Start coding or generate with AI.

Simulated Annealing (N queens)

```
import random
import math

def calculate_conflicts(board):
    """
    Calculate the number of conflicts in the current board configuration.
    A conflict occurs when two queens attack each other.
    """
    conflicts = 0
    n = len(board)
    for i in range(n):
        for j in range(i + 1, n):
            if board[i] == board[j]:
                conflicts += 1
            elif abs(board[i] - board[j]) == abs(i - j):
                conflicts += 1
    return conflicts

def generate_neighbor(board):
    """
    Generate a neighbor by moving one queen to a new row in its column,
    ensuring no column or diagonal conflicts.
    """
    n = len(board)
    new_board = board[:]

    col = random.randint(0, n - 1)

    current_row = new_board[col]
    possible_rows = set(range(n)) - {current_row}

    valid_rows = set()
    for row in possible_rows:
        valid = True
        for c in range(n):
            if c != col and abs(row - new_board[c]) == abs(col - c):
                valid = False
                break
        if valid:
            valid_rows.add(row)

    if valid_rows:

        new_board[col] = random.choice(list(valid_rows))
    return new_board

def simulated_annealing(n, initial_state, max_iterations=10000, initial_temp=1000, cooling_rate=0.99):
    """
    Simulated Annealing to solve the N-Queens problem.
    """
    current_state = initial_state
    current_conflicts = calculate_conflicts(current_state)
    temperature = initial_temp

    for iteration in range(max_iterations):
        if current_conflicts == 0:
            return current_state

        neighbor = generate_neighbor(current_state)
        neighbor_conflicts = calculate_conflicts(neighbor)

        delta = current_conflicts - neighbor_conflicts

        if delta > 0 or random.random() < math.exp(delta / temperature):
            current_state = neighbor
            current_conflicts = neighbor_conflicts

        temperature *= cooling_rate

    return None

def print_solution(board):
    """
    Prints the solution board in a human-readable format.
    """
    n = len(board)
    for row in range(n):
        board_row = ['Q' if col == board[row] else '.' for col in range(n)]
        print(' '.join(board_row))

print('Vignesh B(1BM22CS326):')
```

```
n = int(input("Enter the number of queens: "))
initial_state_input = input(f"Enter the initial state (a list of {n} integers representing the row positions of queens in each column): ")

initial_state = list(map(int, initial_state_input.strip('[]').split(',')))


if len(initial_state) != n or any(queen < 0 or queen >= n for queen in initial_state):
    print("Invalid initial state! Please make sure it's a list of integers between 0 and n-1.")
else:
    solution = simulated_annealing(n, initial_state)

    if solution:
        print("Solution found:")
        print_solution(solution)
    else:
        print("No solution found.")
```

```
Vignesh B(1BM22CS326):
Enter the number of queens: 4
Enter the initial state (a list of 4 integers representing the row positions of queens in each column): 3, 1, 2, 0
Solution found:
. . Q .
Q . . .
. . . Q
. Q . .
```