N – Queens game Implementation

The N-Queens Problem is a classic toy problem in computer science and mathematics. It is not a traditional game with players or winning conditions but rather a puzzle or algorithmic challenge. Here's a detailed explanation of the N-Queens problem:

Overview of the N-Queens Problem

The N-Queens problem involves placing N queens on an N×N chessboard such that no two queens threaten each other. In chess, a queen can attack horizontally, vertically, and diagonally. The goal is to find all possible arrangements of queens that satisfy this condition.

Rules of the N-Queens Problem

1. Chessboard: The problem is played on an N×N grid (chessboard), where N is the number of queens.

2. Queens Placement:

- Place N queens on the chessboard.

- No two queens should share the same row, column, or diagonal.

3. Constraints:

- Row Constraint: Only one queen can be placed in each row.

- Column Constraint: Only one queen can be placed in each column.

- Diagonal Constraint: No two queens can be placed on the same diagonal.

Winning Condition

- The problem is solved when all N queens are placed on the chessboard without violating any of the constraints.

- There is no "winning" or "losing" in the traditional sense, as the goal is to find a valid configuration (or all valid configurations) of queens.

Player Capacity

- The N-Queens problem is typically a single-player puzzle.

- It is not a competitive game but rather a problem-solving exercise.

- However, it can be turned into a multiplayer game by challenging players to find solutions faster or with fewer attempts.

Example: 8-Queens Problem

The most common version of the problem is the 8-Queens Problem, where N = 8. The goal is to place 8 queens on an 8×8 chessboard such that no two queens threaten each other.

Sample Solution for 8-Queens

One possible solution is:

```
Q _ _ _ _ _ _ _
_ _ _ _ Q _ _ _
_ _ _ _ _ _ _ Q
_ _ _ _ _ Q _ _
_ _ Q _ _ _ _ _
_ _ _ _ _ _ Q _
_ Q _ _ _ _ _ _
_ _ _ Q _ _ _ _
```

Here, `Q` represents a queen, and `_` represents an empty square.

How to Solve the N-Queens Problem

1. Backtracking Algorithm:
   - Start placing queens row by row.
   - For each row, try placing a queen in each column.
   - If a placement violates the constraints, backtrack and try the next column.
   - Repeat until all queens are placed or all possibilities are exhausted.

2. Heuristics and Optimizations:
   - Use symmetry to reduce the search space.
   - Implement pruning to avoid exploring invalid configurations early.

3. Recursive Approach:
   - Use recursion to explore all possible placements of queens.

Real-World Applications

The N-Queens problem is not just a toy problem; it has real-world applications in areas such as:

1. Scheduling and Resource Allocation:
   - Assigning tasks to workers without conflicts.
   - Scheduling events or meetings without overlapping resources.
2. VLSI (Very Large Scale Integration) Design:
   - Placing components on a chip without interference.
3. Logistics and Operations Research:
   - Optimizing the placement of facilities or warehouses.
4. Constraint Satisfaction Problems (CSPs):
   - The N-Queens problem is a classic example of CSPs, which are used in AI for solving problems with constraints.

Why It's a Good Toy Problem

1. Simplicity: The rules are easy to understand, but the problem is computationally challenging.
2. Scalability: The problem can be scaled by increasing N, making it suitable for testing algorithms.
3. Educational Value: It teaches backtracking, recursion, and constraint satisfaction, which are fundamental concepts in AI and computer science.

Implementation Ideas

- Implement the N-Queens problem using backtracking in Python or another programming language.
- Visualize the solutions using a graphical library (e.g., `matplotlib` or `pygame`).
- Extend the problem to find all possible solutions for a given N.

