(1) 8 PUZZLE GAME USING DFS

```
class PuzzleState:
    def __init__(self, board, empty_tile_pos, path=[]):
        self.board = board
        self.empty_tile_pos = empty_tile_pos
        self.path = path

    def is_goal(self):
        return self.board == [1, 2, 3, 4, 5, 6, 7, 8, 0]

    def get_possible_moves(self):
        moves = []
        x, y = self.empty_tile_pos
        directions = [(0, 1), (1, 0), (0, -1), (-1, 0)]  # right, down, left, up
        for dx, dy in directions:
            new_x, new_y = x + dx, y + dy
            if 0 <= new_x < 3 and 0 <= new_y < 3:
                moves.append((new_x, new_y))
        return moves

    def make_move(self, new_empty_tile_pos):
        new_board = self.board[:]
        x, y = self.empty_tile_pos
        new_x, new_y = new_empty_tile_pos
        new_board[x * 3 + y], new_board[new_x * 3 + new_y] = new_board[new_x * 3 + new_y],
new_board[x * 3 + y]
        return PuzzleState(new_board, new_empty_tile_pos, self.path + [new_board])

def dfs(initial_state):
    stack = [initial_state]
    visited = set()
```

```
    while stack:
        state = stack.pop()
        if state.is_goal():
            return state.path
        visited.add(tuple(state.board))

        for move in state.get_possible_moves():
            new_state = state.make_move(move)
            if tuple(new_state.board) not in visited:
                stack.append(new_state)
    return None

# Example usage
initial_board = [1, 2, 3, 4, 5, 6, 0, 7, 8]  # The '0' represents the empty tile
empty_tile_pos = (2, 0)  # Position of '0'

initial_state = PuzzleState(initial_board, empty_tile_pos)
solution_path = dfs(initial_state)

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

OUTPUT:

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

=== Code Execution Successful ===
```