**LAB – 1**

1. Implement vaccum cleaner agent.

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 (A/B): ") # user\_input of location vacuum is placed

status\_input = input("Enter status of " + location\_input + " (0 for Clean, 1 for Dirty): ") # status at vacuum location

# other location is the one that's not location\_input

other\_location = 'B' if location\_input == 'A' else 'A'

status\_input\_complement = input("Enter status of " + other\_location + " (0 for Clean, 1 for Dirty): ")

print("Initial Location Condition: " + str(goal\_state))

if location\_input == 'A':

print("Vacuum is placed in Location A")

if status\_input == '1':

print("Location A is Dirty.")

# CLEAN 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 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 CLEAN: " + str(cost))

print("Location B has been Cleaned.")

else:

print("Location B is already clean. No action. Cost: " + str(cost))

else: # status\_input == '0'

print("Location A is already clean.")

if status\_input\_complement == '1':

print("Location B is Dirty.")

print("Moving RIGHT to 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 CLEAN: " + str(cost))

print("Location B has been Cleaned.")

else:

print("Location B is already clean. No action. Cost: " + str(cost))

else: # vacuum in B

print("Vacuum is placed in Location B")

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 B: " + str(cost))

print("Location B has been Cleaned.")

if status\_input\_complement == '1':

print("Location A is Dirty.")

print("Moving LEFT to Location A.")

cost += 1 # cost for moving left

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 CLEAN: " + str(cost))

print("Location A has been Cleaned.")

else:

print("Location A is already clean. No action. Cost: " + str(cost))

else: # status\_input == '0'

print("Location B is already clean.")

if status\_input\_complement == '1':

print("Location A is Dirty.")

print("Moving LEFT to Location A.")

cost += 1 # cost for moving left

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 CLEAN: " + str(cost))

print("Location A has been Cleaned.")

else:

print("Location A is already clean. No action. Cost: " + str(cost))

# done cleaning

print("GOAL STATE:")

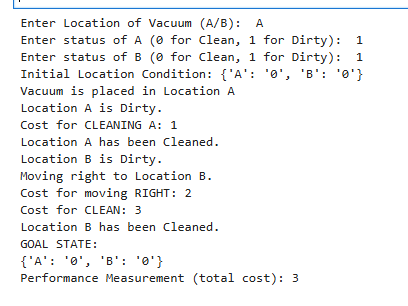
print(goal\_state)

print("Performance Measurement (total cost): " + str(cost))

# Run the vacuum world simulation

vacuum\_world()

**OUTPUT –**

****

1. Implement Tic Tac Toe game.

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):

return board[pos] == ' '

def checkWin():

winning\_combinations = [

(1,2,3), (4,5,6), (7,8,9),

(1,4,7), (2,5,8), (3,6,9),

(1,5,9), (3,5,7)

]

for combo in winning\_combinations:

if board[combo[0]] == board[combo[1]] == board[combo[2]] != ' ':

return True

return False

def checkMoveForWin(move):

winning\_combinations = [

(1,2,3), (4,5,6), (7,8,9),

(1,4,7), (2,5,8), (3,6,9),

(1,5,9), (3,5,7)

]

for combo in winning\_combinations:

if board[combo[0]] == board[combo[1]] == board[combo[2]] == move:

return True

return False

def checkDraw():

return all(board[pos] != ' ' for pos in board) and not checkWin()

def insertLetter(letter, position):

if spaceFree(position):

board[position] = letter

printBoard(board)

else:

print('Position taken, please pick a different position.')

position = int(input('Enter new position: '))

insertLetter(letter, position)

player = 'O'

bot = 'X'

def playerMove():

while True:

try:

position = int(input('Enter position for O (1-9): '))

if position in board and spaceFree(position):

insertLetter(player, position)

break

else:

print("Invalid position or position taken. Try again.")

except ValueError:

print("Please enter a valid number between 1 and 9.")

def compMove():

bestScore = -1000

bestMove = None

for key in board.keys():

if board[key] == ' ':

board[key] = bot

score = minimax(False)

board[key] = ' '

if score > bestScore:

bestScore = score

bestMove = key

insertLetter(bot, bestMove)

def minimax(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(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(True)

board[key] = ' '

if score < bestScore:

bestScore = score

return bestScore

# Game loop

print("Welcome to Tic Tac Toe! Bot is 'X', You are 'O'")

printBoard(board)

while True:

compMove()

if checkWin():

print("Bot wins!")

break

if checkDraw():

print("It's a draw!")

break

playerMove()

if checkWin():

print("You win!")

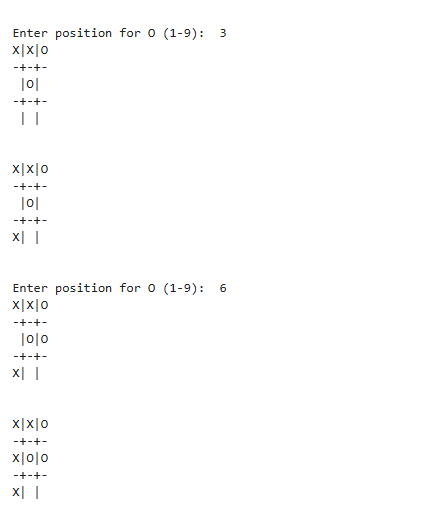
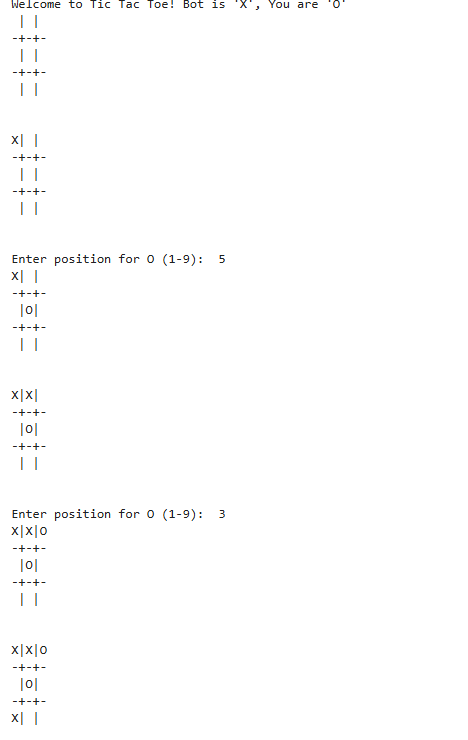
break

if checkDraw():

print("It's a draw!")

break

**OUTPUT –**

****