```python
import heapq


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


moves = [(1,0), (-1,0), (0,1), (0,-1)]


def heuristic(state):
    distance = 0
    for i in range(3):
        for j in range(3):
            val = state[i][j]
            if val != 0:
                goal_x = (val-1)//3
                goal_y = (val-1)%3
                distance += abs(i-goal_x) + abs(j-goal_y)
    return distance


def to_tuple(state):
    return tuple([tuple(row) for row in state])
```

```python
def get_neighbors(state):
    x,y = find_blank(state)
    neighbors = []
    for dx,dy in moves:
        nx,ny = x+dx, y+dy
        if 0 <= nx < 3 and 0 <= ny < 3:
            new_state = [row[:] for row in state]
            new_state[x][y], new_state[nx][ny] = new_state[nx][ny], new_state[x][y]
            neighbors.append(new_state)
    return neighbors

def solve_puzzle(start_state):
    pq = []
    heapq.heappush(pq, (heuristic(start_state), 0, start_state, []))
    visited = set()

    while pq:
        f, g, state, path = heapq.heappop(pq)

        if state == goal_state:
            return path + [state]

        if to_tuple(state) in visited:
            continue
        visited.add(to_tuple(state))
```

```python
start_state = [[1,2,3],
               [4,0,6],
               [7,5,8]]

solution = solve_puzzle(start_state)

if solution:
    print("Steps to solve 8-Puzzle:")
    for step in solution:
        for row in step:
            print(row)
        print("-----")
else:
    print("No solution found!")
```

```
Steps to solve 8-Puzzle:
[1, 2, 3]
[4, 0, 6]
[7, 5, 8]
-----
[1, 2, 3]
[4, 5, 6]
[7, 0, 8]
-----
[1, 2, 3]
[4, 5, 6]
[7, 8, 0]
-----
```