# Importing all necessary libraries

In [1]:

```
import numpy as np
from math import inf as infinity
from copy import deepcopy
```

In [2]:

```
class ticTacToe:

    # Player is X, System is O
    def Player(self,states):
        numberOfO = 0
        numberOfX = 0
        for row in states:
            for item in row:
                if item == "O" or item == "o":
                    numberOfO = numberOfO + 1
                if item == "X" or item == "x":
                    numberOfX = numberOfX + 1
        if numberOfX >= numberOfO:
            return True
        else:
            return False


    def Action(self,states):
        states = np.array(states)
        actionsToDo = np.where(states == '-')
        actionsToDo = list(zip(actionsToDo[0], actionsToDo[1]))
        return actionsToDo


    def resultant(self,states,action):
        if self.Player(states):
            states[action[0]][action[1]] = "O"
            return states
        else:
            states[action[0]][action[1]] = "X"
            return states


    #Checking whether states are terminal or not
    def terminatingStates(self,states):
        col = len(states[0])
        rows = len(states)
        #coloumn elements
        for i in range(col):
            for j in range(rows-1):
                if states[j][i] == states[j+1][i] and states[j][i] != '-':
                    pass
                else:
                    break
            else:
                return True
        #diagonal elements
        for i in range(rows-1):
            if states[i][i] == states[i+1][i+1] and states[i][i] != '-':
                pass
            else:
                break
        else:
            return True
        #row elements
        for i in range(rows):
            for j in range(col - 1):
                if states[i][j] == states[i][j+1] and states[i][j] != '-':
                    pass
                else:
```

```
                    break
            else:
                return True

        # draw match
        isterminatingStates = False
        for row in states:
            try:
                if row.index('-'):
                    isterminatingStates = False
                    break
            except:
                isterminatingStates = True
        return isterminatingStates
        #cross diagnol elements
        for i in range(rows - 1):
            if states[i][(col-1) - i] == states[i+1][(col-1) - i - 1] and states[i][(col-1) - i] != '
-':
                pass
            else:
                break
        else:
            return True


    # Utility Function
    def Utility(self,states):
        rows = len(states)
        col = len(states[0])

        #checking coloumn items
        for i in range(col):
            for j in range(rows-1):
                if states[j][i] == states[j+1][i]:
                    pass
                else:
                    break
            else:
                if states[0][i] == 'O' or states[0][i] == 'o' :
                    return 100
                else:
                    return -100
        #checking row items
        for i in range(rows):
            for j in range(col-1):
                if states[i][j] == states[i][j+1]:
                    pass
                else:
                    break
            else:
                if states[i][0] == 'O' or states[i][0] == 'o' :
                    return 100
                else:
                    return -100

        #checking cross diagnol items
        for i in range(rows-1):
            if states[i][(col-1) - i] == states[i+1][(col-1) - i - 1]:
                pass
            else:
                break
        else:
            if states[i][(col-1) - i] == 'O' or states[i][(col-1) - i] == 'o' :
                    return 100
            else:
                return -100
        return 0

        #checking diagnol items
        for i in range(rows-1):
            if states[i][i] == states[i+1][i+1]:
                pass
            else:
                break
        else:
            if states[i][i] == 'O' or states[i][i] == 'o' :
                    return 100
```

```
            else:
                return -100

    def heuristiSumation(self,countOf_O,countOf_X,nullCount):
        sum = 0
        if countOf_O == 3:
            sum = sum + 100
        if countOf_X == 3:
            sum = sum - 100

        if countOf_X == 2 and nullCount == 1:
            sum = sum - 10
        if countOf_O == 2 and nullCount == 1:
            sum = sum + 10

        if countOf_X == 1 and nullCount == 2:
            sum = sum - 1
        if countOf_O == 1 and nullCount == 2:
            sum = sum + 1
        return sum



    def heuristic(self,states):

        col = len(states[0])
        rows = len(states)
        heuristic = 0

        for i in range(col):
            nullCount = 0
            countOf_X = 0
            countOf_O = 0
            for j in range(rows):
                if states[j][i] == "O" or states[j][i] == 'o':
                    countOf_O += 1
                elif states[j][i] == "X" or states[j][i] == 'x':
                    countOf_X += 1
                else:
                    nullCount += 1
            if countOf_X == 2 and nullCount == 1:
                heuristic = heuristic - 1
            if countOf_O == 2 and nullCount == 1:
                heuristic = heuristic + 1

        for i in range(rows):
            nullCount = 0
            countOf_X = 0
            countOf_O = 0
            for j in range(col):
                if states[i][j] == "O" or states[i][j] == 'o':
                    countOf_O += 1
                elif states[i][j] == "X" or states[i][j] == 'x':
                    countOf_X += 1
                else:
                    nullCount += 1
            if countOf_X == 2 and nullCount == 1:
                heuristic = heuristic - 1
            if countOf_O == 2 and nullCount == 1:
                heuristic = heuristic + 1

        for i in range(rows):
            if states[i][(col-1) - i] == "O" or states[i][(col-1) - i] == "o" :
                countOf_O += 1
            elif states[i][(col-1) - i] == "X" or states[i][(col-1) - i] == "x" :
                countOf_X += 1
            else:
                nullCount += 1

            if countOf_X == 2 and nullCount == 1:
                heuristic = heuristic - 1
            if countOf_O == 2 and nullCount == 1:
                heuristic = heuristic + 1

        for i in range(rows):
            if states[i][i] == "O" or states[i][i] == 'o':
                countOf_O += 1
```

```
            elif states[i][i] == "X" or states[i][i] == 'x':
                countOf_X += 1
            else:
                nullCount += 1
            if countOf_X == 2 and nullCount == 1:
                heuristic = heuristic - 1
            if countOf_O == 2 and nullCount == 1:
                heuristic = heuristic + 1

        return heuristic


    def bestHeuristic(self,states):

        col = len(states[0])
        rows = len(states)
        heuristic = 0

        for i in range(col):
            nullCount = 0
            countOf_X = 0
            countOf_O = 0
            for j in range(rows):
                if states[j][i] == "O" or states[j][i] == 'o':
                    countOf_O += 1
                elif states[j][i] == "X" or states[j][i] == 'x':
                    countOf_X += 1
                else:
                    nullCount += 1
            heuristic = heuristic + self.heuristiSumation(countOf_O,countOf_X,nullCount)

        for i in range(rows):
            nullCount = 0
            countOf_X = 0
            countOf_O = 0
            for j in range(col):
                if states[i][j] == "O" or states[i][j] == 'o':
                    countOf_O += 1
                elif states[i][j] == "X" or states[i][j] == 'x':
                    countOf_X += 1
                else:
                    nullCount += 1
            heuristic = heuristic + self.heuristiSumation(countOf_O,countOf_X,nullCount)


        for i in range(rows):
            if states[i][(col-1) - i] == "O" or states[i][(col-1) - i] == "o" :
                countOf_O += 1
            elif states[i][(col-1) - i] == "X" or states[i][(col-1) - i] == "x" :
                countOf_X += 1
            else:
                nullCount += 1
            heuristic = heuristic + self.heuristiSumation(countOf_O,countOf_X,nullCount)

        for i in range(rows):
            if states[i][i] == "O" or states[i][i] == 'o':
                countOf_O += 1
            elif states[i][i] == "X" or states[i][i] == 'x':
                countOf_X += 1
            else:
                nullCount += 1
            heuristic = heuristic + self.heuristiSumation(countOf_O,countOf_X,nullCount)


        return heuristic
game = ticTacToe()
```

