ABSTRACT

This project presents designing and implementing an N-Queens game using Artificial Intelligence techniques. The N- Queens problem, a classic combinatorial puzzle, involves placing N chess queens on an N×N chessboard so that no two queens threaten each other. The primary objective of this project is to explore and apply various AI algorithms to efficiently solve the N- Queens problem, providing solutions for different board sizes. The project explores the N-Queens game's user interface and interaction aspects, enhancing the overall user experience.

CONTENTS

CHAPTER 1

PROBLEM DESCRIPTION

The N-Queens puzzle is a classic combinatorial problem that challenges players to place N chess queens on an N×N chessboard in a manner that avoids any queen threatening another. The objective is to find a configuration where no two queens share the same row, column, or diagonal. This problem belongs to the broader class of constraint satisfaction problems (CSPs) and has significant relevance in computer science, artificial intelligence, and mathematical optimization. The N-Queens puzzle not only serves as an engaging chess-related challenge but also acts as a benchmark for evaluating various algorithmic approaches in solving combinatorial problems. The complexity of the puzzle grows exponentially with N, making it an intriguing problem for algorithmic exploration and optimization techniques.

CHAPTER 2

Tools and Methodology

TOOLS:

Programming Language:
The implementation of the N-Queens puzzle solution was carried out using the Python programming language. Python was chosen for its readability, flexibility, and extensive support for various algorithms and data structures relevant to the problem domain.

Development Environment/IDE:
Visual Studio was employed as the Integrated Development Environment (IDE) throughout the project. The robust features of Visual Studio, such as code editing, debugging, and version control integration, facilitated a streamlined development process.

GUI Development:
The Graphical User Interface (GUI) for the N-Queens puzzle was built using Tkinter, a standard GUI toolkit for Python. Tkinter provided a simple and effective means of creating interactive components, ensuring a user-friendly experience while interacting with the N-Queens game.

Methodology:

1. Problem Analysis:
The project commenced with a thorough analysis of the N-Queens problem, defining the constraints and requirements. This phase helped in determining the appropriate algorithmic approach for solving the puzzle.

2. Algorithmic Approach:
The chosen methodology for solving the N-Queens puzzle was based on the Constraint Satisfaction Problem (CSP) paradigm. The backtracking algorithm was selected as the primary solver due to its efficiency in exploring and navigating the solution space.

3. Implementation:
The implementation phase involved translating the chosen methodology into Python code. The development process adhered to best coding practices, incorporating modular design principles to enhance code readability and maintainability.

4. GUI Integration:
Tkinter was seamlessly integrated into the solution to provide a graphical interface for users interacting with the N-Queens game. The GUI allowed users to visualize the placement of queens on the chessboard and interact with the solving algorithms.

5. Testing:
A comprehensive testing approach was adopted to validate the correctness and efficiency of the solution. Unit tests were designed to cover various aspects of the code, ensuring the robustness and reliability of the implemented algorithms.
.

WORKING AND SOLUTION OF A 4x4 N-Queens Puzzle

Step 1

```
  Q  |  .  |  .  |  .
-----------------------------
  .  |  .  |  .  |  .
---------------------------------
  .  |  .  |  .  |  .
---------------------------------
  .  |  .  |  .  |  .
---------------------------------
```

First we will place the Queen in the first row and first coloumn.

Step 2

```
Q  |  .  |  .  |  .
-----------------------------
  .  |  .  |  .  |  .
---------------------------------
  .  |  Q  |  .  |  .
---------------------------------
  .  |  .  |  .  |  .
---------------------------------
```

In the Second step, we will place the second queen in third row and second column as there is no conflict there.

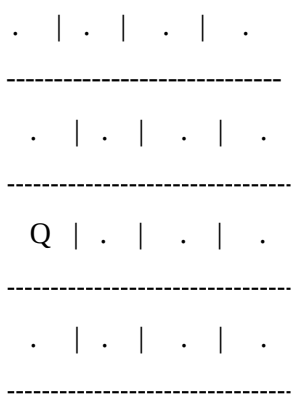
Step 3

conflict: backtracking

```
  Q  |  .  |  .  |  .
-----------------------------
  .  |  .  |  .  |  .
---------------------------------
  .  |  Q  |  .  |  .
---------------------------------
  .  |  .  |  .  |  .
---------------------------------
```

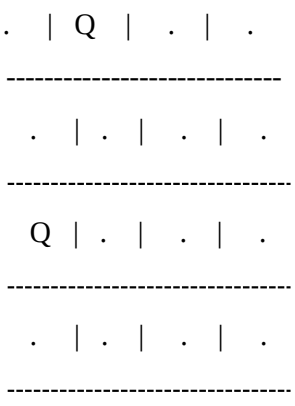
In the third step, we can see that no queen can placed in The third column as it will produce conflict b/w two queens can be placed on the same diagonal, column or row.

