```python
from random import randint

def configureBoard(board, state, size):
    for i in range(size):
        row = int(input(f"Enter row position for queen in column {i + 1} (0 to {size - 1}): "))
        while row < 0 or row >= size or board[row][i] == 1:
            print("Invalid position or a queen already exists there.")
            row = int(input(f"Enter row position for queen in column {i + 1} (0 to {size - 1}): "))
        state[i] = row
        board[row][i] = 1

def printBoard(board, size):
    for i in range(size):
        print(*board[i])
    print("\n")

def compareStates(state1, state2, size):
    for i in range(size):
        if state1[i] != state2[i]:
            return False
    return True

def fill(board, value, size):
    for i in range(size):
        for j in range(size):
            board[i][j] = value

def calculateObjective(board, state, size):
    attacking = 0
    for i in range(size):
        row = state[i]
        col = i - 1
        while col >= 0 and board[row][col] != 1:
            col -= 1
        if col >= 0 and board[row][col] == 1:
            attacking += 1
        col = i + 1
        while col < size and board[row][col] != 1:
            col += 1
        if col < size and board[row][col] == 1:
            attacking += 1
        row, col = state[i] - 1, i - 1
        while col >= 0 and row >= 0 and board[row][col] != 1:
            col -= 1
            row -= 1
        if col >= 0 and row >= 0 and board[row][col] == 1:
            attacking += 1
        row, col = state[i] + 1, i + 1
        while col < size and row < size and board[row][col] != 1:
            col += 1
            row += 1
        if col < size and row < size and board[row][col] == 1:
            attacking += 1
        row, col = state[i] + 1, i - 1
        while col >= 0 and row < size and board[row][col] != 1:
            col -= 1
            row += 1
        if col >= 0 and row < size and board[row][col] == 1:
            attacking += 1
        row, col = state[i] - 1, i + 1
        while col < size and row >= 0 and board[row][col] != 1:
            col += 1
            row -= 1
        if col < size and row >= 0 and board[row][col] == 1:
            attacking += 1
    return int(attacking / 2)

def generateBoard(board, state, size):
    fill(board, 0, size)
    for i in range(size):
        board[state[i]][i] = 1

def copyState(state1, state2, size):
    for i in range(size):
        state1[i] = state2[i]
```

```python
def getNeighbour(board, state, size):
    opBoard = [[0 for _ in range(size)] for _ in range(size)]
    opState = [0 for _ in range(size)]
    copyState(opState, state, size)
    generateBoard(opBoard, opState, size)
    opObjective = calculateObjective(opBoard, opState, size)
    NeighbourBoard = [[0 for _ in range(size)] for _ in range(size)]
    NeighbourState = [0 for _ in range(size)]
    copyState(NeighbourState, state, size)
    generateBoard(NeighbourBoard, NeighbourState, size)
    for i in range(size):
        for j in range(size):
            if j != state[i]:
                NeighbourState[i] = j
                NeighbourBoard[NeighbourState[i]][i] = 1
                NeighbourBoard[state[i]][i] = 0
                temp = calculateObjective(NeighbourBoard, NeighbourState, size)
                if temp <= opObjective:
                    opObjective = temp
                    copyState(opState, NeighbourState, size)
                    generateBoard(opBoard, opState, size)
                NeighbourBoard[NeighbourState[i]][i] = 0
                NeighbourState[i] = state[i]
                NeighbourBoard[state[i]][i] = 1
    copyState(state, opState, size)
    fill(board, 0, size)
    generateBoard(board, state, size)

def hillClimbing(board, state, size):
    neighbourBoard = [[0 for _ in range(size)] for _ in range(size)]
    neighbourState = [0 for _ in range(size)]
    copyState(neighbourState, state, size)
    generateBoard(neighbourBoard, neighbourState, size)
    iteration = 1
    while True:
        print(f"Iteration {iteration}:")
        printBoard(board, size)
        copyState(state, neighbourState, size)
        generateBoard(board, state, size)
        getNeighbour(neighbourBoard, neighbourState, size)
        if compareStates(state, neighbourState, size):
            print("Final State Reached:")
            printBoard(board, size)
            break
        elif calculateObjective(board, state, size) == calculateObjective(neighbourBoard, neighbourState, size):
            neighbourState[randint(0, 100000) % size] = randint(0, 100000) % size
            generateBoard(neighbourBoard, neighbourState, size)
        iteration += 1

size = int(input("Enter board size (1 to 8): "))
while size < 1 or size > 8:
    print("Invalid size. Please enter a number between 1 and 8.")
    size = int(input("Enter board size (1 to 8): "))

state = [0] * size
board = [[0 for _ in range(size)] for _ in range(size)]
configureBoard(board, state, size)
hillClimbing(board, state, size)
```

```
Enter board size (1 to 8): 4
Enter row position for queen in column 1 (0 to 3): 0
Enter row position for queen in column 2 (0 to 3): 1
Enter row position for queen in column 3 (0 to 3): 2
Enter row position for queen in column 4 (0 to 3): 3
Iteration 1:
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1


Iteration 2:
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
```

```
Iteration 3:
0 0 0 0
0 1 0 0
0 0 1 1
1 0 0 0


Iteration 4:
0 0 1 0
0 1 0 0
0 0 0 1
1 0 0 0


Iteration 5:
0 0 1 0
0 0 0 0
0 0 0 1
1 1 0 0


Final State Reached:
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0
```

Start coding or generate with AI.