6. N queens using Simulated Annealing

Code :

```
import random
import math

def create_board_from_input(n, positions):
    """Create a board based on user input positions for each column."""
    return positions

def calculate_conflicts(board):
    """Calculate the number of conflicts on the board."""
    n = len(board)
    conflicts = 0
    for i in range(n):
        for j in range(i + 1, n):
            if board[i] == board[j] or abs(board[i] - board[j]) == j - i:
                conflicts += 1
    return conflicts

def get_neighbors(board):
    """Generate all neighboring boards by changing the position of one
queen."""
    n = len(board)
    neighbors = []
    for i in range(n):
        for j in range(n):
            if board[i] != j:
                neighbor = list(board)
                neighbor[i] = j
                neighbors.append(neighbor)
    return neighbors

def simulated_annealing(n, initial_temperature, cooling_rate, initial_state):
    """Perform simulated annealing to solve the n-queens problem."""
    current_board = initial_state
    current_conflicts = calculate_conflicts(current_board)
    best_board = list(current_board)
    best_conflicts = current_conflicts
    temperature = initial_temperature
    iterations = 0  # Counter for iterations

    while temperature > 1:
        iterations += 1  # Increment the iteration count
        neighbors = get_neighbors(current_board)
```

```
            neighbor = random.choice(neighbors)
            neighbor_conflicts = calculate_conflicts(neighbor)

            delta = neighbor_conflicts - current_conflicts

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

            if current_conflicts < best_conflicts:
                best_board = list(current_board)
                best_conflicts = current_conflicts

            temperature *= cooling_rate

        return best_board, best_conflicts, iterations

    # Main code to accept user input
    if __name__ == "__main__":
        n = int(input("Enter the size of the board (number of queens): "))
        initial_temperature = float(input("Enter the initial temperature (e.g.,
100): "))
        cooling_rate = float(input("Enter the cooling rate (e.g., 0.99): "))

        print(f"Enter the initial positions of queens for each column (values
between 0 and {n-1}, one value per column):")
        initial_state = []
        for i in range(n):
            pos = int(input(f"Position for column {i + 1} (0 to {n-1}): "))
            if 0 <= pos < n:
                initial_state.append(pos)
            else:
                print(f"Invalid input for column {i + 1}, please enter a number
between 0 and {n-1}.")
                break
        else:
            solution, conflicts, iterations = simulated_annealing(n,
initial_temperature, cooling_rate, initial_state)

            print("\nSolution:")
            for i in range(n):
                line = ""
                for j in range(n):
                    if j == solution[i]:
                        line += "Q "
                    else:
```

```
                line += ". "
        print(line)

    print("\nConflicts:", conflicts)
    print(f"Iterations: {iterations}")
```

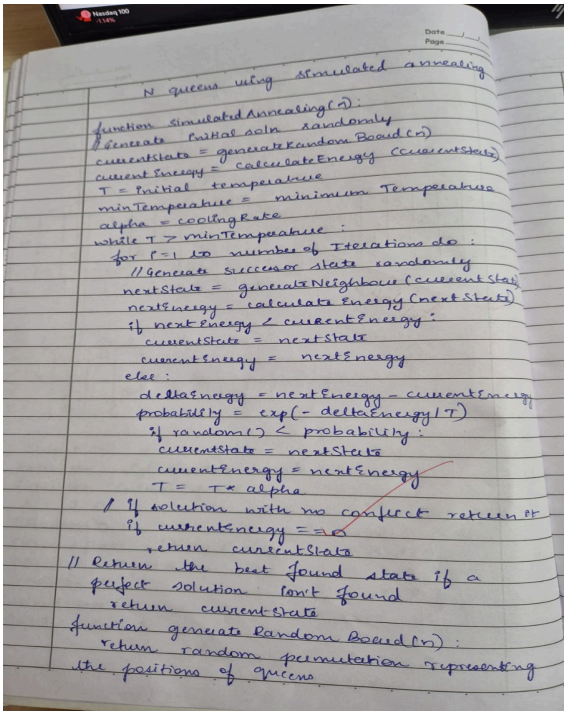
Output:

```
Enter the size of the board (number of queens): 4
Enter the initial temperature (e.g., 100): 50
Enter the cooling rate (e.g., 0.99): 0.22
Enter the initial positions of queens for each column (values between 0 and 3, one value per column):
Position for column 1 (0 to 3): 2
Position for column 2 (0 to 3): 1
Position for column 3 (0 to 3): 3
Position for column 4 (0 to 3): 0

Solution:
. . Q .
. Q . .
. . . Q
Q . . .

Conflicts: 1
Iterations: 3
```

Observation book screenshots:



function generateNeighbour(state):
return modified state with one queen moved to a different row

function calculateEnergy(state):
Conflicts (attacks) between queens
Return count of pairs of queens attacking each other

function exp(value):
return Math.exp(value)

function random():
return Math.random().

Output

Enter the size of the board : 4
Enter the initial temperature : 50
Enter the cooling rate : 0.20
Enter the initial positions of queens for each column
Position for column 1 : 2
Position for column 2 : 1
Position for column 3 : 3
Position for column 4 : 0

15/11/24

Solution

. . Q .
. Q . .
. . . Q
Q . . .

Conflicts : 1
Iterations : 3