PACKAGE TITLE : MULTI USER CHESS GAME

SERVER :

import socket  
import pickle  
#import re  
  
s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
host = socket.gethostname()  
  
port = 12345  
  
s.bind((host, port))  
s.listen(5)  
print("conection initiated")  
client, addr = s.accept() # white client  
print("conection ok")  
client2, addr2 = s.accept() # black client  
  
  
def winCheck(chessBoard):  
 if not (any('wk' in sublist for sublist in chessBoard)):  
 return 2  
 elif not (any('bk' in sublist for sublist in chessBoard)):  
 return 1  
 else:  
 return 0  
  
  
while True:  
 msg = client.recv(4096) # recieve Board from WHITE  
  
 # check if white wins  
 msg1 = pickle.loads(msg)  
 print(winCheck(msg1))  
  
 client2.send(msg) # send the board to black(client2)  
 client2.send(pickle.dumps(winCheck(msg1)))  
  
 msg = client2.recv(4096) # recieve the board from black (client 2)  
 msg1 = pickle.loads(msg)  
 # check if black wins  
  
 client.send(msg) # send it to White (client 1)  
 client.send(pickle.dumps(winCheck(msg1)))

CLIENT1:

# WHITE  
import socket  
import pickle  
  
host = '10.1.82.36'  
port = 12345  
s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
s.connect((host, port))  
print('connected')  
  
chessBoard = [['--'] \* 8 for i in range(8)]  
  
temp = chessBoard  
  
chess\_map\_from\_alpha\_to\_index = {  
 "a": 0,  
 "b": 1,  
 "c": 2,  
 "d": 3,  
 "e": 4,  
 "f": 5,  
 "g": 6,  
 "h": 7  
}  
  
chess\_map\_from\_index\_to\_alpha = {  
 0: "a",  
 1: "b",  
 2: "c",  
 3: "d",  
 4: "e",  
 5: "f",  
 6: "g",  
 7: "h"  
}  
  
  
def getRookMoves(pos, chessBoard):  
 column, row = list(pos.strip().lower())  
 row = int(row) - 1  
 col1 = column  
 column = chess\_map\_from\_alpha\_to\_index[column]  
 i, j = row, column  
 solutionMoves = []  
  
 # top right  
 while 1:  
 try:  
 temp = chessBoard[i + 1][j]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i + 1, j])  
 elif (color == 'b'):  
 solutionMoves.append([i + 1, j])  
 break  
 else:  
 break  
 i = i + 1  
  
  
 except:  
 break  
  
 # top left  
 i, j = row, column  
 # print('top left')  
 while 1:  
  
 try:  
 temp = chessBoard[i - 1][j]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i - 1, j])  
 elif (color == 'b'):  
 solutionMoves.append([i - 1, j])  
 break  
 else:  
 break  
 i = i - 1  
  
  
 except:  
 break  
 # bottom left  
 i, j = row, column  
 # print('top left')  
 while 1:  
 try:  
 temp = chessBoard[i][j - 1]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i, j - 1])  
 elif (color == 'b'):  
 solutionMoves.append([i, j - 1])  
 break  
 else:  
 break  
  
 j = j - 1  
  
 except:  
 break  
  
 # bottom right  
 i, j = row, column  
 while 1:  
 try:  
 temp = chessBoard[i][j + 1]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i, j + 1])  
 elif (color == 'b'):  
 solutionMoves.append([i, j + 1])  
 break  
 else:  
 break  
  
 j = j + 1  
  
 except:  
 break  
  
 temp = [i for i in solutionMoves if i[0] >= 0 and i[1] >= 0]  
 allPossibleMoves = ["".join([chess\_map\_from\_index\_to\_alpha[i[1]], str(i[0] + 1)]) for i in temp]  
 allPossibleMoves.sort()  
 return allPossibleMoves  
  
  
def getKingMoves(pos, chessBoard):  
 column, row = list(pos.strip().lower())  
 row = int(row) - 1  
 column = chess\_map\_from\_alpha\_to\_index[column]  
 i, j = row, column  
 solutionMoves = []  
 try:  
 temp = chessBoard[i + 1][j]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i + 1, j])  
 elif (color == 'b'):  
 solutionMoves.append([i + 1, j])  
 else:  
 se = 20  
  
  
 except:  
 pass  
 try:  
 temp = chessBoard[i + 1][j + 1]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i + 1, j + 1])  
 elif (color == 'b'):  
 solutionMoves.append([i + 1, j + 1])  
  
