AI PRACTICAL NO. 5

Name:Girish Nhavkar

Roll no:9560

Div:TE COMPS A (BATCH B)

PART 1:

```
import heapq

# function to calculate the manhattan distance
def manhattan_distance(puzzle, goal):
    distance = 0
    for i in range(9):
        if puzzle[i] == 0 or goal[i] == 0:
            continue
        x1, y1 = divmod(i, 3)
        x2, y2 = divmod(puzzle.index(goal[i]), 3)
        distance += abs(x1 - x2) + abs(y1 - y2)
    return distance

# function to check if the puzzle is solvable or not
# The puzzle is solvable if the number of inversions(the number of pairs of
tiles that are out of order) is even.
def is_solvable(puzzle, goal):
    puzzle = [i for i in puzzle if i != 0]
    inversions = 0
    for i in range(len(puzzle) - 1):
        for j in range(i + 1, len(puzzle)):
            if puzzle[i] > puzzle[j]:
                inversions += 1
                print(inversions)
    return inversions % 2 == 0

# function to solve the 8 puzzle game
def solve_puzzle(puzzle, goal):
    if not is_solvable(puzzle, goal):
        return [], []
    heap = []
    heapq.heappush(heap, (0, puzzle, []))
    visited = set()
    while heap:
        cost, current, path = heapq.heappop(heap)
        if current == goal:
            return cost, path
        index = current.index(0)
        x, y = divmod(index, 3)
```

```
        for dx, dy in ((1, 0), (-1, 0), (0, 1), (0, -1)):
            x2, y2 = x + dx, y + dy
            if 0 <= x2 < 3 and 0 <= y2 < 3:
                next_puzzle = list(current)
                next_index = x2 * 3 + y2
                next_puzzle[index], next_puzzle[next_index] =
next_puzzle[next_index], next_puzzle[index]
                next_path = path + [(x, y, x2, y2)]
                next_cost = cost + 1 + manhattan_distance(next_puzzle, goal)
                if tuple(next_puzzle) not in visited:
                    visited.add(tuple(next_puzzle))
                    heapq.heappush(heap, (next_cost, next_puzzle, next_path))
    return [], []

if __name__ == '__main__':
    puzzle = [1, 2, 3, 0, 4, 6, 7, 5, 8]
    goal = [1, 2, 3, 4, 5, 6, 7, 8, 0]
    cost, path = solve_puzzle(puzzle, goal)
    if cost:
        print('Found solution with cost', cost)
        print('\nStart State:')
        for i in range(3):
            print(puzzle[i*3:i*3+3])
        for step, move in enumerate(path, start=1):
            x1, y1, x2, y2 = move
            puzzle[x1*3+y1], puzzle[x2*3+y2] = puzzle[x2*3+y2],
puzzle[x1*3+y1]
            print(f'\nStep {step}:')
            for i in range(3):
                print(puzzle[i*3:i*3+3])
    else:
        print('No solution found')
```

OP

```
1
2
Found solution with cost 4

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

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

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

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

PART 2:

```
import heapq

# function to calculate the manhattan distance
def manhattan_distance(puzzle, goal):
    distance = 0
    for i in range(9):
        if puzzle[i] == 0 or goal[i] == 0:
            continue
        x1, y1 = divmod(i, 3)
        x2, y2 = divmod(puzzle.index(goal[i]), 3)
        distance += abs(x1 - x2) + abs(y1 - y2)
    return distance

# function to check if the puzzle is solvable or not
def is_solvable(puzzle, goal):
    puzzle = [i for i in puzzle if i != 0]
    inversions = 0
    for i in range(len(puzzle) - 1):
        for j in range(i + 1, len(puzzle)):
            if puzzle[i] > puzzle[j]:
                inversions += 1
    print("Number of inversions: ",inversions)
    return inversions % 2 == 0

