LAB 1: (01-10-2024)

TIC -TAC-TOE:

```
board={1:' ',2:' ',3:' ',
      4:' ',5:' ',6:' ',
      7:' ',8:' ',9:' '

}

def printBoard(board):
   print(board[1]+'|'+board[2]+'|'+board[3])
   print('-+-+-')
   print(board[4] + '|' + board[5] + '|' + board[6])
   print('-+-+-')
   print(board[7] + '|' + board[8] + '|' + board[9])
   print('\n')

def spaceFree(pos):
   if(board[pos]==' '):
      return True
   else:
      return False

def checkWin():
   if(board[1]==board[2] and board[1]==board[3] and board[1]!=' '):
      return True
   elif(board[4]==board[5] and board[4]==board[6] and board[4]!=' '):
      return True
   elif(board[7]==board[8] and board[7]==board[9] and board[7]!=' '):
      return True
   elif (board[1] == board[5] and board[1] == board[9] and board[1] != ' '):
      return True
   elif (board[3] == board[5] and board[3] == board[7] and board[3] != ' '):
      return True
   elif (board[1] == board[4] and board[1] == board[7] and board[1] != ' '):
      return True
   elif (board[2] == board[5] and board[2] == board[8] and board[2] != ' '):
      return True
   elif (board[3] == board[6] and board[3] == board[9] and board[3] != ' '):
      return True
   else:
      return False

def checkMoveForWin(move):
   if (board[1]==board[2] and board[1]==board[3] and board[1] ==move):
      return True
   elif (board[4]==board[5] and board[4]==board[6] and board[4] ==move):
      return True
   elif (board[7]==board[8] and board[7]==board[9] and board[7] ==move):
      return True
```

```
    elif (board[1]==board[5] and board[1]==board[9] and board[1] ==move):
        return True
    elif (board[3]==board[5] and board[3]==board[7] and board[3] ==move):
        return True
    elif (board[1]==board[4] and board[1]==board[7] and board[1] ==move):
        return True
    elif (board[2]==board[5] and board[2]==board[8] and board[2] ==move):
        return True
    elif (board[3]==board[6] and board[3]==board[9] and board[3] ==move):
        return True
    else:
        return False

def checkDraw():
    for key in board.keys():
        if (board[key]==' '):
            return False
    return True


def insertLetter(letter, position):
    if (spaceFree(position)):
        board[position] = letter
        printBoard(board)

        if (checkDraw()):
            print('Draw!')
        elif (checkWin()):
            if (letter == 'X'):
                print('Bot wins!')
            else:
                print('You win!')
        return

    else:
        print('Position taken, please pick a different position.')
        position = int(input('Enter new position: '))
        insertLetter(letter, position)
        return

player = 'O'
bot ='X'

def playerMove():
    position=int(input('Enter position for O:'))
    insertLetter(player, position)
    return

def compMove():
    bestScore=-1000
```

```
    bestMove=0
    for key in board.keys():
        if (board[key]==' '):
            board[key]=bot
            score = minimax(board, False)
            board[key] = ' '
            if (score > bestScore):
                bestScore = score
                bestMove = key

    insertLetter(bot, bestMove)
    return


def minimax(board, isMaximizing):
    if (checkMoveForWin(bot)):
        return 1
    elif (checkMoveForWin(player)):
        return -1
    elif (checkDraw()):
        return 0

    if isMaximizing:
        bestScore = -1000

        for key in board.keys():
            if board[key] == ' ':
                board[key] = bot
                score = minimax(board, False)
                board[key] = ' '
                if (score > bestScore):
                    bestScore = score
        return bestScore
    else:
        bestScore = 1000

        for key in board.keys():
            if board[key] == ' ':
                board[key] = player
                score = minimax(board, True)
                board[key] = ' '
                if (score < bestScore):
                    bestScore = score
        return bestScore

while not checkWin():
    compMove()
    playerMove()
```

OUTPUT:

```
X| |
-+-+-
 | |
-+-+-
 | |


Enter position for O: 3
X| |O
-+-+-
 | |
-+-+-
 | |


X| |O
-+-+-
X| |
-+-+-
 | |


Enter position for O: 6
X| |O
-+-+-
X| |O
-+-+-
 | |


X| |O
-+-+-
X| |O
-+-+-
X| |


Bot wins!
Enter position for O: 8
X| |O
-+-+-
X| |O
-+-+-
X|O|


You win!
```

