WEEK-06 SIMULATED ANNEALING

```
import random
import math

def print_board(board, n):
    """Prints the current state of the board."""
    for row in range(n):
        line = ""
        for col in range(n):
            if board[col] == row:
                line += " Q "  # Queen is
represented by "Q"
            else:
                line += " . "  # Empty space
represented by "."
        print(line)
    print()

def calculate_conflicts(board, n):
    """Calculates the number of conflicts (attacks)
between queens."""
    conflicts = 0
    for i in range(n):
        for j in range(i + 1, n):
            # Check if queens are in the same row
or diagonal
            if board[i] == board[j] or abs(board[i]
- board[j]) == abs(i - j):
                conflicts += 1
    return conflicts

def simulated_annealing(n, initial_temp=1000,
cooling_rate=0.995, max_iterations=10000):
    """Simulated Annealing algorithm to solve N-
Queens."""
    # Initial random board configuration (one queen
in each column)
```

```
    board = [random.randint(0, n - 1) for _ in
range(n)]
    current_conflicts = calculate_conflicts(board,
n)
    temperature = initial_temp
    iteration = 0

    print("Initial board:")
    print_board(board, n)
    print(f"Initial conflicts:
{current_conflicts}\n")

    while current_conflicts > 0 and iteration <
max_iterations:
        # Generate a neighboring state by moving a
queen to another row in its column
        col = random.randint(0, n - 1)
        original_row = board[col]
        new_row = random.randint(0, n - 1)
        while new_row == original_row:
            new_row = random.randint(0, n - 1)  #
Ensure we are moving the queen to a new row
        board[col] = new_row

        # Calculate the number of conflicts in the
new configuration
        new_conflicts = calculate_conflicts(board,
n)

        # If the new state has fewer conflicts,
accept it.
        # If the new state has more conflicts,
accept it with a certain probability.
        if new_conflicts < current_conflicts or
random.random() < math.exp((current_conflicts -
new_conflicts) / temperature):
            current_conflicts = new_conflicts
        else:
```

```
            # If no improvement, revert the move
            board[col] = original_row

        # Reduce the temperature according to the
cooling schedule
        temperature *= cooling_rate

        iteration += 1

        # Display progress
        if iteration % 1000 == 0:
            print(f"Iteration {iteration}:
Conflicts = {current_conflicts}, Temperature =
{temperature}")
            print_board(board, n)

    return board, current_conflicts

def main():
    # Input dynamic parameters
    print("Welcome to the N-Queens Problem Solver
using Simulated Annealing!")
    n = int(input("Enter the size of the board (N):
"))
    initial_temp = float(input("Enter the initial
temperature (e.g., 1000): "))
    cooling_rate = float(input("Enter the cooling
rate (e.g., 0.995): "))
    max_iterations = int(input("Enter the maximum
number of iterations (e.g., 10000): "))

    solution, conflicts = simulated_annealing(n,
initial_temp, cooling_rate, max_iterations)

    print("Final solution:")
    print_board(solution, n)
    if conflicts == 0:
```

```
        print("A solution was found with no
conflicts!")
    else:
        print(f"No solution was found after
{max_iterations} iterations. Final number of
conflicts: {conflicts}")

if __name__ == "__main__":
    main()
print("Suvina A Shetty")
print("1BM22CS299")
```

OUTPUT

```
Welcome to the N-Queens Problem Solver using Simulated Annealing!
Enter the size of the board (N): 4
Enter the initial temperature (e.g., 1000): 1000
Enter the cooling rate (e.g., 0.995): 0.99
Enter the maximum number of iterations (e.g., 10000): 200
Initial board:
 . . . Q
 . . . .
 Q Q Q .
 . . . .

Initial conflicts: 4

Final solution:
 . . Q .
 Q . . .
 . . . Q
 . Q . .

A solution was found with no conflicts!
Suvina A Shetty
1BM22CS299
```