Experiment 1: 8-Puzzle Program

Aim: Implement an Algorithm in Python for solving the 8-Puzzle Problem.

```
import heapq

# Define the goal state
goal_state = [[1, 2, 3],
              [4, 5, 6],
              [7, 8, 0]]

# Define the possible moves
moves = [(0, 1), (1, 0), (0, -1), (-1, 0)]

# Define move names for output
move_names = {(0, 1): 'right', (1, 0): 'down', (0, -1): 'left', (-1, 0): 'up'}
def get_blank_position(board):
    for i in range(3):
        for j in range(3):
            if board[i][j] == 0:
                return i, j
def get_heuristic(board):
    count = 0
    for i in range(3):
        for j in range(3):
            if board[i][j] != goal_state[i][j]:
                count += 1
    return count
def is_valid_move(x, y):
    return 0 <= x < 3 and 0 <= y < 3



def generate_children(board, current_cost):
    children = []
    x, y = get_blank_position(board)
    for dx, dy in moves:
        new_x, new_y = x + dx, y + dy
        if is_valid_move(new_x, new_y):
            new_board = [row[:] for row in board]
            new_board[x][y], new_board[new_x][new_y] = new_board[new_x][new_y],
```

```
new_board[x][y]
            children.append((new_board, (dx, dy), current_cost + 1))
    return children


def a_star(initial_state):
    open_list = []
    closed_set = set()
    heapq.heappush(open_list, (get_heuristic(initial_state), 0, initial_state, []))
    while open_list:
        _, cost, current_state, path = heapq.heappop(open_list)
        if current_state == goal_state:
            return path
        if tuple(map(tuple, current_state)) in closed_set:
            continue
        closed_set.add(tuple(map(tuple, current_state)))
        for child_state, move, move_cost in generate_children(current_state, cost):
            if tuple(map(tuple, child_state)) not in closed_set:
                heapq.heappush(open_list, (move_cost + get_heuristic(child_state), move_cost,
child_state, path + [(move_names[(move[0], move[1])], move_cost)]))
    return []


# Example usage:
initial_state = [[1, 2, 3],
                 [4, 5, 6],
                 [0, 7, 8]]

steps = a_star(initial_state)
if steps:
    print("Steps required to solve the puzzle:")
    total_cost = 0
    for i, (move, cost) in enumerate(steps):
        total_cost += cost
        print(f"{i + 1}. Move {move}, Cost: {cost}, Total Cost: {total_cost}")
else:
    print("No solution exists.")
```

Output:

Enter initial state:

1 2 3

0 4 6

7 5 8

costs: 3

3 1 2

3 0 4

6 7 5

8 1 2

3 4 0

6 7 5

8 1 2

3 4 5

6 7 0

1 2 3

4 5 6

7 8 0

Result:

Code has been Implemented successfully.