```
import heapq

class Node:
    def __init__(self, state, parent=None, g=0, h=0):
        self.state = state
        self.parent = parent
        self.g = g
        self.h = h
        self.f = g + h

    def __lt__(self, other):
        return self.f < other.f

def findBlank(state):
    for i in range(3):
        for j in range(3):
            if state[i][j] == 0:
                return i, j

def getNeighbors(state):
    blankRow, blankCol = findBlank(state)
    neighbors = []
    possibleMoves = [(0, 1), (0, -1), (1, 0), (-1, 0)]
    for dr, dc in possibleMoves:
        newRow, newCol = blankRow + dr, blankCol + dc
        if 0 <= newRow < 3 and 0 <= newCol < 3:
            newState = [row[:] for row in state]
            newState[blankRow][blankCol], newState[newRow][newCol] = newState[n
            neighbors.append(newState)
    return neighbors

def misplacedTiles(state, goal):
    misplaced = 0
    for i in range(3):
        for j in range(3):
            if state[i][j] != goal[i][j] and state[i][j] != 0:
                misplaced += 1
    return misplaced

def manhattanDistance(state, goal):
    distance = 0
    for i in range(3):
        for j in range(3):
            if state[i][j] != 0:
                goalRow, goalCol = -1, -1
                for x in range(3):
                    for y in range(3):
```

```
                        if state[i][j] == goal[x][y]:
                            goalRow, goalCol = x, y
                            break
                distance += abs(i - goalRow) + abs(j - goalCol)
    return distance

def solve8puzzle(initialState, goalState, heuristic):
    openList = []
    if heuristic == "manhattan":
        h = manhattanDistance(initialState, goalState)
    else:
        h = misplacedTiles(initialState, goalState)

    heapq.heappush(openList, Node(initialState, None, 0, h))
    closed_set = set()

    while openList:
        currNode = heapq.heappop(openList)

        if tuple(map(tuple, currNode.state)) == tuple(map(tuple, goalState)):
            path = []
            while currNode:
                path.append(currNode.state)
                currNode = currNode.parent
            return path[::-1]

        closed_set.add(tuple(map(tuple, currNode.state)))

        for neighbor_state in getNeighbors(currNode.state):
            if tuple(map(tuple, neighbor_state)) not in closed_set:
                g = currNode.g + 1
                if heuristic == "manhattan":
                    h = manhattanDistance(neighbor_state, goalState)
                else:
                    h = misplacedTiles(neighbor_state, goalState)
                neighbor_node = Node(neighbor_state, currNode, g, h)
                heapq.heappush(openList, neighbor_node)

    return None

initialState = []
goalState = []

print("Output: Vignesh B 1BM22CS326")
print("Enter the initial state (3x3 matrix, use 0 for the blank tile):")
for i in range(3):
    row = list(map(int, input().split()))
    initialState.append(row)
```

```
    print("Enter the goal state (3x3 matrix, use 0 for the blank tile):")
    for i in range(3):
        row = list(map(int, input().split()))
        goalState.append(row)

    # Prompt the user to choose the heuristic
    heuristic = input("Choose a heuristic (1 for Misplaced Tiles, 2 for Manhattan D
    if heuristic == '1':
        heuristic = "misplaced"
    elif heuristic == '2':
        heuristic = "manhattan"
    else:
        print("Invalid choice. Defaulting to Manhattan Distance.")
        heuristic = "manhattan"

    path = solve8puzzle(initialState, goalState, heuristic)

    if path:
        print("Solution found!")
        for i, state in enumerate(path):
            print(f"Step {i}:")
            for row in state:
                print(row)
        print("Number of moves:", len(path) - 1)
    else:
        print("No solution found.")
```

```
Output: Vignesh B 1BM22CS326
Enter the initial state (3x3 matrix, use 0 for the blank tile):
2 8 3
1 6 4
0 7 5
Enter the goal state (3x3 matrix, use 0 for the blank tile):
1 2 3
8 0 4
7 6 5
Choose a heuristic (1 for Misplaced Tiles, 2 for Manhattan Distance): 1
Solution found!
Step 0:
[2, 8, 3]
[1, 6, 4]
[0, 7, 5]
Step 1:
[2, 8, 3]
[1, 6, 4]
[7, 0, 5]
Step 2:
[2, 8, 3]
[1, 0, 4]
[7, 6, 5]
```

```
Step 3:
[2, 0, 3]
[1, 8, 4]
[7, 6, 5]
Step 4:
[0, 2, 3]
[1, 8, 4]
[7, 6, 5]
Step 5:
[1, 2, 3]
[0, 8, 4]
[7, 6, 5]
Step 6:
[1, 2, 3]
[8, 0, 4]
[7, 6, 5]
Number of moves: 6
```