  
 except:  
 pass  
 try:  
 temp = chessBoard[i + 1][j - 1]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i + 1, j - 1])  
 elif (color == 'b'):  
 solutionMoves.append([i + 1, j - 1])  
  
  
 except:  
 pass  
 try:  
 temp = chessBoard[i - 1][j]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i - 1, j])  
 elif (color == 'b'):  
 solutionMoves.append([i - 1, j])  
  
  
 except:  
 pass  
 try:  
 temp = chessBoard[i][j + 1]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i, j + 1])  
 elif (color == 'b'):  
 solutionMoves.append([i, j + 1])  
  
  
 except:  
 pass  
 try:  
 temp = chessBoard[i][j - 1]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i, j - 1])  
 elif (color == 'b'):  
 solutionMoves.append([i, j - 1])  
  
  
 except:  
 pass  
 try:  
 temp = chessBoard[i - 1][j - 1]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i - 1, j - 1])  
 elif (color == 'b'):  
 solutionMoves.append([i - 1, j - 1])  
  
  
 except:  
 pass  
 try:  
 temp = chessBoard[i - 1][j + 1]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i - 1, j + 1])  
 elif (color == 'b'):  
 solutionMoves.append([i - 1, j + 1])  
  
  
 except:  
 pass  
  
 # Filter all negative values  
 temp = [i for i in solutionMoves if i[0] >= 0 and i[1] >= 0]  
 allPossibleMoves = ["".join([chess\_map\_from\_index\_to\_alpha[i[1]], str(i[0] + 1)]) for i in temp]  
 allPossibleMoves.sort()  
 return allPossibleMoves  
  
  
def getBishopMoves(pos, chessBoard):  
 column, row = list(pos.strip().lower())  
 row = int(row) - 1  
 col1 = column  
 column = chess\_map\_from\_alpha\_to\_index[column]  
 i, j = row, column  
 solutionMoves = []  
  
 # top right  
 while 1:  
 try:  
 temp = chessBoard[i + 1][j + 1]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i + 1, j + 1])  
 elif (color == 'b'):  
 solutionMoves.append([i + 1, j + 1])  
 break  
 else:  
 break  
 i = i + 1  
 j = j + 1  
  
 except:  
 break  
  
 # top left  
 i, j = row, column  
 # print('top left')  
 while 1:  
  
 try:  
 temp = chessBoard[i - 1][j + 1]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i - 1, j + 1])  
 elif (color == 'b'):  
 solutionMoves.append([i - 1, j + 1])  
 break  
 else:  
 break  
 i = i - 1  
 j = j + 1  
  
 except:  
 break  
 # bottom left  
 i, j = row, column  
 # print('top left')  
 while 1:  
 try:  
 temp = chessBoard[i + 1][j - 1]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i + 1, j - 1])  
 elif (color == 'b'):  
 solutionMoves.append([i + 1, j - 1])  
 break  
 else:  
 break  
 i = i + 1  
 j = j - 1  
  
 except:  
 break  
  
 # bottom right  
 i, j = row, column  
 while 1:  
 try:  
 temp = chessBoard[i - 1][j - 1]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i - 1, j - 1])  
 elif (color == 'b'):  
 solutionMoves.append([i - 1, j - 1])  
 break  
 else:  
 break  
 i = i - 1  
 j = j - 1  
  
 except:  
 break  
  
 temp = [i for i in solutionMoves if i[0] >= 0 and i[1] >= 0]  
 allPossibleMoves = ["".join([chess\_map\_from\_index\_to\_alpha[i[1]], str(i[0] + 1)]) for i in temp]  
 allPossibleMoves.sort()  
 return allPossibleMoves  
  
  
def getQueenMoves(pos, chessBoard):  
 column, row = list(pos.strip().lower())  
 row = int(row) - 1  
 col1 = column  
 column = chess\_map\_from\_alpha\_to\_index[column]  
 i, j = row, column  
 solutionMoves = []  
  
