**Experiment - 3**

Samkit Shah

2019130060

TE Comps

Batch - C

**Aim**:

To build a Tic-Tac-Toe using A\* Algorithm.

Code:

game = {1: ' ', 2: ' ', 3: ' ',

        4: ' ', 5: ' ', 6: ' ',

        7: ' ', 8: ' ', 9: ' '}

def insert(letter, position):

    if is\_free(position):

        game[position] = letter

        Display(game)

        if (is\_draw()):

            print("Draw!")

            exit()

        if Win():

            if letter == comp:

                print("comp wins!")

                exit()

            else:

                print("Player wins!")

                exit()

        return

    else:

        print("Can't insert")

        position = int(input("enter position:  "))

        insert(letter, position)

        return

def is\_free(position):

    if game[position] == ' ':

        return True

    else:

        return False

def Display(game):

    print(game[1] + '|' + game[2] + '|' + game[3])

    print('-+-+-')

    print(game[4] + '|' + game[5] + '|' + game[6])

    print('-+-+-')

    print(game[7] + '|' + game[8] + '|' + game[9])

    print("\n")

def is\_winner(mark):

    pos = False

    for j in range(1,8,3):

        check=True

        for i in range(j,j+2):

            if game[i]!=mark or game[i]!=game[i+1]:

                check=False

                break

        pos = pos|check

    for j in range(1,4):

        check=True

        for i in range(j,j+5,3):

            if game[i]!=mark or game[i]!=game[i+3]:

                check=False

                break

        pos = pos|check

    if(game[1] == mark and game[1]==game[5] and game[5]==game[9]):

        return True

    if game[3]==mark and game[3] == game[5] and game[5]==game[7]:

        return True

    return pos

def is\_draw():

    for key in game.keys():

        if (game[key] == ' '):

            return False

    return True

def player\_turn():

    position = int(input("Enter the position"))

    insert(player, position)

    return

def Win():

    return is\_winner(comp)|is\_winner(player)

def computer():

    Move = 0

    MaxScore = -1000

    for key in game.keys():

        if (game[key] == ' '):

            game[key] = comp

            score = find\_score(game, 0, False)

            game[key] = ' '

            if (score > MaxScore):

                MaxScore = score

                Move = key

    insert(comp, Move)

    return

def find\_score(game, h, is\_bot):

    if (is\_winner(comp)):

        return 1000

    elif (is\_winner(player)):

        return -1000

    elif (is\_draw()):

        return 0

    if (is\_bot):

        bestScore = -1000

        for key in game.keys():

            if (game[key] == ' '):

                game[key] = comp

                score = find\_score(game, h + 1, False)

                game[key] = ' '

                if (score > bestScore):

                    bestScore = score

        return bestScore

    else:

        bestScore = 1000

        for key in game.keys():

            if (game[key] == ' '):

                game[key] = player

                score = find\_score(game, h + 1, True)

                game[key] = ' '

                if (score < bestScore):

                    bestScore = score

        return bestScore

def show\_layout():

    print("This is the grid layout")

    print("1, 2, 3 ")

    print("4, 5, 6 ")

    print("7, 8, 9 ")

    print("\n")

if \_\_name\_\_=="\_\_main\_\_":

    Display(game)

    show\_layout()

    print("Your turn")

    player = 'X'

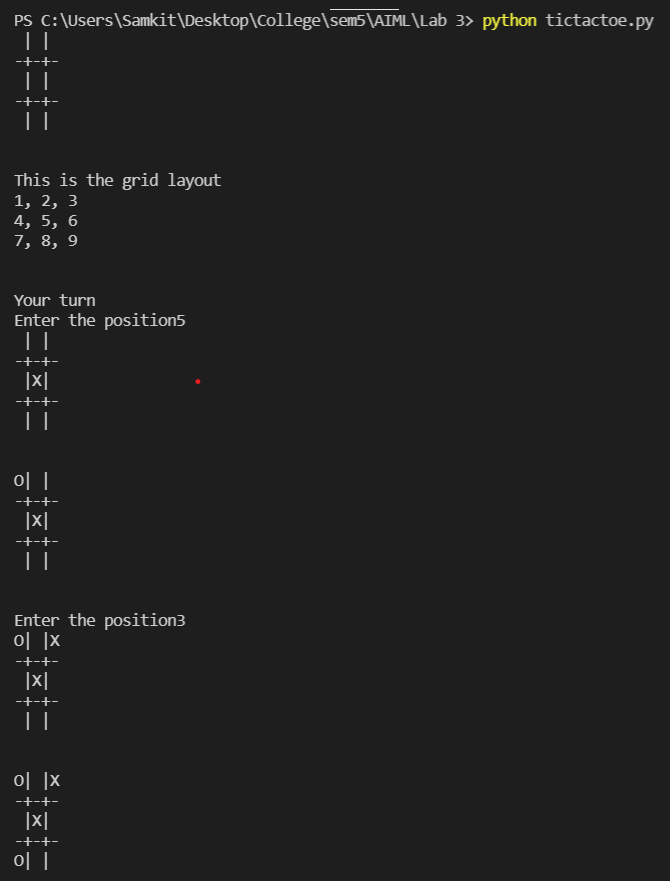
    comp = 'O'

    while not Win():

        player\_turn()

        computer()

**Output**:



Conclusion:

The goal of this project was to implement the tic-tac-toe game using the informed search technique. To determine the best location for the 'X' or 'O,' I first calculated the difference between the winning combinations of bot(O) and user(X) for each choice in that round, then calculated which choice would not lead to computer victory and which would lead to user victory using the maximum values obtained by comparing the number of moves required for victory for each choice and selecting the one with the fewest moves. If there are several equivalent movements, the piece is put at random.