```
import random

# Function to print the 8-puzzle board
def print_board(state):
    for i in range(3):
        print(state[i*3:(i+1)*3])
    print()

# Function to calculate the heuristic (Manhattan distance) for a given state
def manhattan_distance(state, goal):
    distance = 0
    for i in range(1, 9):
        x1, y1 = divmod(state.index(i), 3)
        x2, y2 = divmod(goal.index(i), 3)
        distance += abs(x1 - x2) + abs(y1 - y2)
    return distance

# Generate possible moves for the blank space in the puzzle
def generate_moves(state):
    moves = []
    blank_index = state.index(0)
    row, col = divmod(blank_index, 3)

    possible_moves = {'UP': (row - 1, col), 'DOWN': (row + 1, col),
                      'LEFT': (row, col - 1), 'RIGHT': (row, col + 1)}

    for move, (r, c) in possible_moves.items():
        if 0 <= r < 3 and 0 <= c < 3:
            new_index = r * 3 + c
            new_state = state[:]
            new_state[blank_index], new_state[new_index] = new_state[new_index], new_state[blank_index]
            moves.append((new_state, move))

    return moves

# Hill-Climbing search algorithm implementation
def hill_climbing(start, goal):
    current_state = start
    current_cost = manhattan_distance(current_state, goal)

    while True:
        neighbors = generate_moves(current_state)
        best_neighbor = None
        best_cost = current_cost

        # Evaluate all neighbors to find the best move
        for neighbor_state, move in neighbors:
            neighbor_cost = manhattan_distance(neighbor_state, goal)
            if neighbor_cost < best_cost:  # Minimize the heuristic cost
                best_cost = neighbor_cost
                best_neighbor = neighbor_state

        # If no better neighbor is found, we reached a peak (local optimum)
        if best_neighbor is None:
            break

        current_state = best_neighbor
        current_cost = best_cost
        print_board(current_state)
        print(f"Current Heuristic Cost: {current_cost}\n")
    return current_state

# Define the initial state and goal state of the 8-puzzle problem
start_state = [1, 2, 3, 4, 0, 5, 6, 7, 8]  # 0 represents the blank space
goal_state = [1, 2, 3, 4, 5, 6, 7, 8, 0]

# Run the Hill-Climbing algorithm
print("Initial State:")
print_board(start_state)
final_state = hill_climbing(start_state, goal_state)

# Display the result
print("Final State Reached:")
print_board(final_state)

if final_state == goal_state:
    print("Goal state achieved!")
else:
    print("Reached a local optimum (couldn't find the global optimum).")
```