 # BISHOP LOGIC  
  
 # top right  
 while 1:  
 try:  
 temp = chessBoard[i + 1][j + 1]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i + 1, j + 1])  
 elif (color == 'b'):  
 solutionMoves.append([i + 1, j + 1])  
 break  
 else:  
 break  
 i = i + 1  
 j = j + 1  
  
 except:  
 break  
  
 # top left  
 i, j = row, column  
 # print('top left')  
 while 1:  
  
 try:  
 temp = chessBoard[i - 1][j + 1]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i - 1, j + 1])  
 elif (color == 'b'):  
 solutionMoves.append([i - 1, j + 1])  
 break  
 else:  
 break  
 i = i - 1  
 j = j + 1  
  
 except:  
 break  
 # bottom left  
 i, j = row, column  
 # print('top left')  
 while 1:  
 try:  
 temp = chessBoard[i + 1][j - 1]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i + 1, j - 1])  
 elif (color == 'b'):  
 solutionMoves.append([i + 1, j - 1])  
 break  
 else:  
 break  
 i = i + 1  
 j = j - 1  
  
 except:  
 break  
  
 # bottom right  
 i, j = row, column  
 while 1:  
 try:  
 temp = chessBoard[i - 1][j - 1]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i - 1, j - 1])  
 elif (color == 'b'):  
 solutionMoves.append([i - 1, j - 1])  
 break  
 else:  
 break  
 i = i - 1  
 j = j - 1  
  
 except:  
 break  
  
 # ROOK LOGIC  
  
 # top right  
 i, j = row, column  
 while 1:  
 try:  
 temp = chessBoard[i + 1][j]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i + 1, j])  
 elif (color == 'b'):  
 solutionMoves.append([i + 1, j])  
 break  
 else:  
 break  
 i = i + 1  
  
  
 except:  
 break  
  
 # top left  
 i, j = row, column  
 # print('top left')  
 while 1:  
  
 try:  
 temp = chessBoard[i - 1][j]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i - 1, j])  
 elif (color == 'b'):  
 solutionMoves.append([i - 1, j])  
 break  
 else:  
 break  
 i = i - 1  
  
  
 except:  
 break  
 # bottom left  
 i, j = row, column  
 # print('top left')  
 while 1:  
 try:  
 temp = chessBoard[i][j - 1]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i, j - 1])  
 elif (color == 'b'):  
 solutionMoves.append([i, j - 1])  
 break  
 else:  
 break  
  
 j = j - 1  
  
 except:  
 break  
  
 # bottom right  
 i, j = row, column  
 while 1:  
 try:  
 temp = chessBoard[i][j + 1]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i, j + 1])  
 elif (color == 'b'):  
 solutionMoves.append([i, j + 1])  
 break  
 else:  
 break  
  
 j = j + 1  
  
 except:  
 break  
  
 temp = [i for i in solutionMoves if i[0] >= 0 and i[1] >= 0]  
 allPossibleMoves = ["".join([chess\_map\_from\_index\_to\_alpha[i[1]], str(i[0] + 1)]) for i in temp]  
 allPossibleMoves.sort()  
 return allPossibleMoves  
  
  
def getKnightMoves(pos, chessBoard):  
 column, row = list(pos.strip().lower())  
 row = int(row) - 1  
 column = chess\_map\_from\_alpha\_to\_index[column]  
 i, j = row, column  
 solutionMoves = []  
 try:  
 temp = chessBoard[i + 1][j - 2]  
 solutionMoves.append([i + 1, j - 2])  
 except:  
 pass  
 try:  
 temp = chessBoard[i + 2][j - 1]  
 solutionMoves.append([i + 2, j - 1])  
 except:  
 pass  
 try:  
 temp = chessBoard[i + 2][j + 1]  
 solutionMoves.append([i + 2, j + 1])  
 except:  
 pass  
 try:  
 temp = chessBoard[i + 1][j + 2]  
 solutionMoves.append([i + 1, j + 2])  
 except:  
 pass  
 try:  
 temp = chessBoard[i - 1][j + 2]  
 solutionMoves.append([i - 1, j + 2])  
 except:  
 pass  
 try:  
 temp = chessBoard[i - 2][j + 1]  
 solutionMoves.append([i - 2, j + 1])  
 except:  
 pass  
 try:  
 temp = chessBoard[i - 2][j - 1]  
 solutionMoves.append([i - 2, j - 1])  
 except:  
 pass  
 try:  
 temp = chessBoard[i - 1][j - 2]  
 solutionMoves.append([i - 1, j - 2])  
 except:  
 pass  
  
