Name: Rohit Kudache

USN : 1BM18CS083

LAB : AI LAB TEST - 1

Program : 8-puzzle using A^* algorithm

program :

```
def process(self):
    # accept start and goal state
    print("Enter start state \n")
    start = self.accept()
    print("Enter goal state \n")
    goal = self.accept()

    start = Node(start, 0, 0)
    start.fval = self.f(start, goal)
# put start node in open List
    print("\n")
    self.open.append(start)
    while True:
        cur = self.open[0]
        for i in cur.data:
            for j in i:
                print(j, end=" ")
            print(" ")
# if h value of node is 0 then we reached
# goal state.
        if(self.h(cur.data, goal) == 0):
            break
```

Rohit

13/11/2020

Continued ---

```
        for i in cur.generate_child():
            i.fval = self.f(isGoal)
            self.open.append(i)
        self.closed.append(cur)
        del self.open[0]

        # sorting open list based on f value
        # f(n) = h(n) + g(n)
        self.open.sort(key=lambda x: x.fval, reverse=False)

puz = puzzle(3)
puz.process()

# function f
    def f(self, start, goal):
        return self.h(start.data, goal) + start.level

# function h
    def h(self, start, goal):
        # manhattan distance abs |(x2 - x1)| and
        # abs |y2 - y1| or number of misplaced tiles.
        temp = 0
        for i in range(0, self.n):
            for j in range(0, self.n):
                if start[i][j] != goal[i][j] and
                    start[i][j] != '_':
                    temp += 1
        return temp.
```

Rohit

13/11/20

Rohit Kudache (1BM18CS083)

```
def astar(start, goal):
    States = [start]
    g = 0
    visited_state = set()
    while len(state):
        print(f"level : {g}")
    moves = []
for state in states:
    visited_state.add(tuple(state))
    print_grid(state)
    if state = goal
        print("success")
        return
move t = [move for move in possible_moves
                (state, visited_state) if move
                    not in moves]
costs = [g + h(move, goal) for move in moves]
states = [moves[i] for i in range[len(moves) if
                cost[i] == min(costs)]
g t = 1
print("NOSOLUTION")
# We will implement function for possible
# moves and gen.
```

03

13/11/20