Vaccum cleaner:

```
Def vacuum_world():
 # initializing goal_state
 # 0 indicates Clean and 1 indicates Dirty
 goal_state = {'A': '0', 'B': '0'}
 cost = 0
 location_input = input("Enter Location of Vacuum") #user_input of
location vacuum is placed
 status_input = input("Enter status of " + location_input) #user_input if
location is dirty or clean
 status_input_complement = input("Enter status of other room")
 print("Initial Location Condition" + str(goal_state))
 if location_input == 'A':
 # Location A is Dirty.
 print("Vacuum is placed in Location A")
 if status_input == '1':
 print("Location A is Dirty.")
 # suck the dirt and mark it as clean
 goal_state['A'] = '0'
 cost += 1 #cost for suck
 print("Cost for CLEANING A " + str(cost))
 print("Location A has been Cleaned.")
 if status_input_complement == '1':
 # if B is Dirty
 print("Location B is Dirty.")
 print("Moving right to the Location B. ")
 cost += 1 #cost for moving right
 print("COST for moving RIGHT" + str(cost))
 # suck the dirt and mark it as clean
```

```
 goal_state['B'] = '0'
 cost += 1 #cost for suck
 print("COST for SUCK " + str(cost))
 print("Location B has been Cleaned. ")
 else:
 print("No action" + str(cost))
 # suck and mark clean
 print("Location B is already clean.")

if status_input == '0':
 print("Location A is already clean ")
 if status_input_complement == '1':# if B is Dirty
 print("Location B is Dirty.")
 print("Moving RIGHT to the Location B. ")
 cost += 1 #cost for moving right
 print("COST for moving RIGHT " + str(cost))
 # suck the dirt and mark it as clean
 goal_state['B'] = '0'
 cost += 1 #cost for suck
 print("Cost for SUCK" + str(cost))
 print("Location B has been Cleaned. ")
 else:
 print("No action " + str(cost))
 print(cost)
 # suck and mark clean
 print("Location B is already clean.")
 else:
 print("Vacuum is placed in location B")
 # Location B is Dirty.
```

```
if status_input == '1':
print("Location B is Dirty.")
# suck the dirt and mark it as clean
goal_state['B'] = '0'
cost += 1 # cost for suck
print("COST for CLEANING " + str(cost))
print("Location B has been Cleaned.")
if status_input_complement == '1':
# if A is Dirty
print("Location A is Dirty.")
print("Moving LEFT to the Location A. ")
cost += 1 # cost for moving right
print("COST for moving LEFT" + str(cost))
# suck the dirt and mark it as clean
goal_state['A'] = '0'
cost += 1 # cost for suck
print("COST for SUCK " + str(cost))
print("Location A has been Cleaned.")
else:
print(cost)
# suck and mark clean
print("Location B is already clean.")
if status_input_complement == '1': # if A is Dirty
print("Location A is Dirty.")
print("Moving LEFT to the Location A. ")
cost += 1 # cost for moving right
print("COST for moving LEFT " + str(cost))
# suck the dirt and mark it as clean
goal_state['A'] = '0'
```

```
cost += 1 # cost for suck
print("Cost for SUCK " + str(cost))
print("Location A has been Cleaned. ")
else:
print("No action " + str(cost))
# suck and mark clean
print("Location A is already clean.")
# done cleaning
print("GOAL STATE: ")
print(goal_state)
print("Performance Measurement: " + str(cost))
```

Output:

vacuum_world()

OUTPUT:

```
Enter Location of Vacuum (A or B):  A
Enter status of A (0 for Clean, 1 for Dirty):  1
Enter status of the other room (0 for Clean, 1 for Dirty):  0
Initial Location Condition: {'A': '0', 'B': '0'}
Vacuum is placed in Location A
Location A is Dirty.
Cost for CLEANING A: 1
Location A has been Cleaned.
Location B is already clean.
GOAL STATE:
{'A': '0', 'B': '0'}
Performance Measurement: 1
```