 # Filter all negative values  
 temp = [i for i in solutionMoves if i[0] >= 0 and i[1] >= 0]  
 allPossibleMoves = ["".join([chess\_map\_from\_index\_to\_alpha[i[1]], str(i[0] + 1)]) for i in temp]  
 allPossibleMoves.sort()  
 return allPossibleMoves  
  
  
def assign(pos, piece):  
 column, row = list(pos.strip().lower())  
 row = int(row) - 1  
 column = chess\_map\_from\_alpha\_to\_index[column]  
 i, j = row, column  
  
 chessBoard[i][j] = piece  
  
#find what piece is in a particular pos  
def findpiece(pos):  
 column, row = list(pos.strip().lower())  
 row = int(row) - 1  
 column = chess\_map\_from\_alpha\_to\_index[column]  
 i, j = row, column  
  
 return chessBoard[i][j]  
  
#if a user chooses white color the other user shud choose black  
#validating the color  
def validinit(piece):  
 column, row = list(piece.strip().lower())  
 if (column == 'w'):  
 return True  
 else:  
 print('Select white pieces only')  
 return False  
  
  
def move(piece, pos):  
 if (piece == 'r'):  
 while 1:  
 x = getRookMoves(pos, chessBoard)  
  
 if (len(x) == 0):  
 print('No possible moves for Rook')  
 return False  
 print(x)  
 finald = input('Enter the destination location from the displayed list')  
 if (finald in x):  
 return finald  
 else:  
 print('Invalid destination')  
 continue  
 elif (piece == 'h'):  
 while 1:  
 x = getKnightMoves(pos, chessBoard)  
 print(x)  
 finald = input('Enter the destination location from the displayed list')  
 if (finald in x):  
 return finald  
 else:  
 print('Invalid destination')  
 continue  
 elif (piece == 'b'):  
 while 1:  
 x = getBishopMoves(pos, chessBoard)  
  
 if (len(x) == 0):  
 print('No possible moves for Bishop')  
 return False  
 print(x)  
 finald = input('Enter the destination location from the displayed list')  
 if (finald in x):  
 return finald  
 else:  
 print('Invalid destination')  
 continue  
 elif (piece == 'q'):  
 while 1:  
 x = getQueenMoves(pos, chessBoard)  
  
 if (len(x) == 0):  
 print('No possible moves for Queen')  
 return False  
 print(x)  
 finald = input('Enter the destination location from the displayed list')  
 if (finald in x):  
 return finald  
 else:  
 print('Invalid destination')  
 continue  
 elif (piece == 'k'):  
 while 1:  
 x = getKingMoves(pos, chessBoard)  
 print(x)  
 finald = input('Enter the destination location from the displayed list')  
 if (finald in x):  
 return finald  
 else:  
 print('Invalid destination')  
 continue  
  
 else:  
 finald = input('Enter the destination')  
 return finald  
  
  
def finalassign(piece, initial, final):  
 fpiece = findpiece(final)  
 color, fpiece = list(fpiece.strip().lower())  
 if (color == 'w'):  
 return False  
 else:  
 assign(final, piece)  
 assign(initial, '--')  
 return True  
  
  
# assign('e2','k')  
  
# assigning initlal board  
  
# white pieces 2nd row  
# syntax : assign(pos,piece)  
assign('a1', 'wr')  
assign('b1', 'wh')  
assign('c1', 'wb')  
assign('d1', 'wq')  
assign('e1', 'wk')  
assign('f1', 'wb')  
assign('g1', 'wh')  
assign('h1', 'wr')  
  
# white peices 1st row  
assign('a2', 'wp')  
assign('b2', 'wp')  
assign('c2', 'wp')  
assign('d2', 'wp')  
assign('e2', 'wp')  
assign('f2', 'wp')  
assign('g2', 'wp')  
assign('h2', 'wp')  
  
# black pieces 2d row  
assign('a8', 'br')  
assign('b8', 'bh')  
assign('c8', 'bb')  
assign('d8', 'bq')  
assign('e8', 'bk')  
assign('f8', 'bb')  
assign('g8', 'bh')  
assign('h8', 'br')  
  
# black pieces 1st row  
assign('a7', 'bp')  
assign('b7', 'bp')  
assign('c7', 'bp')  
assign('d7', 'bp')  
assign('e7', 'bp')  
assign('f7', 'bp')  
assign('g7', 'bp')  
assign('h7', 'bp')  
  
# print Board  
# print(chessBoard)  
chessBoard = temp  
  
  
def printBoard():  
 print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')  
 for i in reversed(chessBoard):  
 print(i)  
 print('\n')  
 print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')  
  
