```
import random
import math

# Objective (Energy) Function: Number of conflicts in the board
def calculate_conflicts(board):
    conflicts = 0
    for i in range(len(board)):
        for j in range(i + 1, len(board)):
            if board[i] == board[j]:  # Same row
                conflicts += 1
            if abs(board[i] - board[j]) == abs(i - j):  # Same diagonal
                conflicts += 1
    return conflicts

# Function to make a random move (neighbor) in the state space
def make_random_move(board):
    new_board = board[:]
    col = random.randint(0, len(board) - 1)
    new_row = random.randint(0, len(board) - 1)
    new_board[col] = new_row
    return new_board

# Simulated Annealing Algorithm
def simulated_annealing(N, initial_board):
    current_board = initial_board[:]
    current_conflicts = calculate_conflicts(current_board)

    # Define cooling parameters
    T = 1000  # Initial temperature
    T_min = 0.0001  # Minimum temperature
    alpha = 0.99  # Cooling rate

    iteration = 0  # Track number of iterations

    # Main loop of the algorithm
    while T > T_min and current_conflicts > 0:
        # Remove or comment out the iteration print statement to suppress it
        # Every 100 iterations, print the current state (commented out as per the request)
        # iteration += 1
        # if iteration % 100 == 0:
        #     print(f"Iteration {iteration}: Current Conflicts = {current_conflicts}")

        # Generate a new state by making a random move
        new_board = make_random_move(current_board)
        new_conflicts = calculate_conflicts(new_board)

        # Calculate the energy difference
        delta_E = new_conflicts - current_conflicts

        # Decide to move to the new state
        if delta_E < 0 or random.uniform(0, 1) < math.exp(-delta_E / T):
            current_board, current_conflicts = new_board, new_conflicts

        # Cool down
        T *= alpha

        # If the board is conflict-free, break early
        if current_conflicts == 0:
            print(f"Solution found!")
            break

    return current_board, current_conflicts

# Get input from the user for the size of the board
N = int(input("Enter the size of the board (N): "))

# Get the initial state of the board from the user
initial_state = input(f"Enter the initial state as a list of {N} integers")

# Convert the input string to a list of integers
initial_board = [int(x) for x in initial_state.strip('[]').split(',')]

# Check if the length of the input board is correct
if len(initial_board) != N:
    print(f"Error: The initial state must have exactly {N} integers.")
else:
    # Run the Simulated Annealing algorithm
    solution_board, solution_conflicts = simulated_annealing(N, initial_board)

    # Output the solution
    if solution_conflicts == 0:
```

```
        print("Board configuration:", solution_board)
    else:
        print("Solution not found. Final conflicts:", solution_conflicts)
```

```
Enter the size of the board (N): 8
Enter the initial state as a list of 8 integers1,0,3,6,5,7,2,4
Solution found!
Board configuration: [2, 4, 7, 3, 0, 6, 1, 5]
```