```python
from heapq import heappush, heappop
goal_state = [[1, 2, 3],
              [4, 5, 6],
              [7, 8, 0]]

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

def manhattan_distance(state):
    distance = 0
    for i in range(3):
        for j in range(3):
            val = state[i][j]
            if val != 0:
                goal_i, goal_j = find_position(goal_state, val)
                distan
                ce += abs(i - goal_i) + abs(j - goal_j)
    return distance
def get_neighbors(state):
    neighbors = []
    x, y = find_position(state, 0)
    directions = [(-1,0),(1,0),(0,-1),(0,1)]
    for dx, dy in directions:
        new_x, new_y = x + dx, y + dy
        if 0 <= new_x < 3 and 0 <= new_y < 3:
            new_state = [row[:] for row in state]
            new_state[x][y], new_state[new_x][new_y] = new_state[new_x][new_y], new_state[x][y]
            neighbors.append(new_state)
    return neighbors

def state_to_tuple(state):
    return tuple(tuple(row) for row in state)
```

```python
def solve_puzzle(start_state):
    open_list = []
    closed_set = set()
    heappush(open_list, (manhattan_distance(start_state), 0, start_state, []))
    while open_list:
        est_total_cost, cost, state, path = heappop(open_list)
        if state == goal_state:
            return path + [state]
        closed_set.add(state_to_tuple(state))
        for neighbor in get_neighbors(state):
            if state_to_tuple(neighbor) not in closed_set:
                heappush(open_list, (
                    cost + 1 + manhattan_distance(neighbor),
                    cost + 1,
                    neighbor,
                    path + [state]
                ))
    return None

def print_state(state):
    for row in state:
        print(' '.join(str(x) for x in row))
    print()
start_state = [[1, 2, 3],
               [5, 0, 6],
               [4, 7, 8]]
solution = solve_puzzle(start_state)

if solution:
    print("Steps to solve the 8-puzzle:\n")
    for step, state in enumerate(solution):
        print(f"Step {step}:")
        print_state(state)
else:
    print("No solution found.")
```

```
>>> %Run -c $EDITOR_CONTENT

  Steps to solve the 8-puzzle:

  Step 0:
  1 2 3
  5 0 6
  4 7 8

  Step 1:
  1 2 3
  0 5 6
  4 7 8

  Step 2:
  1 2 3
  4 5 6
  0 7 8

  Step 3:
  1 2 3
  4 5 6
  7 0 8

  Step 4:
  1 2 3
  4 5 6
  7 8 0


>>>
```