  
# assign('d2','--')  
# assign('d3','wq')  
# assign('d2','--')  
# assign('d4','wr')  
  
printBoard()  
  
while True:  
 # White player turn code goes here  
 initialp = input('Enter the location of the piece to move')  
 initial = findpiece(initialp)  
 # print(initial)  
 if (validinit(initial)):  
 # continue  
 color, piece = list(initial.strip().lower())  
 finald = move(piece, initialp)  
 if (finald == False):  
 continue  
 if (finalassign(initial, initialp, finald)):  
 finalassign(initial, initialp, finald)  
 # conitinue  
 printBoard()  
  
 msg = pickle.dumps(chessBoard)  
 s.send(msg) # send board to server  
  
 if not (any('bk' in sublist for sublist in chessBoard)):  
 print('You won the game')  
 break  
 else:  
 print('You cant cut your own piece - INVALID')  
 continue  
  
 else:  
 print('Invalid input ENter again')  
 continue  
  
 # break  
  
 msg = s.recv(4096) # recieve Board from server  
 status = s.recv(4096) # recieve status from server  
 msg = pickle.loads(msg)  
 chessBoard = msg  
 printBoard()  
  
 status = pickle.loads(status)  
 if (int(status) == 2):  
 print('You lost the game, Your King has been defeated')  
 break  
 # this means black has completed the move  
 # so do nothing loop over after updating chess board

CLIENT2:

# BLACK  
import socket  
import pickle  
  
host = '10.1.82.36'  
port = 12345  
s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
s.connect((host, port))  
  
chessBoard = [[1] \* 8 for i in range(8)]  
  
chess\_map\_from\_alpha\_to\_index = {  
 "a": 0,  
 "b": 1,  
 "c": 2,  
 "d": 3,  
 "e": 4,  
 "f": 5,  
 "g": 6,  
 "h": 7  
}  
  
chess\_map\_from\_index\_to\_alpha = {  
 0: "a",  
 1: "b",  
 2: "c",  
 3: "d",  
 4: "e",  
 5: "f",  
 6: "g",  
 7: "h"  
}  
  
  
def findpiece(pos):  
 column, row = list(pos.strip().lower())  
 row = int(row) - 1  
 column = chess\_map\_from\_alpha\_to\_index[column]  
 i, j = row, column  
 return chessBoard[i][j]  
  
  
def getRookMoves(pos, chessBoard):  
 column, row = list(pos.strip().lower())  
 row = int(row) - 1  
 col1 = column  
 column = chess\_map\_from\_alpha\_to\_index[column]  
 i, j = row, column  
 solutionMoves = []  
  
 # top right  
 while 1:  
 try:  
 temp = chessBoard[i + 1][j]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i + 1, j])  
 elif (color == 'w'):  
 solutionMoves.append([i + 1, j])  
 break  
 else:  
 break  
 i = i + 1  
  
  
 except:  
 break  
  
 # top left  
 i, j = row, column  
 # print('top left')  
 while 1:  
  
 try:  
 temp = chessBoard[i - 1][j]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i - 1, j])  
 elif (color == 'w'):  
 solutionMoves.append([i - 1, j])  
 break  
 else:  
 break  
 i = i - 1  
  
  
 except:  
 break  
 # bottom left  
 i, j = row, column  
 # print('top left')  
 while 1:  
 try:  
 temp = chessBoard[i][j - 1]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i, j - 1])  
 elif (color == 'w'):  
 solutionMoves.append([i, j - 1])  
 break  
 else:  
 break  
  
 j = j - 1  
  
 except:  
 break  
  
 # bottom right  
 i, j = row, column  
 while 1:  
 try:  
 temp = chessBoard[i][j + 1]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i, j + 1])  
 elif (color == 'w'):  
 solutionMoves.append([i, j + 1])  
 break  
 else:  
 break  
  
 j = j + 1  
  
 except:  
 break  
  
 temp = [i for i in solutionMoves if i[0] >= 0 and i[1] >= 0]  
 allPossibleMoves = ["".join([chess\_map\_from\_index\_to\_alpha[i[1]], str(i[0] + 1)]) for i in temp]  
 allPossibleMoves.sort()  
 return allPossibleMoves  
  
  
def getQueenMoves(pos, chessBoard):  
 column, row = list(pos.strip().lower())  
 row = int(row) - 1  
 col1 = column  
 column = chess\_map\_from\_alpha\_to\_index[column]  
 i, j = row, column  
 solutionMoves = []  
  