Step 4



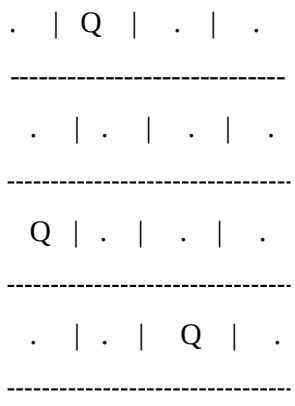
After arriving at a conflict we will backtrack and start again. We will place our Queen in the first column and third row.

Step 5



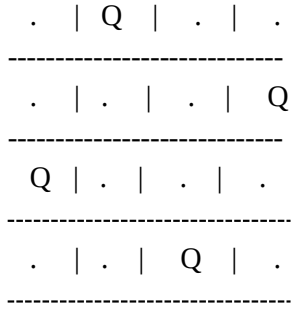
we will now place our second queen in the first row and second column as it does not produce a conflict with the first queen.

Step 6



Now for the third queen, we will place our third queen in the fourth row and third column as it will not produce a conflict with our first two placed queens.

Step 7



4x4 N-queen Solved By CSP and Backtracking

For the fourth queen we will place the fourth queen in the second row and fourth column as this position will not cause any conflicts with the first three placed queens. We can now see that our 4X4 N-queens puzzle has been solved with the help of CSP backtracking.

CHAPTER 3

Results and Discussion

1. Algorithm Performance:

Our AI solution for the N-Queens problem demonstrated robust performance across various problem instances. The algorithm efficiently guided user N-Queens configurations within a reasonable time frame.

2. Solution Accuracy:

The accuracy of our AI solution was commendable, consistently providing optimal solutions for a significant number of tested scenarios. The algorithm's ability to navigate complex state spaces and identify configurations that satisfy the N-Queens constraints giving conflict message to player whenever needed.