# MiniMax Function

In [3]:

```
def miniMax(states):

    if game.terminatingStates(states):
```

```
        return game.Utility(states)

    if game.Player(states):
        value = -infinity
        for action in game.Action(states):
            """Source for Deepcopy : https://docs.python.org/3/library/copy.html"""
            value = max(value , miniMax(game.resultant(deepcopy(states),action)))
        return value
    else:
        value = infinity
        for action in game.Action(states):
            """Source for Deepcopy : https://docs.python.org/3/library/copy.html"""
            value = min(value , miniMax(game.resultant(deepcopy(states),action)))
        return value
```

# MiniMax with DepthLimit

In [4]:

```
def depthLimitMinMax(states,depth):
    if game.terminatingStates(states):
        return game.Utility(states)
    elif depth == 5:
        return game.heuristic(states)
    if game.Player(states):
        value = -infinity
        for action in game.Action(states):
            """Source for Deepcopy : https://docs.python.org/3/library/copy.html"""
            value = max(value , depthLimitMinMax(game.resultant(deepcopy(states),action) , depth+1
))
        return value
    else:
        value = infinity
        for action in game.Action(states):
            """Source for Deepcopy : https://docs.python.org/3/library/copy.html"""
            value = min(value , depthLimitMinMax(game.resultant(deepcopy(states),action) , depth+1)
)
        return value
```

# MiniMax with AlphaBeta Pruning

In [5]:

```
def alphaBetaPruning(states,alpha,beta):
    if game.terminatingStates(states):
        return game.Utility(states)
    if game.Player(states):
        value = -infinity
        for action in game.Action(states):
            value = max(value , alphaBetaPruning(game.resultant(deepcopy(states),action),alpha,beta
))
            alpha = max(alpha, value)
            if alpha >= beta:
                break
        return value
    else:
        value = infinity
        for action in game.Action(states):
            value = min(value , alphaBetaPruning(game.resultant(deepcopy(states),action),alpha,beta
))
            beta = min(beta, value)
            if alpha >= beta:
                break
        return value
```

# MiniMax with both DepthLimit and Alpha- Beta Pruning

In [6]:

```
def alphaBetaAndDepthLimit(states,alpha,beta,depth):
    if depth == 5:
        return game.heuristic(states)
    if game.terminatingStates(states):
        return game.Utility(states)
    if game.Player(states):
        value = -infinity
        for action in game.Action(states):
            value = max(value , alphaBetaAndDepthLimit(game.resultant(deepcopy(states),action),alph
a,beta,depth+1))
            alpha = max(alpha, value)
            if alpha >= beta:
                break
        return value
    else:
        value = infinity
        for action in game.Action(states):
            value = min(value , alphaBetaAndDepthLimit(game.resultant(deepcopy(states),action),alph
a,beta,depth+1))
            beta = min(beta, value)
            if alpha >= beta:
                break
        return value
```

# Experimental MinMax

In [7]:

```
def experimentalMiniMax(states,alpha,beta,depth):
    if depth == 3:
        return game.bestHeuristic(states)
    if game.terminatingStates(states):
        return game.Utility(states)
    if game.Player(states):
        value = -infinity
        for action in game.Action(states):
            value = max(value ,experimentalMiniMax(game.resultant(deepcopy(states),action),alpha,be
ta,depth+1))
            alpha = max(alpha, value)
            if alpha >= beta:
                break
        return value
    else:
        value = infinity
        for action in game.Action(states):
            value = min(value ,experimentalMiniMax(game.resultant(deepcopy(states),action),alpha,be
ta,depth+1))
            beta = min(beta, value)
            if alpha >= beta:
                break
        return value
```

In [8]:

```
def nextMove(states,typeOfAlgo):
    bestvalue = -infinity
    bestaction = None

    for action in game.Action(states):
        if typeOfAlgo == miniMax:
            value = typeOfAlgo(game.resultant(deepcopy(states),action))

        if typeOfAlgo == depthLimitMinMax:
            value = typeOfAlgo(game.resultant(deepcopy(states),action),0)

        if typeOfAlgo == alphaBetaAndDepthLimit :
            value = typeOfAlgo(game.resultant(deepcopy(states),action),-infinity,infinity,0)

        if typeOfAlgo == alphaBetaPruning :
            value = typeOfAlgo(game.resultant(deepcopy(states),action),-infinity,infinity)

        if typeOfAlgo == experimentalMiniMax :
            value = typeOfAlgo(game.resultant(deepcopy(states),action),-infinity,infinity,0)
```

```
        if value > bestvalue:
            bestvalue = value
            bestaction = action
    return bestaction
```

In [9]:

```
states = [["-","-","-"],["-","-","-"],["-","-","-"]]
noOfSquares = 3
while True:
    if game.terminatingStates(states):
        if game.Utility(states)>0:
            print("System won")
            break
        elif game.Utility(states)<0:
            print("Player won")
            break
        else:
            print("draw")
            break
    action = nextMove(states,miniMax)
    states[action[0]][action[1]] = 'O'
    print("System move",states)
    if game.terminatingStates(states):
        if game.Utility(states)>0:
            print(states)
            print("System won")
            break
        elif game.Utility(states)<0:
            print("Player won")
            break
        else:
            print("Draw")
            break
    i = int(input("enter row number:"))
    j = int(input("enter column number :"))
    states[i][j] = 'X'
    print("Player move",states)
```

```
System move [['O', '-', '-'], ['-', '-', '-'], ['-', '-', '-']]
enter row number:0
enter column number :1
Player move [['O', 'X', '-'], ['-', '-', '-'], ['-', '-', '-']]
System move [['O', 'X', '-'], ['O', '-', '-'], ['-', '-', '-']]
enter row number:1
enter column number :2
Player move [['O', 'X', '-'], ['O', '-', 'X'], ['-', '-', '-']]
System move [['O', 'X', '-'], ['O', '-', 'X'], ['O', '-', '-']]
[['O', 'X', '-'], ['O', '-', 'X'], ['O', '-', '-']]
System won
```