 # BISHOP LOGIC  
  
 # top right  
 while 1:  
 try:  
 temp = chessBoard[i + 1][j + 1]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i + 1, j + 1])  
 elif (color == 'w'):  
 solutionMoves.append([i + 1, j + 1])  
 break  
 else:  
 break  
 i = i + 1  
 j = j + 1  
  
 except:  
 break  
  
 # top left  
 i, j = row, column  
 # print('top left')  
 while 1:  
  
 try:  
 temp = chessBoard[i - 1][j + 1]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i - 1, j + 1])  
 elif (color == 'w'):  
 solutionMoves.append([i - 1, j + 1])  
 break  
 else:  
 break  
 i = i - 1  
 j = j + 1  
  
 except:  
 break  
 # bottom left  
 i, j = row, column  
 # print('top left')  
 while 1:  
 try:  
 temp = chessBoard[i + 1][j - 1]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i + 1, j - 1])  
 elif (color == 'w'):  
 solutionMoves.append([i + 1, j - 1])  
 break  
 else:  
 break  
 i = i + 1  
 j = j - 1  
  
 except:  
 break  
  
 # bottom right  
 i, j = row, column  
 while 1:  
 try:  
 temp = chessBoard[i - 1][j - 1]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i - 1, j - 1])  
 elif (color == 'w'):  
 solutionMoves.append([i - 1, j - 1])  
 break  
 else:  
 break  
 i = i - 1  
 j = j - 1  
  
 except:  
 break  
  
 # ROOK LOGIC  
  
 # top right  
 i, j = row, column  
 while 1:  
 try:  
 temp = chessBoard[i + 1][j]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i + 1, j])  
 elif (color == 'w'):  
 solutionMoves.append([i + 1, j])  
 break  
 else:  
 break  
 i = i + 1  
  
  
 except:  
 break  
  
 # top left  
 i, j = row, column  
 # print('top left')  
 while 1:  
  
 try:  
 temp = chessBoard[i - 1][j]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i - 1, j])  
 elif (color == 'w'):  
 solutionMoves.append([i - 1, j])  
 break  
 else:  
 break  
 i = i - 1  
  
  
 except:  
 break  
 # bottom left  
 i, j = row, column  
 # print('top left')  
 while 1:  
 try:  
 temp = chessBoard[i][j - 1]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i, j - 1])  
 elif (color == 'w'):  
 solutionMoves.append([i, j - 1])  
 break  
 else:  
 break  
  
 j = j - 1  
  
 except:  
 break  
  
 # bottom right  
 i, j = row, column  
 while 1:  
 try:  
 temp = chessBoard[i][j + 1]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i, j + 1])  
 elif (color == 'w'):  
 solutionMoves.append([i, j + 1])  
 break  
 else:  
 break  
  
 j = j + 1  
  
 except:  
 break  
  
 temp = [i for i in solutionMoves if i[0] >= 0 and i[1] >= 0]  
 allPossibleMoves = ["".join([chess\_map\_from\_index\_to\_alpha[i[1]], str(i[0] + 1)]) for i in temp]  
 allPossibleMoves.sort()  
 return allPossibleMoves  
  
  
def getBishopMoves(pos, chessBoard):  
 column, row = list(pos.strip().lower())  
 row = int(row) - 1  
 col1 = column  
 column = chess\_map\_from\_alpha\_to\_index[column]  
 i, j = row, column  
 solutionMoves = []  
  
 # top right  
 while 1:  
 try:  
 temp = chessBoard[i + 1][j + 1]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i + 1, j + 1])  
 elif (color == 'w'):  
 solutionMoves.append([i + 1, j + 1])  
 break  
 else:  
 break  
 i = i + 1  
 j = j + 1  
  
 except:  
 break  
  
 # top left  
 i, j = row, column  
 # print('top left')  
 while 1:  
  
 try:  
 temp = chessBoard[i - 1][j + 1]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i - 1, j + 1])  
 elif (color == 'w'):  
 solutionMoves.append([i - 1, j + 1])  
 break  
 else:  
 break  
 i = i - 1  
 j = j + 1  
  
 except:  
 break  
 # bottom left  
 i, j = row, column  
 # print('top left')  
 while 1:  
 try:  
 temp = chessBoard[i + 1][j - 1]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i + 1, j - 1])  
 elif (color == 'w'):  
 solutionMoves.append([i + 1, j - 1])  
 break  
 else:  
 break  
 i = i + 1  
 j = j - 1  
  
 except:  
 break  
  
 # bottom right  
 i, j = row, column  
 while 1:  
 try:  
 temp = chessBoard[i - 1][j - 1]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i - 1, j - 1])  
 elif (color == 'w'):  
 solutionMoves.append([i - 1, j - 1])  
 break  
 else:  
 break  
 i = i - 1  
 j = j - 1  
  
 except:  
 break  
  
 temp = [i for i in solutionMoves if i[0] >= 0 and i[1] >= 0]  
 allPossibleMoves = ["".join([chess\_map\_from\_index\_to\_alpha[i[1]], str(i[0] + 1)]) for i in temp]  
 allPossibleMoves.sort()  
 return allPossibleMoves  
  
  
def getKnightMoves(pos, chessBoard):  
 column, row = list(pos.strip().lower())  
 row = int(row) - 1  
 column = chess\_map\_from\_alpha\_to\_index[column]  
 i, j = row, column  
 solutionMoves = []  
 try:  
 temp = chessBoard[i + 1][j - 2]  
 solutionMoves.append([i + 1, j - 2])  
 except:  
 pass  
 try:  
 temp = chessBoard[i + 2][j - 1]  
 solutionMoves.append([i + 2, j - 1])  
 except:  
 pass  
 try:  
 temp = chessBoard[i + 2][j + 1]  
 solutionMoves.append([i + 2, j + 1])  
 except:  
 pass  
 try:  
 temp = chessBoard[i + 1][j + 2]  
 solutionMoves.append([i + 1, j + 2])  
 except:  
 pass  
 try:  
 temp = chessBoard[i - 1][j + 2]  
 solutionMoves.append([i - 1, j + 2])  
 except:  
 pass  
 try:  
 temp = chessBoard[i - 2][j + 1]  
 solutionMoves.append([i - 2, j + 1])  
 except:  
 pass  
 try:  
 temp = chessBoard[i - 2][j - 1]  
 solutionMoves.append([i - 2, j - 1])  
 except:  
 pass  
 try:  
 temp = chessBoard[i - 1][j - 2]  
 solutionMoves.append([i - 1, j - 2])  
 except:  
 pass  
  
 # Filter all negative values  
 temp = [i for i in solutionMoves if i[0] >= 0 and i[1] >= 0]  
 allPossibleMoves = ["".join([chess\_map\_from\_index\_to\_alpha[i[1]], str(i[0] + 1)]) for i in temp]  
 allPossibleMoves.sort()  
 return allPossibleMoves  
  
  
def getKingMoves(pos, chessBoard):  
 column, row = list(pos.strip().lower())  
 row = int(row) - 1  
 column = chess\_map\_from\_alpha\_to\_index[column]  
 i, j = row, column  
 solutionMoves = []  
 try:  
 temp = chessBoard[i + 1][j]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i + 1, j])  
 elif (color == 'b'):  
 solutionMoves.append([i + 1, j])  
 else:  
 se = 20  
  
  
 except:  
 pass  
 try:  
 temp = chessBoard[i + 1][j + 1]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i + 1, j + 1])  
 elif (color == 'b'):  
 solutionMoves.append([i + 1, j + 1])  
  
  
 except:  
 pass  
 try:  
 temp = chessBoard[i + 1][j - 1]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i + 1, j - 1])  
 elif (color == 'b'):  
 solutionMoves.append([i + 1, j - 1])  
  
  
 except:  
 pass  
 try:  
 temp = chessBoard[i - 1][j]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i - 1, j])  
 elif (color == 'b'):  
 solutionMoves.append([i - 1, j])  
  
  
 except:  
 pass  
 try:  
 temp = chessBoard[i][j + 1]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i, j + 1])  
 elif (color == 'b'):  
 solutionMoves.append([i, j + 1])  
  
  
 except:  
 pass  
 try:  
 temp = chessBoard[i][j - 1]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i, j - 1])  
 elif (color == 'b'):  
 solutionMoves.append([i, j - 1])  
  
  
 except:  
 pass  
 try:  
 temp = chessBoard[i - 1][j - 1]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i - 1, j - 1])  
 elif (color == 'b'):  
 solutionMoves.append([i - 1, j - 1])  
  
  
 except:  
 pass  
 try:  
 temp = chessBoard[i - 1][j + 1]  
 color, piece = list(temp.strip().lower())  
 if (color == '-'):  
 solutionMoves.append([i - 1, j + 1])  
 elif (color == 'b'):  
 solutionMoves.append([i - 1, j + 1])  
  
  
 except:  
 pass  
  
 # Filter all negative values  
 temp = [i for i in solutionMoves if i[0] >= 0 and i[1] >= 0]  
 allPossibleMoves = ["".join([chess\_map\_from\_index\_to\_alpha[i[1]], str(i[0] + 1)]) for i in temp]  
 allPossibleMoves.sort()  
 return allPossibleMoves  
  
  
def assign(pos, piece):  
 column, row = list(pos.strip().lower())  
 row = int(row) - 1  
 column = chess\_map\_from\_alpha\_to\_index[column]  
 i, j = row, column  
  
 chessBoard[i][j] = piece  
  
  
def printBoard():  
 print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')  
 for i in reversed(chessBoard):  
 print(i)  
 print('\n')  
 print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')  
  
  
def validinit(piece):  
 column, row = list(piece.strip().lower())  
 if (column == 'b'):  
 return True  
 else:  
 print('Select Black pieces only')  
 return False  
  
  
def move(piece, pos):  
 if (piece == 'r'):  
 while 1:  
 x = getRookMoves(pos, chessBoard)  
  
 if (len(x) == 0):  
 print('No possible moves for Rook')  
 return False  
 print(x)  
 finald = input('Enter the destination location from the displayed list')  
 if (finald in x):  
 return finald  
 else:  
 print('Invalid destination')  
 continue  
 elif (piece == 'h'):  
 while 1:  
 x = getKnightMoves(pos, chessBoard)  
 print(x)  
 finald = input('Enter the destination location from the displayed list')  
 if (finald in x):  
 return finald  
 else:  
 print('Invalid destination')  
 continue  
 elif (piece == 'b'):  
 while 1:  
 x = getBishopMoves(pos, chessBoard)  
  
 if (len(x) == 0):  
 print('No possible moves for Bishop')  
 return False  
 print(x)  
 finald = input('Enter the destination location from the displayed list')  
 if (finald in x):  
 return finald  
 else:  
 print('Invalid destination')  
 continue  
 elif (piece == 'q'):  
 while 1:  
 x = getQueenMoves(pos, chessBoard)  
  
 if (len(x) == 0):  
 print('No possible moves for Queen')  
 return False  
 print(x)  
 finald = input('Enter the destination location from the displayed list')  
 if (finald in x):  
 return finald  
 else:  
 print('Invalid destination')  
 continue  
 elif (piece == 'k'):  
 while 1:  
 x = getKingMoves(pos, chessBoard)  
 print(x)  
 finald = input('Enter the destination location from the displayed list')  
 if (finald in x):  
 return finald  
 else:  
 print('Invalid destination')  
 continue  
 else:  
 finald = input('Enter the destination')  
 return finald  
  
  
def finalassign(piece, initial, final):  
 fpiece = findpiece(final)  
 color, fpiece = list(fpiece.strip().lower())  
 if (color == 'b'):  
 return False  
 else:  
 assign(final, piece)  
 assign(initial, '--')  
 return True  
  
  
while True:  
  
 msg = s.recv(4096) # recieve board from server  
 status = s.recv(4096) # recieve status from server  
 msg = pickle.loads(msg)  
 chessBoard = msg  
 status = pickle.loads(status)  
 print(status)  
 if (int(status) == 1):  
 printBoard()  
 print('You lost the game, Your King has been defeated')  
 break  
  
 printBoard()  
 # BLACKS Turn code goes here...  
 while 1: # main while loop  
 initialp = input('Enter the location of the piece to move')  
 initial = findpiece(initialp)  
  
 if (validinit(initial)):  
 # continue  
 color, piece = list(initial.strip().lower())  
 finald = move(piece, initialp)  
 if (finald == False):  
 continue  
 if (finalassign(initial, initialp, finald)):  
 # conitinue  
 printBoard()  
 msg = pickle.dumps(chessBoard)  
 s.send(msg)  
  
 if not (any('wk' in sublist for sublist in chessBoard)):  
 print('You won the game')  
 break  
  
 break  
 else:  
 print('You cant cut your own piece - INVALID')  
 continue