Method 2.Manhattan Distance

```
class Node:
    def __init__(self, state, parent=None, move=None, cost=0):
        self.state = state          # The current state of the puzzle
        self.parent = parent        # The parent node
        self.move = move            # The move taken to reach this state
        self.cost = cost            # The cost to reach this node

    def heuristic(self):
        """Calculate the Manhattan distance for the current state."""
        goal_positions = {
            1: (0, 0), 2: (0, 1), 3: (0, 2),
            4: (1, 0), 5: (1, 1), 6: (1, 2),
            7: (2, 0), 8: (2, 1), 0: (2, 2)
        }
        distance = 0
        for i in range(len(self.state)):
            for j in range(len(self.state[i])):
                tile = self.state[i][j]
                if tile != 0:
                    goal_x, goal_y = goal_positions[tile]
                    distance += abs(goal_x - i) + abs(goal_y - j)
        return distance

def get_blank_position(state):
    """Find the position of the blank (0) in the state."""
    for i in range(len(state)):
        for j in range(len(state[i])):
            if state[i][j] == 0:
                return i, j

def get_possible_moves(position):
```

```
    """Get possible moves from the blank position."""
    x, y = position
    moves = []
    if x > 0: moves.append((x - 1, y, 'Down'))  # Up
    if x < 2: moves.append((x + 1, y, 'Up'))    # Down
    if y > 0: moves.append((x, y - 1, 'Right'))  # Left
    if y < 2: moves.append((x, y + 1, 'Left'))   # Right
    return moves

def generate_new_state(state, blank_pos, new_blank_pos):
    """Generate a new state by moving the blank tile."""
    new_state = [row[:] for row in state]  # Deep copy
    new_state[blank_pos[0]][blank_pos[1]], new_state[new_blank_pos[0]][new_blank_pos[1]]
= \
        new_state[new_blank_pos[0]][new_blank_pos[1]],
new_state[blank_pos[0]][blank_pos[1]]
    return new_state

def a_star_search(initial_state):
    """Perform A* search."""
    open_list = []
    closed_list = set()

    initial_node = Node(state=initial_state, cost=0)
    open_list.append(initial_node)

    while open_list:
        # Sort the open list by total estimated cost (cost + heuristic)
        open_list.sort(key=lambda node: node.cost + node.heuristic())
        current_node = open_list.pop(0)

        # Print the current state, move, and heuristic value
        move_description = current_node.move if current_node.move else "Start"
```

```
        print("Current state:")
        for row in current_node.state:
            print(row)
        print(f"Move: {move_description}")
        print(f"Heuristic value (Manhattan distance): {current_node.heuristic()}")
        print(f"Cost to reach this node: {current_node.cost}\n")

        if current_node.heuristic() == 0:  # Goal state reached
            # Construct the path
            path = []
            while current_node:
                path.append(current_node)
                current_node = current_node.parent
            return path[::-1]  # Return reversed path

        closed_list.add(tuple(map(tuple, current_node.state)))

        blank_pos = get_blank_position(current_node.state)
        for new_blank_pos in get_possible_moves(blank_pos):
            new_state = generate_new_state(current_node.state, blank_pos, (new_blank_pos[0],
new_blank_pos[1]))

            if tuple(map(tuple, new_state)) in closed_list:
                continue

            cost = current_node.cost + 1
            move_direction = new_blank_pos[2]  # Get the direction of the move
            new_node = Node(state=new_state, parent=current_node, move=move_direction,
cost=cost)

            if new_node not in open_list:  # Avoid duplicates in the open list
                open_list.append(new_node)
```

```
    return None  # No solution found

# Example usage:
initial_state = [[1, 2, 3], [4, 0, 5], [7, 8, 6]]  # An example initial state
solution_path = a_star_search(initial_state)

if solution_path:
    print("Solution path:")
    for step in solution_path:
        for row in step.state:
            print(row)
        print()
else:
    print("No solution found.")
```

Output:

```
Current state:
[1, 2, 3]
[4, 0, 5]
[7, 8, 6]
Move: Start
Heuristic value (Manhattan distance): 2
Cost to reach this node: 0

Current state:
[1, 2, 3]
[4, 5, 0]
[7, 8, 6]
Move: Left
Heuristic value (Manhattan distance): 1
Cost to reach this node: 1
```

Current state:

[1, 2, 3]

[4, 5, 6]

[7, 8, 0]

Move: Up

Heuristic value (Manhattan distance): 0

Cost to reach this node: 2

Solution path:

[1, 2, 3]

[4, 0, 5]

[7, 8, 6]

[1, 2, 3]

[4, 5, 0]

[7, 8, 6]

[1, 2, 3]

[4, 5, 6]

[7, 8, 0]