```
from copy import deepcopy

initial_state = [
    [0,2,3],
    [1,4,6],
    [7,5,8]
]

# 0 is considered empty space

goal_state = [
    [1,2,3],
    [4,9,6],
    [7,8,5]
]

# Store the previously visited states to avoid infinite loops
state_history = []

INF = 9999

def manhattan(current_state):
    "Function to find the Manhattan distance"

    total_distance = 0

    for i in range(len(current_state)):
        for j in range(len(current_state[i])):

            # Calculate poistion in current state
            cur_pos = (i*3) + j + 1

            if current_state[i][j] == 0:
                continue

            for k in range(len(goal_state)):
                for l in range(len(goal_state[k])):

                    if current_state[i][j] == goal_state[k][l]:

                        #Calculate position in goal state
                        goal_pos = (k*3) + l + 1

                        abs_diff = abs(goal_pos-cur_pos)

                        if abs_diff >= 3:
                            total_distance += int(abs_diff/3) + abs_diff%3
                        else: total_distance += abs_diff

    return total_distance


def display():
    'Displays the current state'
    for i in range(len(initial_state)):
        for j in range(len(initial_state[i])):
            tile = initial_state[i][j]
            if tile == 0:
                print(" ", end=" ")
            else: print(tile, end=" ")
        print()
    print()



def moves():
    "Function to find all possible moves."
    # Returns integer array of all possible moves
    # 0 - up
    # 1 - right
    # 2 - down
    # 3 - left

    possible_moves = [0,1,2,3]

    # Finding the position of the empty space
```

```
        for i in range(len(initial_state)):
            for j in range(len(initial_state[i])):
                if initial_state[i][j] == 0:
                    if j == 2: possible_moves.remove(1)
                    if j == 0: possible_moves.remove(3)
                    if i == 0: possible_moves.remove(0)
                    if i == 2: possible_moves.remove(2)

                    return possible_moves

    def explore(move):
        'Function to explore a move and return its heuristic score'
        # Returns INF if the move has already been performed i.e if it exists in state history.

        current_state = deepcopy(initial_state)
        # Deepcopy is used to copy the list by value (recursively) instead of just copying the reference.

        # Finding the position of the empty space
        for i in range(len(current_state)):
            for j in range(len(current_state[i])):
                if current_state[i][j] == 0:
                    if move == 0:
                        #Swap it with the element above
                        current_state[i][j] = current_state[i-1][j]
                        current_state[i-1][j] = 0
                        if current_state not in state_history:
                            return manhattan(current_state)
                        return INF
                    elif move == 1:
                        #Swap it with the element right
                        current_state[i][j] = current_state[i][j+1]
                        current_state[i][j+1] = 0
                        if current_state not in state_history:
                            return manhattan(current_state)
                        return INF
                    elif move == 2:
                        #Swap it with the element down
                        current_state[i][j] = current_state[i+1][j]
                        current_state[i+1][j] = 0
                        if current_state not in state_history:
                            return manhattan(current_state)
                        return INF
                    elif move == 3:
                        #Swap it with the element left
                        current_state[i][j] = current_state[i][j-1]
                        current_state[i][j-1] = 0
                        if current_state not in state_history:
                            return manhattan(current_state)
                        return INF



    def execute(move):
        'Function to execute a move passed as arguement'

        state_history.append(deepcopy(initial_state))

        # Finding the position of the empty space
        for i in range(len(initial_state)):
            for j in range(len(initial_state[i])):
                if initial_state[i][j] == 0:
                    if move == 0:
                        #Swap it with the element above
                        initial_state[i][j] = initial_state[i-1][j]
                        initial_state[i-1][j] = 0
                    elif move == 1:
                        #Swap it with the element right
                        initial_state[i][j] = initial_state[i][j+1]
                        initial_state[i][j+1] = 0
                    elif move == 2:
                        #Swap it with the element down
                        initial_state[i][j] = initial_state[i+1][j]
                        initial_state[i+1][j] = 0
                    elif move == 3:
                        #Swap it with the element left
                        initial_state[i][j] = initial_state[i][j-1]
                        initial_state[i][j-1] = 0
```

```
                return


def __main__():
    'Solution of 8 Puzzle problem'

    display()

    while (initial_state != goal_state):
        possible_moves = moves()
        best_move = possible_moves[0]
        smallest_score = explore(best_move)

        for move in possible_moves:
            move_score = explore(move)
            if move_score < smallest_score:
                smallest_score = move_score
                best_move = move

        execute(best_move)
        display()



__main__()
```

Streaming output truncated to the last 5000 lines.

```
3 5 6

7 2 4
1 8
3 5 6

7 2
1 8 4
3 5 6

7 2 4
1 8
3 5 6

7 2
1 8 4
3 5 6

7 2 4
1 8
3 5 6

7 2
1 8 4
3 5 6

7 2 4
1 8
3 5 6

7 2
1 8 4
3 5 6

7 2 4
1 8
3 5 6

7 2
1 8 4
3 5 6

7 2 4
1 8
3 5 6

7 2
1 8 4
3 5 6

7 2 4
1 8
3 5 6
```

```
7 2
1 8 4
3 5 6
```