

```

class PuzzleState:
    def __init__(self, board, depth=0):
        self.board = board
        self.empty_tile_pos = self.find_empty_tile()
        self.depth = depth

    def find_empty_tile(self):
        for i in range(3):
            for j in range(3):
                if self.board[i][j] == 0:
                    return (i, j)

    def is_goal(self, goal_state):
        return self.board == goal_state

    def generate_successors(self):
        successors = []
        x, y = self.empty_tile_pos
        moves = [(-1, 0), (1, 0), (0, -1), (0, 1)]
        for dx, dy in moves:
            new_x, new_y = x + dx, y + dy
            if 0 <= new_x < 3 and 0 <= new_y < 3:
                new_board = [row[:] for row in self.board]
                new_board[x][y], new_board[new_x][new_y] = new_board[new_x][new_y], new_board[x][y]
                successors.append(PuzzleState(new_board, self.depth + 1))
        return successors

def depth_limited_search(state, goal_state, depth):
    if state.is_goal(goal_state):
        return state
    if depth == 0:
        return None

    for successor in state.generate_successors():
        result = depth_limited_search(successor, goal_state, depth - 1)
        if result is not None:
            return result

    return None

def iterative_deepening_search(initial_state, goal_state, max_depth):
    for depth_limit in range(max_depth + 1):
        result = depth_limited_search(initial_state, goal_state, depth_limit)

```

```

        if result is not None:
            return True
    return False


initial_board = [
    [1, 2, 3],
    [0, 4, 6],
    [7, 5, 8]
]

goal_board = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 0]
]

initial_state = PuzzleState(initial_board)
max_depth = 3

if iterative_deepening_search(initial_state, goal_board, max_depth):
    print("Goal is reachable within the maximum depth.")
else:
    print("Goal is not reachable within the maximum depth.")

```

 Goal is reachable within the maximum depth.

```

class PuzzleState:
    def __init__(self, board, depth=0):
        self.board = board
        self.empty_tile_pos = self.find_empty_tile()
        self.depth = depth

    def find_empty_tile(self):
        for i in range(3):
            for j in range(3):
                if self.board[i][j] == 0:
                    return (i, j)

    def is_goal(self, goal_state):
        return self.board == goal_state

    def generate_successors(self):
        successors = []
        x, y = self.empty_tile_pos
        moves = [(-1, 0), (1, 0), (0, -1), (0, 1)]
        for dx, dy in moves:
            new_x, new_y = x + dx, y + dy
            if 0 <= new_x < 3 and 0 <= new_y < 3:
                new_board = [row[:] for row in self.board]
                new_board[x][y], new_board[new_x][new_y] = new_board[new_x][new_y], new_board[x][y]
                successors.append(PuzzleState(new_board, self.depth + 1))
        return successors

def depth_limited_search(state, goal_state, depth):
    if state.is_goal(goal_state):
        return state
    if depth == 0:
        return None

    for successor in state.generate_successors():
        result = depth_limited_search(successor, goal_state, depth - 1)
        if result is not None:
            return result

    return None

def iterative_deepening_search(initial_state, goal_state, max_depth):
    for depth_limit in range(max_depth + 1):
        result = depth_limited_search(initial_state, goal_state, depth_limit)
        if result is not None:
            return True
    return False


initial_board = [
    [1, 2, 3],
    [0, 4, 6],
    [7, 5, 8]
]

goal_board = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 0]
]

initial_state = PuzzleState(initial_board)
max_depth = 3

if iterative_deepening_search(initial_state, goal_board, max_depth):
    print("Goal is reachable within the maximum depth.")
else:
    print("Goal is not reachable within the maximum depth.")

```

 Goal is reachable within the maximum depth.