In [2]:

```
def accept(n):
    """ Accepts the puzzle from the user """
    puz = []
    for i in range(n):
        puz.append([val for val in input().split()])
    return puz

def print_board(board,n):
    for i in range(n):
        print()
        for j in range(n):
            print(board[i][j],end=' ')

#Find the position of blank space
def find_space(Current,n):
    for blank_row_pos in range(n):
        for blank_col_pos in range(n):
            if Current[blank_row_pos][blank_col_pos]=='_':
                return blank_row_pos,blank_col_pos

#Copy the current node to new node for shuffling the blank space and create a new conf
def copy_current(Current):
    temp=[]
    for i in range(len(Current)):
        row=[]
        for val in Current[i]:
            row.append(val)
        temp.append(row)
    return(temp)

#Move the blank space in given direction, if out of range return None
def shuffle(Current,brow_pos,bcol_pos,move_x,move_y):
    if move_x >= 0 and move_x < len(Current) and move_y >= 0 and move_y < len(Current)
        temp=[]
        temp=copy_current(Current)
        change=temp[move_x][move_y]
        temp[move_x][move_y]=temp[brow_pos][bcol_pos]
        temp[brow_pos][bcol_pos]=change
        return temp
    else:
        return None

#Function to calculate g_score: the number of nodes traversed from a start node to get
def g_score(Node):
    return Node[1] #Node=[Board,level,fscore]

#Function to calculate h_score: the number of misplaced tiles by comparing the current
def h_score(Current,Goal,n):
    hscore=0
    for i in range(n):
        for j in range(n):
            if (Current[i][j] != Goal[i][j]) and (Current[i][j]!='_'):
                hscore +=1

    return hscore

#Function to calculate f_Score= g_score + h_Score
def f_score(Node,Goal,n):
```

```
    Current=Node[0]
    return g_score(Node) + h_score(Current,Goal,n)

#Generate the child nodes by moving the blank in any four direction (up,down,left,righ
def move_gen(Node,Goal,n):
    Current=Node[0]
    level=Node[1]
    fscore=0
    row,col=find_space(Current,n)
    move_positions=[[row,col-1],[row,col+1],[row-1,col],[row+1,col]] #left,right,up,do
    children=[] #List of child nodes of current node
    for move in move_positions:
        child=shuffle(Current,row,col,move[0],move[1])
        if child is not None:
            cNode=[child,0,0] #Dummy node for calculating f_Score
            fscore=f_score(cNode,Goal,n)
            Node=[child,level+1,fscore]
            children.append(Node)
    print("\n\n The Children ::",children)
    return children

#Function goal_test to see the goal configuration is reached
def goal_test(Current,Goal,n):
    if h_score(Current,Goal,n) == 0:
        return True
    else:
        return False

#Function to Sort OPEN based on f_score
def sort(L):
    L.sort(key = lambda x: x[2],reverse=False)
    return L

#Function for starting the Game
def play_game(Start, Goal, n):
    #when game starts
    fscore=0 #fscore initialized to zero
    gscore=0 #gscore initialized to zero

    level=0 #the start configuration is root node s at level-0 of the state space tree
    '''
    print_board(Start,n);
    print_board(Goal,n)
    print("\n\nI AM HERE !!!\n\n")
    '''
    Node=[Start,level,fscore]
    fscore=f_score(Node,Goal,n)
    #Every Node is [board configuration ,level,gscore]
    Node = [Start,level,fscore] # current node is Start node
    print("\nThe Node is=\n",Node)
    OPEN = [] #OPEN list as frontier
    CLOSED = [] #CLOSED as explored
    OPEN.append(Node)
    levelcount=0
    #Explored the current node to reach to the Goal configuration
    while True:
        N=OPEN[0] #first node of open
        del OPEN[0] # delete first node of open
        Current=N[0] #Extract board configuration
        print("\n\n The current configuration is ::",Current)
```

```
        CLOSED.append(N)
        #if goal configuration is reached terminate
        if goal_test(Current,Goal,n) == True:
            print("\nGoal reached!!")
            print("CLOSED=",CLOSED)
            break
        CHILD=move_gen(N,Goal,n)
        #print("\n\n The CHILD is ::",CHILD)
        OPEN=[]
        for child in CHILD:
            OPEN.append(child)
        #sort the OPEN list based on fscore value of each node
        sort(OPEN)
    #print("\n\n The OPEN is ::",OPEN)

#Drive Code
n=int(input("Enter the board size:"))
print("\nEnter Start Configuration of board")
Start=accept(n)

print("\nEnter Goal Configuration of board")
Goal=accept(n)
play_game(Start, Goal, n)
```

```
Enter the board size:3

Enter Start Configuration of board
1 2 3
_ 4 6
7 5 8

Enter Goal Configuration of board
1 2 3
4 5 6
7 8 _

The Node is=
 [[['1', '2', '3'], ['_', '4', '6'], ['7', '5', '8']], 0, 3]


 The current configuration is :: [['1', '2', '3'], ['_', '4', '6'], ['7', '5', '8']]


 The Children :: [[[['1', '2', '3'], ['4', '_', '6'], ['7', '5', '8']], 1, 2],
[[['_', '2', '3'], ['1', '4', '6'], ['7', '5', '8']], 1, 4], [[['1', '2', '3'], ['7',
'4', '6'], ['_', '5', '8']], 1, 4]]


 The current configuration is :: [['1', '2', '3'], ['4', '_', '6'], ['7', '5', '8']]


 The Children :: [[[['1', '2', '3'], ['_', '4', '6'], ['7', '5', '8']], 2, 3],
[[['1', '2', '3'], ['4', '6', '_'], ['7', '5', '8']], 2, 3], [[['1', '_', '3'], ['4',
'2', '6'], ['7', '5', '8']], 2, 3], [[['1', '2', '3'], ['4', '5', '6'], ['7', '_',
'8']], 2, 1]]


 The current configuration is :: [['1', '2', '3'], ['4', '5', '6'], ['7', '_', '8']]


 The Children :: [[[['1', '2', '3'], ['4', '5', '6'], ['_', '7', '8']], 3, 2],
[[['1', '2', '3'], ['4', '5', '6'], ['7', '8', '_']], 3, 0], [[['1', '2', '3'], ['4',
'_', '6'], ['7', '5', '8']], 3, 2]]


 The current configuration is :: [['1', '2', '3'], ['4', '5', '6'], ['7', '8', '_']]

Goal reached!!
CLOSED= [[[['1', '2', '3'], ['_', '4', '6'], ['7', '5', '8']], 0, 3], [[['1', '2',
'3'], ['4', '_', '6'], ['7', '5', '8']], 1, 2], [[['1', '2', '3'], ['4', '5', '6'],
['7', '_', '8']], 2, 1], [[['1', '2', '3'], ['4', '5', '6'], ['7', '8', '_']], 3, 0]]
```

In [ ]: