13/11/20 AI Lab Test 1

Sanjeev Kumar Singh
1BM18CS093
B3

~~Program1~~

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 :
```

Sanjeev

1BM18CS093

Sanjeev Kumar Singh
~~1BM18CS093~~
B3

```
Visited _states. add (tuple (state))
  print- grid (State)
    if State == target :
    print ("success")
  return
moves += [move for move in possible - moves states, 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~~ 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 ~~t~~ 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 f. val = 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. fval = Self . f (i, goal )
    Self . open. append (i)
    Self. closed. append (Curr)
  del self. open[0]
Self. open . Sort ( Key = lambda x : x .fval , reverse False)
  puz = puzzle(3)
    puz. process ( )
```

Sanjeev