Using misplaced tiles for 8 puzzle game

Input –

```
from queue import PriorityQueue


class PuzzleState:
    def __init__(self, tiles, empty_tile_index, moves=0):
        self.tiles = tiles
        self.empty_tile_index = empty_tile_index
        self.moves = moves

    def is_goal(self):
        return self.tiles == GOAL_STATE

    def get_possible_moves(self):
        index = self.empty_tile_index
        possible_moves = []
        directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]
        for direction in directions:
            new_index = index + direction[0] * 3 + direction[1]
            if 0 <= new_index < 9:
                if direction[0] == -1 and index % 3 != 0: continue  # Up
                if direction[0] == 1 and index % 3 != 2: continue  # Down
                if direction[1] == -1 and index < 6: continue  # Left
                if direction[1] == 1 and index > 2: continue  # Right
                possible_moves.append(new_index)
        return possible_moves

    def generate_new_state(self, new_empty_index):
        new_tiles = list(self.tiles)
        new_tiles[self.empty_tile_index], new_tiles[new_empty_index] = new_tiles[new_empty_index],
new_tiles[self.empty_tile_index]
```

```
        return PuzzleState(tuple(new_tiles), new_empty_index, self.moves + 1)

    def heuristic(self):
        return sum(1 for i, tile in enumerate(self.tiles) if tile != 0 and tile != GOAL_STATE[i])

    def __lt__(self, other):
        return False  # This prevents direct comparison; we'll use tuples instead

def a_star(initial_state):
    open_set = PriorityQueue()
    open_set.put((0, initial_state))
    closed_set = set()

    while not open_set.empty():
        current_cost, current_state = open_set.get()

        if current_state.is_goal():
            return current_state.moves

        closed_set.add(current_state.tiles)

        for new_empty_index in current_state.get_possible_moves():
            new_state = current_state.generate_new_state(new_empty_index)

            if new_state.tiles in closed_set:
                continue

            cost = new_state.moves + new_state.heuristic()
            open_set.put((cost, new_state))

    return -1  # If no solution is found
```

```
if __name__ == "__main__":
    # User input for initial state
    initial_tiles = input("Enter the initial state (9 numbers separated by spaces, use 0 for the empty tile): ")
    initial_tiles = tuple(map(int, initial_tiles.split()))

    # User input for goal state
    goal_tiles = input("Enter the goal state (9 numbers separated by spaces, use 0 for the empty tile): ")
    global GOAL_STATE
    GOAL_STATE = tuple(map(int, goal_tiles.split()))

    empty_tile_index = initial_tiles.index(0)
    initial_state = PuzzleState(initial_tiles, empty_tile_index)

    result = a_star(initial_state)
    print(f"Minimum moves to solve the puzzle: {result}")
```

output-

```
In [3]: runfile('C:/Users/Admin/.spyder-py3/temp.py', wdir='C:/Users/Admin/.spyder-py3')

Enter the initial state (9 numbers separated by spaces, use 0 for the empty tile): 1 5 6 2 4 3 7 0 8

Enter the goal state (9 numbers separated by spaces, use 0 for the empty tile): 1 2 3 8 0 4 7 6 5
Minimum moves to solve the puzzle: -1

In [4]: runfile('C:/Users/Admin/.spyder-py3/temp.py', wdir='C:/Users/Admin/.spyder-py3')

Enter the initial state (9 numbers separated by spaces, use 0 for the empty tile): 1 2 3 4 5 6 7 8 0

Enter the goal state (9 numbers separated by spaces, use 0 for the empty tile): 1 2 3 4 5 6 7 0 8
Minimum moves to solve the puzzle: 1
```