SOURCE CODE

```
def is_valid(board, row, col, n):
    """
    Checks if placing a queen at (row, col) is valid.
    Ensures no other queen is in the same column, diagonal left, or diagonal right.
    """
    for i in range(row):
        if board[i] == col or \
                board[i] - i == col - row or \
                board[i] + i == col + row:
            return False
    return True


def check_n_queens(board, n):
    """
    Validates the entire board to check if all placed queens satisfy the N-Queens constraints.
    """
    for i in range(n):
        if not is_valid(board, i, board[i], n):
            return False
    return True


def print_board(board, n):
    """
    Prints the current board with 'Q' for queens and '-' for empty spaces.
    """
    print("\nCurrent Board:")
    for i in range(n):
        row = ["Q" if board[i] == j else "-" for j in range(n)]
        print(" ".join(row))


def visualize_board(board, n):
    """
    Visualizes the board in a grid format using 'Q' for queens and '_' for empty spaces.
    """
    print("\nBoard Layout:")
    for i in range(n):
        row = ["Q" if board[i] == j else "_" for j in range(n)]
        print("|" + " | ".join(row) + "|")


def main():
```

```
    """
    Main function to execute the N-Queens game.
    - Takes input for board size N
    - Allows player to place queens one by one
    - Visualizes the board after each move
    - Checks if the final placement is correct
    """
    n = int(input("Enter the value of N: "))
    board = [-1] * n
    visualize_board(board, n)  # Show initial empty board
    print(f"Place {n} queens on an {n}x{n} board.")

    for i in range(n):
        while True:
            try:
                # Asking user for the position of queen
                row, col = map(int, input(f"Enter row and column for Queen {i + 1} (1 to {n}): ").split())
                if 1 <= row <= n and 1 <= col <= n and board[row - 1] == -1:
                    board[row - 1] = col - 1  # Place queen
                    visualize_board(board, n)  # Show updated board
                    break
                else:
                    print("Invalid position! Try again.")
            except ValueError:
                print("Invalid input! Enter two numbers separated by space.")

    print_board(board, n)  # Final board display
    if check_n_queens(board, n):
        print("Congratulations! You placed the queens correctly.")
    else:
        print("Incorrect placement! Try again.")


if __name__ == "__main__":
    main()
```

Code Execution

Initially, the program asks the user to input the value of N in order to create a N x N matrix for placing the Queens.

```
C:\Users\rajin\AppData\Local\Programs\Python\Python313\python.exe "C:\Users\rajin\On
Enter the value of N:
```

After the creation of the N x N matrix, the program asks the position for placing the Queen. The index of the matrix was altered in code to make it easier for everyone in choosing the matrix location.

```
C:\Users\rajin\AppData\Local\Programs\Python\Python313\python.exe "C:\Users\rajin\OneDr
Enter the value of N: 5

Board Layout:
|_ | _ | _ | _ | _|
|_ | _ | _ | _ | _|
|_ | _ | _ | _ | _|
|_ | _ | _ | _ | _|
|_ | _ | _ | _ | _|
Place 5 queens on an 5x5 board.
Enter row and column for Queen 1 (1 to 5):
```

According to the location entered by the user, the program places a ‘Q’ sign at that specific location.

```
Board Layout:
|_ | _ | _ | _ | _|
|_ | _ | _ | _ | _|
|_ | _ | _ | _ | _|
|_ | _ | _ | _ | _|
|_ | _ | _ | _ | _|
Place 5 queens on an 5x5 board.
Enter row and column for Queen 1 (1 to 5): 1 1

Board Layout:
|Q | _ | _ | _ | _|
|_ | _ | _ | _ | _|
|_ | _ | _ | _ | _|
|_ | _ | _ | _ | _|
|_ | _ | _ | _ | _|
Enter row and column for Queen 2 (1 to 5):
```

After placement of all N number of Queens, the program encloses the result if it is a ‘Win’ or ‘Lose’.

If all the Queens are placed in a correct manner according to the game rules, then the program prints the message for Winning.

```
|Q | _ | _ | _ | _|
|_ | _ | Q | _ | _|
|_ | _ | _ | _ | Q|
|_ | Q | _ | _ | _|
|_ | _ | _ | Q | _|

Current Board:
Q - - - -
- - Q - -
- - - - Q
- Q - - -
- - - Q -
Congratulations! You placed the queens correctly.

Process finished with exit code 0
```

If the Queens are placed incorrectly, then the program prints the respective message for ‘Lose’.

```
|Q | _ | _ | _ | _|
|_ | Q | _ | _ | _|
|_ | _ | _ | _ | Q|
|_ | _ | Q | _ | _|
|_ | _ | _ | Q | _|

Current Board:
Q - - - -
- Q - - -
- - - - Q
- - Q - -
- - - Q -
Incorrect placement! Try again.

Process finished with exit code 0
```