Lab 3:-

8-Puzzel Game

Code:-

Using Depth First Search (DFS)

```
class SlidingPuzzle:
    def __init__(self, board, empty_pos, path=[]):
        self.board = board
        self.empty_pos = empty_pos
        self.path = path

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

    def get_moves(self):
        x, y = self.empty_pos
        possible_moves = []
        for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
            nx, ny = x + dx, y + dy
            if 0 <= nx < 3 and 0 <= ny < 3:
                new_board = self.board[:]
                new_board[x * 3 + y], new_board[nx * 3 + ny] = new_board[nx * 3 + ny], new_board[x * 3 +
y]
                possible_moves.append((new_board, (nx, ny)))
        return possible_moves

def depth_first_search(initial_puzzle):
    stack, visited = [initial_puzzle], set()
    while stack:
        current_puzzle = stack.pop()
        if current_puzzle.is_solved():
            return current_puzzle.path
```

```
        visited.add(tuple(current_puzzle.board))
        for new_board, new_empty_pos in current_puzzle.get_moves():
            new_state = SlidingPuzzle(new_board, new_empty_pos, current_puzzle.path + [new_board])
            if tuple(new_board) not in visited:
                stack.append(new_state)
    return None

def display_board(board):
    for i in range(0, 9, 3):
        print(board[i:i + 3])
    print()

def main():
    initial_board = [1, 2, 3, 4, 0, 5, 7, 8, 6]
    empty_pos = initial_board.index(0)
    initial_puzzle = SlidingPuzzle(initial_board, (empty_pos // 3, empty_pos % 3))

    print("Initial state:")
    display_board(initial_board)

    solution = depth_first_search(initial_puzzle)

    if solution:
        print("Solution found:")
        for step in solution:
            display_board(step)
    else:
        print("No solution found.")

if __name__ == "__main__":
```

```
main()
```

Output:-

```
Type "help", "copyright", "credits" or "license()" for mor
>>>
===== RESTART: C:/Users/User/AppData/Local/Programs/Python
Initial state:
[1, 2, 3]
[4, 0, 5]
[7, 8, 6]

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

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

>>> |
```

Code:-

Using Manhattan Distance

```
class SlidingPuzzleSolver:

    def __init__(self, initial_state):

        self.initial_state = initial_state

        self.goal_state = [[1, 2, 3], [4, 5, 6], [7, 8, 0]]


    def manhattan_distance(self, state):

        distance = 0

        for i in range(3):

            for j in range(3):

                if state[i][j] != 0:

                    goal_i = (state[i][j] - 1) // 3

                    goal_j = (state[i][j] - 1) % 3

                    distance += abs(i - goal_i) + abs(j - goal_j)

        return distance
```

```
    def get_neighbors(self, state):
        i, j = next((i, j) for i in range(3) for j in range(3) if state[i][j] == 0)
        moves = [(i - 1, j), (i + 1, j), (i, j - 1), (i, j + 1)]
        return [self.swap(state, i, j, x, y) for x, y in moves if 0 <= x < 3 and 0 <= y < 3]

    def swap(self, state, i1, j1, i2, j2):
        new_state = [row[:] for row in state]
        new_state[i1][j1], new_state[i2][j2] = new_state[i2][j2], new_state[i1][j1]
        return new_state

    def dfs_with_manhattan(self, state, visited=set()):
        if state == self.goal_state:
            return [state]
        visited.add(str(state))
        neighbors = sorted(self.get_neighbors(state), key=lambda x: self.manhattan_distance(x))
        for neighbor in neighbors:
            if str(neighbor) not in visited:
                path = self.dfs_with_manhattan(neighbor, visited)
                if path:
                    return [state] + path
        return None

    def solve(self):
        solution = self.dfs_with_manhattan(self.initial_state)
        return solution


initial_state = [[int(x) for x in input(f"Enter row {i + 1}: ").split()] for i in range(3)]
solver = SlidingPuzzleSolver(initial_state)
solution = solver.solve()
```

```
if solution:
    print("Solution found:")
    for state in solution:
        print(*state, sep='\n', end='\n\n')
else:
    print("No solution found.")
```

Output:-

```
>>
    ===== RESTART: C:/Users/User/AppData/Local/Programs/I
    Enter row 1: 1 0 3
    Enter row 2: 4 2 6
    Enter row 3: 7 5 8
    Solution found:
    [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]
```