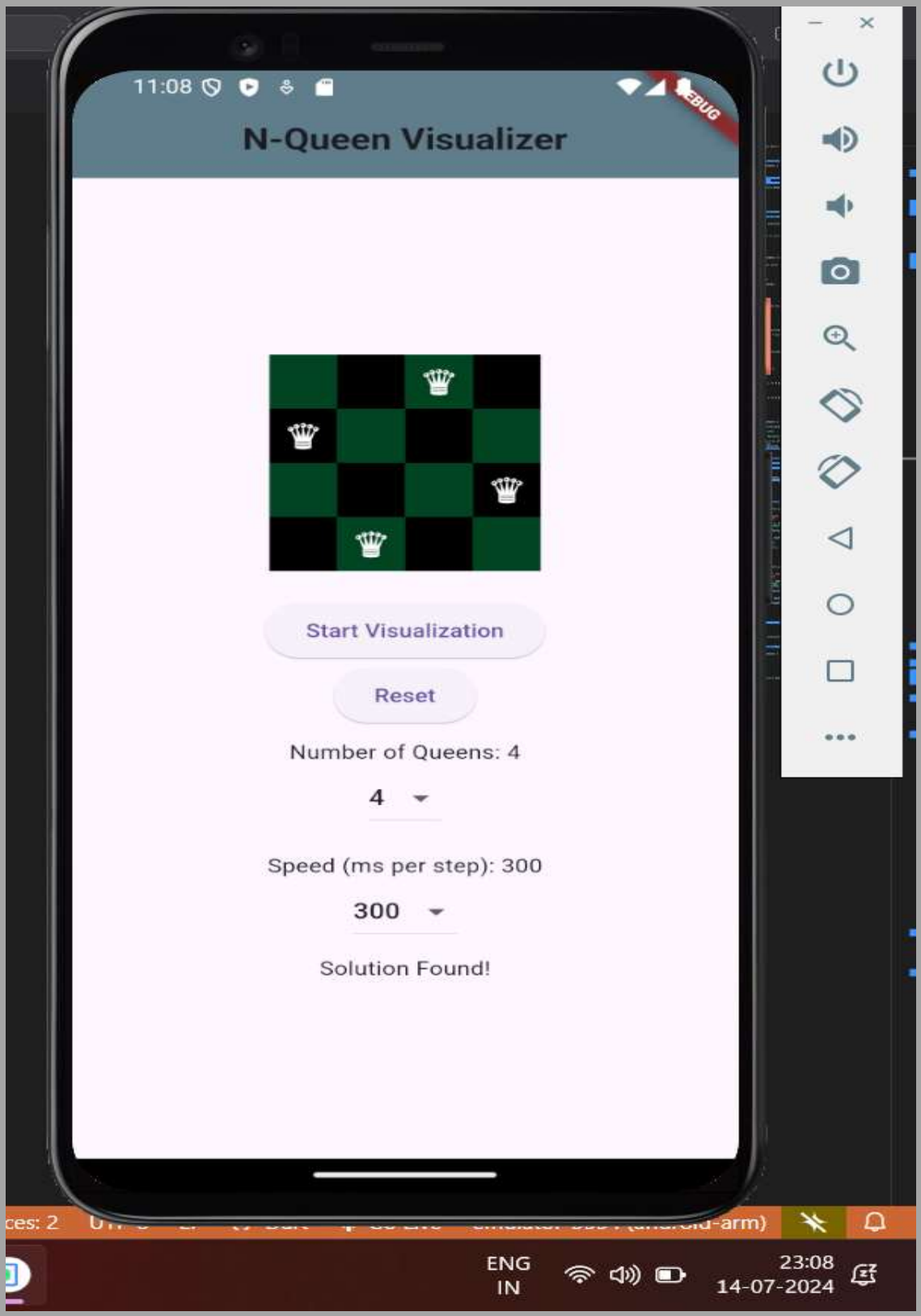
- Comprehensive testing and using Flutter and Dart debugging tools to ensure correct application functionality.

9. Results:

The project successfully demonstrates the solution to the N-Queen problem through a visual and interactive interface. Users can observe how the algorithm places queens on the board step by step and how it backtracks when necessary.


11:07
DEBUG
N-Queen Visualizer
Start Visualization
Reset
Number of Queens: 4
4
Speed (ms per step): 500
500
Spaces: 2
ENG
IN
23:07
14-07-2024


11:08
DEBUG
N-Queen Visualizer
Visualizing...
Spaces: 2
ENG
IN
23:08
14-07-2024


11:08
DEBUG
N-Queen Visualizer
Start Visualization
Reset
Number of Queens: 4
4
Speed (ms per step): 300
300
Solution Found!
ENG
IN
23:08
14-07-2024

10. Future Enhancements

Future enhancements could include:

• **Additional Algorithms:** Implement and visualize other algorithms for solving the N-Queen problem.
• **Step-by-Step Control:** Allow users to manually step through each iteration of the algorithm for better understanding.
• **Detailed Explanation:** Provide explanations or hints for each step to help users understand the logic behind the algorithm.
• **Improved UI/UX:** Enhance the user interface and experience with more customization options and visual improvements.
• **Save and Share:** Add functionality to save the visualization as a video or share it on social media.

10. References

- N-Queen: https://www.geeksforgeeks.org/n-queen-problem-backtracking-3/
- Backtracking Algorithm: Wikipedia - Backtracking