```
############################################################
#                                                          #
#                       EXPERIMENT 6                       #
#                     8 Puzzle Problem                     #
#                  Nathan Cordeiro 22co09                  #
#                                                          #
############################################################

from collections import deque

# Define the goal state as a 3x3 matrix
GOAL_MATRIX = [[1, 2, 3], [4, 5, 6], [7, 8, 0]]

# Convert a matrix to a tuple for use in sets and dictionaries
def matrix_to_tuple(matrix):
    return tuple(tuple(row) for row in matrix)

def tuple_to_matrix(tpl):
    return [list(row) for row in tpl]

# Function to get the position of the blank space (0)
def find_blank(matrix):
    for r in range(3):
        for c in range(3):
            if matrix[r][c] == 0:
                return r, c

# Function to get all possible moves from the current matrix
def get_neighbors(matrix):
    neighbors = []
    row, col = find_blank(matrix)

    moves = [(-1, 0), (1, 0), (0, 1), (0, -1)]  # Up, Down, Left, Right

    for dr, dc in moves:
        new_row, new_col = row + dr, col + dc
        if 0 <= new_row < 3 and 0 <= new_col < 3:
            # Create a new matrix by swapping the blank space with the target tile
            new_matrix = [list(row) for row in matrix]
            new_matrix[row][col], new_matrix[new_row][new_col] = new_matrix[new_row][new_col],
new_matrix[row][col]
            neighbors.append(matrix_to_tuple(new_matrix))

    return neighbors

# Breadth-First Search implementation
def bfs(start_matrix):
```

```
    start_tuple = matrix_to_tuple(start_matrix)
    goal_tuple = matrix_to_tuple(GOAL_MATRIX)

    queue = deque([(start_tuple, [])])  # Queue of (matrix, path_to_matrix)
    visited = set()
    visited.add(start_tuple)

    while queue:
        current_matrix, path = queue.popleft()

        if current_matrix == goal_tuple:
            return path + [current_matrix]

        for neighbor in get_neighbors(tuple_to_matrix(current_matrix)):
            if neighbor not in visited:
                visited.add(neighbor)
                queue.append((neighbor, path + [current_matrix]))

    return None

# Main function to run the program
if __name__ == "__main__":
    start_matrix = [[1, 2, 3], [4, 5, 6], [0, 7,8 ]]  # Example start state
    solution = bfs(start_matrix)

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

OUTPUT:

```
Solution found:
(1, 2, 3)
(4, 5, 6)
(0, 7, 8)

(1, 2, 3)
(4, 5, 6)
(7, 0, 8)

(1, 2, 3)
(4, 5, 6)
(7, 8, 0)
```