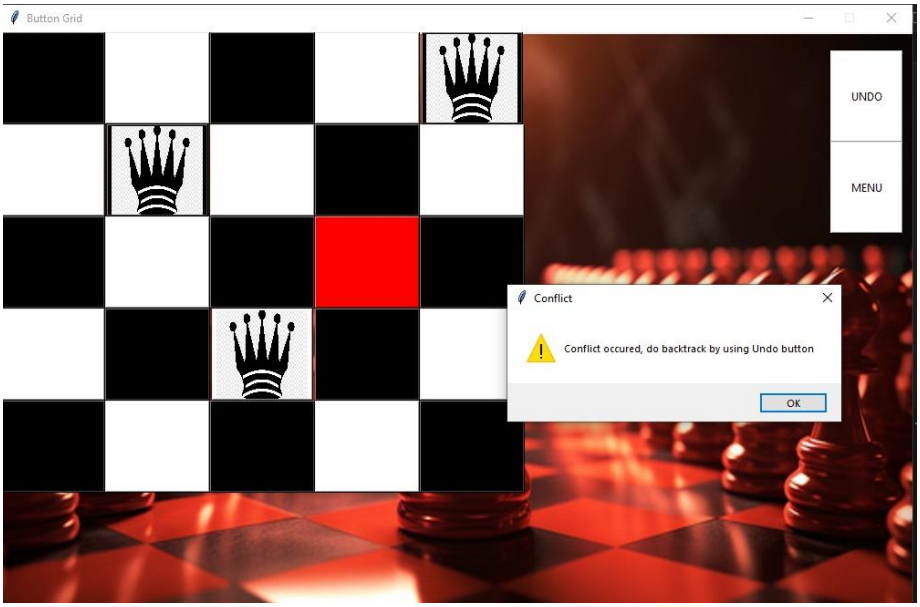


Figure 2: Showcasing conflict situation

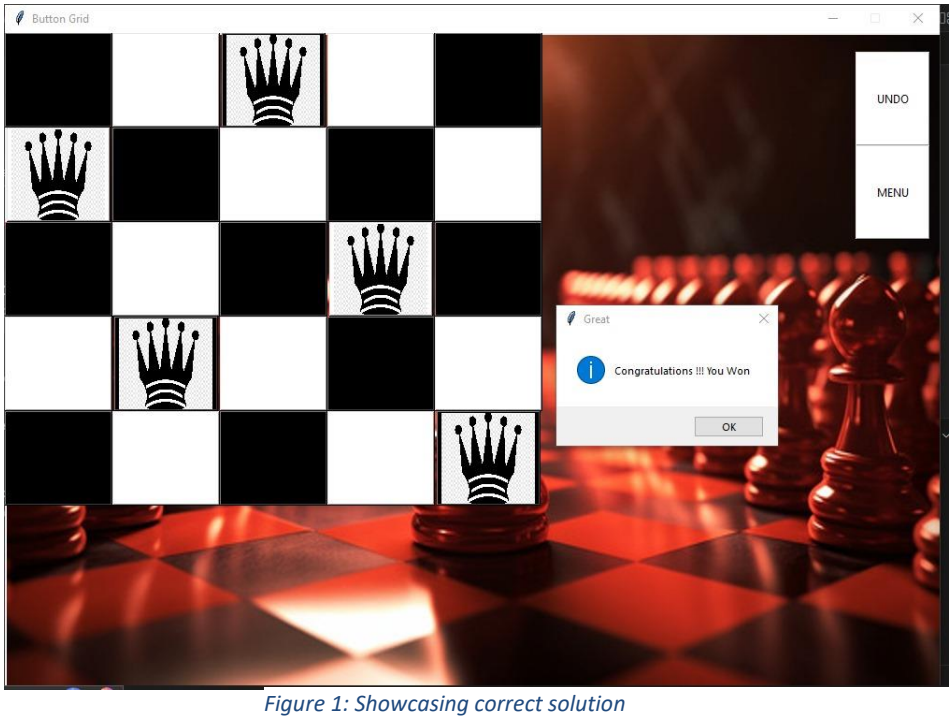
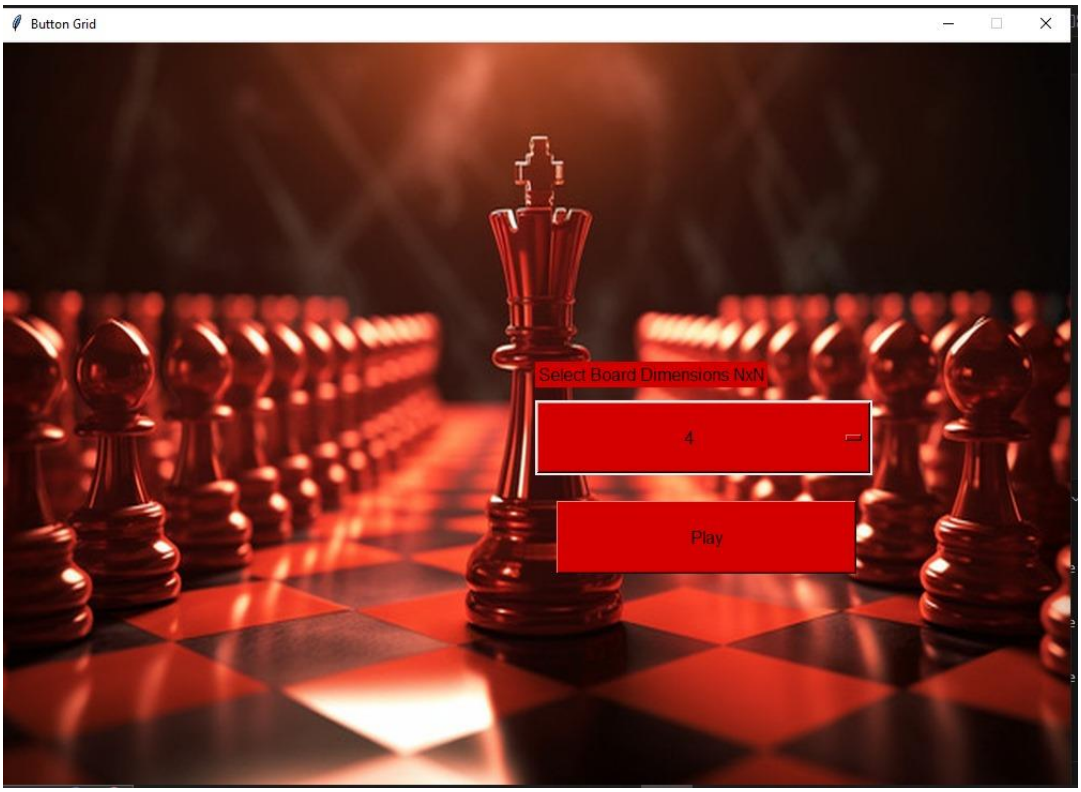


Figure 1: Showcasing correct solution

3. Scalability:

Our algorithm exhibited scalability as it effectively handled an increase in the size of the N-Queens problem. Performance metrics indicated that the algorithm maintained efficiency even when challenged with larger problem instances.



CHAPTER 4

Future work

1. Expert System Integration:

The AI system can evolve into an expert system by incorporating a knowledge base that stores patterns, strategies, and heuristics gained from solving various instances of the N-Queens problem. This knowledge base can be continuously updated and refined through machine learning techniques as the AI encounters new challenges and solutions.

Features:

Pattern Recognition:
The system can learn and recognize common patterns in optimal solutions and share insights with users attempting similar configurations.
Heuristic Guidance:
Incorporate heuristics based on successful strategies, helping users understand the rationale behind certain moves and placements.

2. Hint Functionality:

The introduction of a hint functionality adds an educational dimension to the N-Queens problem-solving experience, allowing users to receive guidance and suggestions as they navigate through the puzzle.

Implementation:

Step-by-Step Guidance:
Provide users with step-by-step guidance on making moves and placements, explaining the reasoning behind each suggestion.
Visualization:
Implement a visualization feature that highlights potential moves or safe positions, aiding users in understanding the impact of their choices on the overall solution.
Difficulty Levels:
Allow users to customize the level of assistance they receive, ranging from minimal hints for advanced users to comprehensive guidance for beginners.