```python
# Dinesh Rathod (TA57) - Experiment 2
import heapq

# Define the 8-Puzzle class
class EightPuzzle:
    def __init__(self, initial_state):
        self.initial_state = initial_state
        self.goal_state = [1, 2, 3, 8, 0, 4, 7, 6, 5]  # The goal state
        self.actions = [(1, 0), (-1, 0), (0, 1), (0, -1)]  # Possible move directions

    def is_valid(self, state):
        # Check if a given state is a valid state
        return set(state) == set(self.goal_state)

    def get_blank_position(self, state):
        # Find the position of the blank (0) in the puzzle
        return state.index(0) // 3, state.index(0) % 3

    def get_neighbors(self, state):
        # Generate neighboring states by moving the blank tile
        x, y = self.get_blank_position(state)
        neighbors = []

        for dx, dy in self.actions:
            new_x, new_y = x + dx, y + dy

            if 0 <= new_x < 3 and 0 <= new_y < 3:
                new_state = state[:]
                new_index = new_x * 3 + new_y
                new_state[x * 3 + y], new_state[new_index] = new_state[new_index],
new_state[x * 3 + y]
                neighbors.append(new_state)

        return neighbors

    def heuristic(self, state):
```

```
        # A heuristic function for estimating the cost to reach the goal state
        h = 0
        for i in range(9):
            if state[i] != self.goal_state[i]:
                h += 1
        return h

    def a_star_search(self):
        open_list = [(self.initial_state, 0, self.heuristic(self.initial_state))]
        closed_set = set()
        g_values = {tuple(self.initial_state): 0}

        while open_list:
            state, g, h = heapq.heappop(open_list)

            if self.is_valid(state):
                return state

            if state in closed_set:
                continue

            closed_set.add(state)

            for neighbor in self.get_neighbors(state):
                if neighbor not in closed_set:
                    new_g = g + 1
                    if new_g < g_values.get(tuple(neighbor), float('inf')):
                        g_values[tuple(neighbor)] = new_g
                        f = new_g + self.heuristic(neighbor)
                        heapq.heappush(open_list, (neighbor, new_g, f))

        return None


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

```
puzzle = EightPuzzle(initial_state)
solution = puzzle.a_star_search()

if solution:
    print("Solution found:")
    for i in range(0, 9, 3):
        print(solution[i:i+3])
else:
    print("No solution found.")
```

Output:

```
Solution found:
[2, 8, 3]
[1, 6, 4]
[7, 0, 5]
```