```python
import random
import numpy as np
import matplotlib.pyplot as plt


def GenerateRandomBoard(n):
    """
    :param n: Number queens for board
    :return: List of the random boards.
    """
    return [random.randint(0, n - 1) for _ in range(n)]


def ShowBoard(board, title1, title2):
    """

    :param board: Board with Queens
    :param title1: Initial State Plot
    :param title2: Final State Plot
    :return:
    """
    n = len(board)
    fig, (ax1, ax2) = plt.subplots(1, 2)

    # Plot initial state
    ax1.matshow(np.zeros((n, n)), cmap='gray')
    for i in range(n):
        ax1.plot(board[i], i, 'ro')
    ax1.set_title(title1)

    # Plot final state
    ax2.matshow(np.zeros((n, n)), cmap='gray')
    for i in range(n):
        ax2.plot(board[i], i, 'ro')
    ax2.set_title(title2)

    plt.show()


def GetQueenAttackCounts(board):
    """
    :param board: Any board state
    :return: Number of attacking queens
    """
    n = len(board)
    attacks = 0
    for i in range(n):
        for j in range(i + 1, n):
            if board[i] == board[j] or abs(board[i] - board[j]) == abs(j - i):
                attacks += 1
    return attacks


def HillClimbingAlgorithm(n, max_iterations=100):
    """
    :param n: Number of queens
    :param max_iterations: Number of times before the Algorithm ends
    :return:
    """
    solutions_found = 0

    for i in range(1, 101):  # Repeat the program 100 times
        board = GenerateRandomBoard(n)
        initial_attacks = GetQueenAttackCounts(board)
        current_attacks = initial_attacks
        iterations = 0
```

```
        while current_attacks > 0 and iterations < max_iterations:
            neighbor = list(board)
            row = random.randint(0, n - 1)
            col = random.randint(0, n - 1)
            neighbor[row] = col
            neighbor_attacks = GetQueenAttackCounts(neighbor)

            if neighbor_attacks < current_attacks:
                board = neighbor
                current_attacks = neighbor_attacks

            iterations += 1

        if current_attacks == 0:
            solutions_found += 1

        if i <= 10:  # Plot initial and final states for the first 10 runs
            print(f"Run {i}: Initial Attacks = {initial_attacks}, Final Attacks =
{current_attacks}")
            ShowBoard(board, f"Run {i} Initial State", f"Run {i} Final State")

    return solutions_found


def main():
    """
    :return: Driver method.
    """
    N = 8
    solutions = HillClimbingAlgorithm(N)
    print(f"Total solutions found for N = {N}: {solutions} out of 100 runs")


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