# function to solve the 8 puzzle game
def solve_puzzle(puzzle, goal):
    if not is_solvable(puzzle, goal):
        return [], []
    heap = []
    heapq.heappush(heap, (0, puzzle, []))
    visited = set()
    while heap:
        cost, current, path = heapq.heappop(heap)
        if current == goal:
            return cost, path
        index = current.index(0)
        x, y = divmod(index, 3)
        for dx, dy in ((1, 0), (-1, 0), (0, 1), (0, -1)):
            x2, y2 = x + dx, y + dy
            if 0 <= x2 < 3 and 0 <= y2 < 3:
                next_puzzle = list(current)
                next_index = x2 * 3 + y2
                next_puzzle[index], next_puzzle[next_index] =
next_puzzle[next_index], next_puzzle[index]
                next_path = path + [(x, y, x2, y2)]
                next_cost = cost + 1 + manhattan_distance(next_puzzle, goal)
                if tuple(next_puzzle) not in visited:
```

```
                    visited.add(tuple(next_puzzle))
                    heapq.heappush(heap, (next_cost, next_puzzle, next_path))
    return [], []

if __name__ == '__main__':
    puzzle = [1, 2, 3, 7, 4, 6, 0, 5, 8]
    goal = [1, 2, 3, 4, 5, 6, 7, 8, 0]
    cost, path = solve_puzzle(puzzle, goal)
    if cost:
        print('\nFound solution with cost', cost)
        print('\nStart State:')
        for i in range(3):
            print(puzzle[i*3:i*3+3])
        for step, move in enumerate(path, start=1):
            x1, y1, x2, y2 = move
            puzzle[x1*3+y1], puzzle[x2*3+y2] = puzzle[x2*3+y2],
puzzle[x1*3+y1]
            print(f'\nStep {step}:')
            for i in range(3):
                print(puzzle[i*3:i*3+3])
    else:
        print('No solution found')
```

OP

```
Number of inversions:  4

Found solution with cost 7

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

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

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

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

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

PART 3

```
import heapq

# function to calculate the manhattan distance
def manhattan_distance(puzzle, goal):
    distance = 0
    for i in range(9):
```

```
        if puzzle[i] == 0 or goal[i] == 0:
            continue
        x1, y1 = divmod(i, 3)
        x2, y2 = divmod(puzzle.index(goal[i]), 3)
        distance += abs(x1 - x2) + abs(y1 - y2)
    return distance

# function to check if the puzzle is solvable or not
def is_solvable(puzzle, goal):
    puzzle = [i for i in puzzle if i != 0]
    inversions = 0
    for i in range(len(puzzle) - 1):
        for j in range(i + 1, len(puzzle)):
            if puzzle[i] > puzzle[j]:
                inversions += 1
    print("Number of inversions: ",inversions)
    return inversions % 2 == 0

# function to solve the 8 puzzle game
def solve_puzzle(puzzle, goal):
    if not is_solvable(puzzle, goal):
        return [], []
    heap = []
    heapq.heappush(heap, (0, puzzle, []))
    visited = set()
    while heap:
        cost, current, path = heapq.heappop(heap)
        if current == goal:
            return cost, path
        index = current.index(0)
        x, y = divmod(index, 3)
        for dx, dy in ((1, 0), (-1, 0), (0, 1), (0, -1)):
            x2, y2 = x + dx, y + dy
            if 0 <= x2 < 3 and 0 <= y2 < 3:
                next_puzzle = list(current)
                next_index = x2 * 3 + y2
                next_puzzle[index], next_puzzle[next_index] =
next_puzzle[next_index], next_puzzle[index]
                next_path = path + [(x, y, x2, y2)]
                next_cost = cost + 1 + manhattan_distance(next_puzzle, goal)
                if tuple(next_puzzle) not in visited:
                    visited.add(tuple(next_puzzle))
                    heapq.heappush(heap, (next_cost, next_puzzle, next_path))
    return [], []

if __name__ == '__main__':
    puzzle = [1, 2, 3, 4, 5, 6, 8, 7, 0]
    goal = [1, 2, 3, 4, 5, 6, 7, 8, 0]
```

```
    cost, path = solve_puzzle(puzzle, goal)
    if cost:
        print('\nFound solution with cost', cost)
        print('\nStart State:')
        for i in range(3):
            print(puzzle[i*3:i*3+3])
        for step, move in enumerate(path, start=1):
            x1, y1, x2, y2 = move
            puzzle[x1*3+y1], puzzle[x2*3+y2] = puzzle[x2*3+y2],
puzzle[x1*3+y1]
            print(f'\nStep {step}:')
            for i in range(3):
                print(puzzle[i*3:i*3+3])
    else:
        print('No solution found')
```

OP

Number of inversions: 1

No solution found