13/11/20 AI Lab Test 1

Sanjeev Kumar Singh
1BM18CS093
B3

~~Program~~ Program 1

Writeup

Creating function that finds an total estimated cost through node n

1) def h(State, Start)
// Manhattan distance
dist = 0

for i in State :

d1, d2 = State index (i), target, index (i)
x1, y1 = d1 // 3, d1 % 3
x2, y2 = d2 // 3, d2 % 3

dist += abs (x1 - x2) + abs (y1 - y2)
return dist

2) Create Search fun.

To traverse across tree using f(n) to select next node

a) Make sure it to discard visited sites
b) Create possible move fun
c) Create move Generator fun

def astar (Src, target)
States = [Src]
g = 0
Visited_States = set()
while len (States):
print (f"Level : {g}")
moves = []
for State in States :

1BM18CS093 Sanjeev Kumar Singh

~~1BM18CS093~~ 33

```
Visited_States.add(tuple(State))
print_grid(State)
if State == target:
print("Success")
return
moves += [move for move in possible_moves(State, Visited_States) if move not in moves]
Costs = [g + h(move, target) for move in moves]
States = [moves[i] for i in range(len(moves)) if Costs[i] == min(Costs)]
g += 1
print("No Solution")
// def possible_moves(State, Visited_States):
// def gen(State, direction, b)
```

Sanjeev Kumar Singh
1BM18CS093

```
class Puzzle:
    def __init__(self, size):
        self.n = size
        self.open = []
        self.closed = []

    def accept(self):
        puz = []
        for i in range(0, self.n):
            temp = input().split(" ")
            puz.append(temp)
        return puz

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

    def h(self, start, goal):
        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
```


Sanjeev

Sanjeev Kumar Singh
1BM18CS093

```
def process(self):
print("Enter start matrix \n")
    start = self.accept()
    print("Enter Goal matrix \n")
     goal = self.accept()
start = node(start,0,0)
    start.fvd = self.f(start, goal)
    self.open.append(start)
     print("\n\n")
     while True:
     curr = self.open[0]
       print("  ")
       print("  |  ")
       print("  |  ")
        for i in curr.data
     for j in i
          print(j, end = " ")
           print("  ")
if(self.h(curr.data, goal)==0);
     break.
for i in curr.generate_child():
     i.fvd = self.f(i, goal)
     self.open.append(i)
     self.closed.append(curr)
     del self.open[0]
self.open.sort(key = lambda x: x.fvd, reverse=False)
     puz = puzzle(3)
      puz.process()
```


Sanjeev

AI Lab Test 1

Sangeev Kumar Singh
1BM18CS093

Main fun

```
def index(mylist, v)
    for i, v in enumerate(mylist):
        if v in n:
        return (i, n.index(v))
def Manhattan(temp):
    sum=0
    for i in range(3):
    for j in range(3):
    if temp[i][j] != 0:
        b= index(temp, temp[i][j])
        c= index(goal, temp[i][j])
        sum += abs(b[0]-c[0])
        sum += (abs(b[1]-c[1]))
    return sum

def possible_moves(temp, v, visited)
possible_moves=[]
    b= index(temp, 0)
    direction=[]
    if b[0]<2:
    direction.append("d")
        if b[0]>0:
        direction.append("u")
if b[1]<2:
    direction.append('r')
```


Sangeev

Sanjeev Kumar Singh
IBM18CS093

```
if b[1] > 0:
    direction.append('l')
for i in direction:
move = gen(temp, i, b)
    if move not in visited:
    possible_mvs.append(move)
    return possible_moves,

def Solve(visited, limit, src)
if src == Goal:
    print("Req moves to reach Goal state" + str(limit))
    return True
    if limit > 3
        return false
    min = math.inf
    visited.append(src)
    possible_action = possible_moves(src, visited)

new_moves()
for action in possible_action:
man_dist = manhattan(action)
    if action not in visited and man_dist < min:
        min = min_dist
    new_move = action
print("move:", limit+1)
print_matrix(new_move)
    if Solve(visited, limit+1, new_move):
        return True

else
return false.
```

Sanjeev