**Experiment No.5 (AI)**

**Input:**

from copy import deepcopy

from colorama import Fore, Back, Style

#direction matrix

DIRECTIONS = {"U": [-1, 0], "D": [1, 0], "L": [0, -1], "R": [0, 1]}

#target matrix

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

# unicode for draw puzzle in command promt or terminal

left\_down\_angle = '\u2514'

right\_down\_angle = '\u2518'

right\_up\_angle = '\u2510'

left\_up\_angle = '\u250C'

middle\_junction = '\u253C'

top\_junction = '\u252C'

bottom\_junction = '\u2534'

right\_junction = '\u2524'

left\_junction = '\u251C'

#bar color

bar = Style.BRIGHT + Fore.CYAN + '\u2502' + Fore.RESET + Style.RESET\_ALL

dash = '\u2500'

#Line draw code

first\_line = Style.BRIGHT + Fore.CYAN + left\_up\_angle + dash + dash + dash + top\_junction + dash + dash + dash + top\_junction + dash + dash + dash + right\_up\_angle + Fore.RESET + Style.RESET\_ALL

middle\_line = Style.BRIGHT + Fore.CYAN + left\_junction + dash + dash + dash + middle\_junction + dash + dash + dash + middle\_junction + dash + dash + dash + right\_junction + Fore.RESET + Style.RESET\_ALL

last\_line = Style.BRIGHT + Fore.CYAN + left\_down\_angle + dash + dash + dash + bottom\_junction + dash + dash + dash + bottom\_junction + dash + dash + dash + right\_down\_angle + Fore.RESET + Style.RESET\_ALL

#puzzle print function

def print\_puzzle(array):

print(first\_line)

for a in range(len(array)):

for i in array[a]:

if i == 0:

print(bar, Back.RED + ' ' + Back.RESET, end=' ')

else:

print(bar, i, end=' ')

print(bar)

if a == 2:

print(last\_line)

else:

print(middle\_line)

#it is the node which store each state of puzzle

class Node:

def \_\_init\_\_(self, current\_node, previous\_node, g, h, dir):

self.current\_node = current\_node

self.previous\_node = previous\_node

self.g = g

self.h = h

self.dir = dir

def f(self):

return self.g + self.h

def get\_pos(current\_state, element):

for row in range(len(current\_state)):

if element in current\_state[row]:

return (row, current\_state[row].index(element))

#it is a distance calculation algo

def euclidianCost(current\_state):

cost = 0

for row in range(len(current\_state)):

for col in range(len(current\_state[0])):

pos = get\_pos(END, current\_state[row][col])

cost += abs(row - pos[0]) + abs(col - pos[1])

return cost

#get adjucent Nodes

def getAdjNode(node):

listNode = []

emptyPos = get\_pos(node.current\_node, 0)

for dir in DIRECTIONS.keys():

newPos = (emptyPos[0] + DIRECTIONS[dir][0], emptyPos[1] + DIRECTIONS[dir][1])

if 0 <= newPos[0] < len(node.current\_node) and 0 <= newPos[1] < len(node.current\_node[0]):

newState = deepcopy(node.current\_node)

newState[emptyPos[0]][emptyPos[1]] = node.current\_node[newPos[0]][newPos[1]]

newState[newPos[0]][newPos[1]] = 0

# listNode += [Node(newState, node.current\_node, node.g + 1, euclidianCost(newState), dir)]

listNode.append(Node(newState, node.current\_node, node.g + 1, euclidianCost(newState), dir))

return listNode

#get the best node available among nodes

def getBestNode(openSet):

firstIter = True

for node in openSet.values():

if firstIter or node.f() < bestF:

firstIter = False

bestNode = node

bestF = bestNode.f()

return bestNode

#this functionn create the smallest path

def buildPath(closedSet):

node = closedSet[str(END)]

branch = list()

while node.dir:

branch.append({

'dir': node.dir,

'node': node.current\_node

})

node = closedSet[str(node.previous\_node)]

branch.append({

'dir': '',

'node': node.current\_node

})

branch.reverse()

return branch

#main function of node

def main(puzzle):

open\_set = {str(puzzle): Node(puzzle, puzzle, 0, euclidianCost(puzzle), "")}

closed\_set = {}

while True:

test\_node = getBestNode(open\_set)

closed\_set[str(test\_node.current\_node)] = test\_node

if test\_node.current\_node == END:

return buildPath(closed\_set)

adj\_node = getAdjNode(test\_node)

for node in adj\_node:

if str(node.current\_node) in closed\_set.keys() or str(node.current\_node) in open\_set.keys() and open\_set[

str(node.current\_node)].f() < node.f():

continue

open\_set[str(node.current\_node)] = node

del open\_set[str(test\_node.current\_node)]

if \_\_name\_\_ == '\_\_main\_\_':

#it is start matrix

br = main([[6, 2, 8],

[4, 7, 1],

[0, 3, 5]])

print('total steps : ', len(br) - 1)

print()

print(dash + dash + right\_junction, "INPUT", left\_junction + dash + dash)

for b in br:

if b['dir'] != '':

letter = ''

if b['dir'] == 'U':

letter = 'UP'

elif b['dir'] == 'R':

letter = "RIGHT"

elif b['dir'] == 'L':

letter = 'LEFT'

elif b['dir'] == 'D':

letter = 'DOWN'

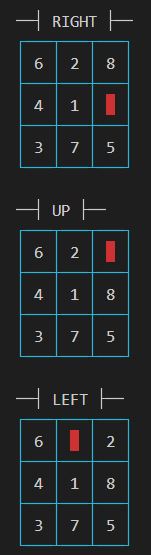
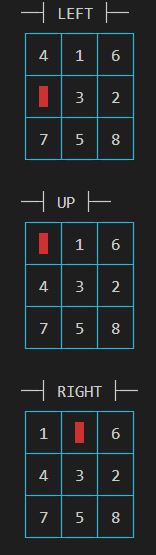
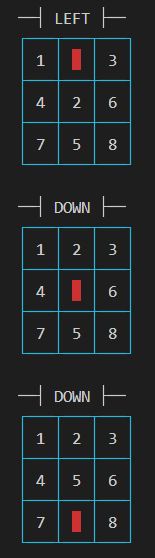
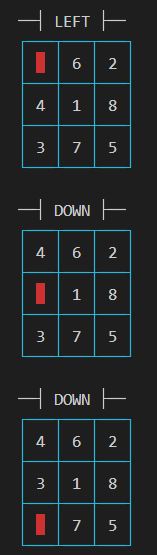
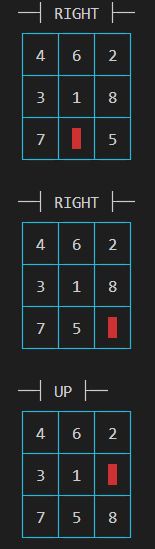
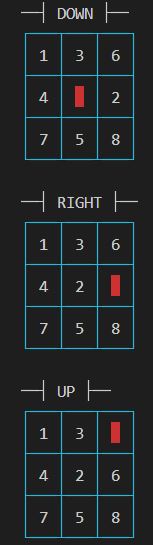
print(dash + dash + right\_junction, letter, left\_junction + dash + dash)

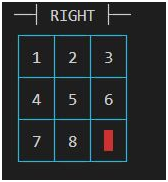
print\_puzzle(b['node'])

print()

print(dash + dash + right\_junction, 'ABOVE IS THE OUTPUT', left\_junction + dash + dash)

**OUTPUT:-**

****

****