In [10]:

```
states = [["-","-","-"],["-","-","-"],["-","-","-"]]
noOfSquares = 3
while True:
    if game.terminatingStates(states):
        if game.Utility(states)>0:
            print("System won")
            break
        elif game.Utility(states)<0:
            print("Player won")
            break
        else:
            print("draw")
            break
    action = nextMove(states,depthLimitMinMax)
    states[action[0]][action[1]] = 'O'
    print("System move",states)
    if game.terminatingStates(states):
        if game.Utility(states)>0:
            print(states)
            print("System won")
            break
```

```
            break
        elif game.Utility(states)<0:
            print("Player won")
            break
        else:
            print("Draw")
            break
    i = int(input("enter row number:"))
    j = int(input("enter column number :"))
    states[i][j] = 'X'
    print("Player move",states)
```

```
System move [['O', '-', '-'], ['-', '-', '-'], ['-', '-', '-']]
enter row number:1
enter column number :2
Player move [['O', '-', '-'], ['-', '-', 'X'], ['-', '-', '-']]
System move [['O', '-', '-'], ['O', '-', 'X'], ['-', '-', '-']]
enter row number:2
enter column number :1
Player move [['O', '-', '-'], ['O', '-', 'X'], ['-', 'X', '-']]
System move [['O', 'O', '-'], ['O', '-', 'X'], ['-', 'X', '-']]
enter row number:2
enter column number :0
Player move [['O', 'O', '-'], ['O', '-', 'X'], ['X', 'X', '-']]
System move [['O', 'O', 'O'], ['O', '-', 'X'], ['X', 'X', '-']]
[['O', 'O', 'O'], ['O', '-', 'X'], ['X', 'X', '-']]
System won
```

In [11]:

```
states = [["-","-","-"],["-","-","-"],["-","-","-"]]
noOfSquares = 3
while True:
    if game.terminatingStates(states):
        if game.Utility(states)>0:
            print("System won")
            break
        elif game.Utility(states)<0:
            print("Player won")
            break
        else:
            print("draw")
            break
    action = nextMove(states,alphaBetaPruning)
    states[action[0]][action[1]] = 'O'
    print("System move",states)
    if game.terminatingStates(states):
        if game.Utility(states)>0:
            print(states)
            print("System won")
            break
        elif game.Utility(states)<0:
            print("Player won")
            break
        else:
            print("Draw")
            break
    i = int(input("enter row number:"))
    j = int(input("enter column number :"))
    states[i][j] = 'X'
    print("Player move",states)
```

```
System move [['O', '-', '-'], ['-', '-', '-'], ['-', '-', '-']]
enter row number:2
enter column number :0
Player move [['O', '-', '-'], ['-', '-', '-'], ['X', '-', '-']]
System move [['O', 'O', '-'], ['-', '-', '-'], ['X', '-', '-']]
enter row number:0
enter column number :2
Player move [['O', 'O', 'X'], ['-', '-', '-'], ['X', '-', '-']]
System move [['O', 'O', 'X'], ['-', 'O', '-'], ['X', '-', '-']]
enter row number:2
enter column number :1
Player move [['O', 'O', 'X'], ['-', 'O', '-'], ['X', 'X', '-']]
System move [['O', 'O', 'X'], ['-', 'O', '-'], ['X', 'X', 'O']]
```

Draw

In [12]:

```
states = [["-","-","-"],
          ["-","-","-"],
          ["-","-","-"]]
noOfSquares = 3
while True:
    if game.terminatingStates(states):
        if game.Utility(states)>0:
            print("System won")
            break
        elif game.Utility(states)<0:
            print("Player won")
            break
        else:
            print("draw")
            break
    action = nextMove(states,alphaBetaAndDepthLimit)
    states[action[0]][action[1]] = 'O'
    print("System move",states)
    if game.terminatingStates(states):
        if game.Utility(states)>0:
            print(states)
            print("System won")
            break
        elif game.Utility(states)<0:
            print("Player won")
            break
        else:
            print("Draw")
            break
    i = int(input("enter row number:"))
    j = int(input("enter column number :"))
    states[i][j] = 'X'
    print("Player move",states)
```

```
System move [['-', '-', 'O'], ['-', '-', '-'], ['-', '-', '-']]
enter row number:2
enter column number :1
Player move [['-', '-', 'O'], ['-', '-', '-'], ['-', 'X', '-']]
System move [['-', '-', 'O'], ['O', '-', '-'], ['-', 'X', '-']]
enter row number:2
enter column number :2
Player move [['-', '-', 'O'], ['O', '-', '-'], ['-', 'X', 'X']]
System move [['-', '-', 'O'], ['O', '-', '-'], ['O', 'X', 'X']]
enter row number:2
enter column number :0
Player move [['-', '-', 'O'], ['O', '-', '-'], ['X', 'X', 'X']]
Player won
```

In [13]:

```
states = [["-","-","-"],["-","-","-"],["-","-","-"]]
noOfSquares = 3
while True:
    if game.terminatingStates(states):
        if game.Utility(states)>0:
            print("System won")
            break
        elif game.Utility(states)<0:
            print("Player won")
            break
        else:
            print("draw")
            break
    action = nextMove(states,experimentalMiniMax)
    states[action[0]][action[1]] = 'O'
    print("System move",states)
    if game.terminatingStates(states):
        if game.Utility(states)>0:
            print(states)
            print("System won")
            break
```

```
            break
        elif game.Utility(states)<0:
            print("Player won")
            break
        else:
            print("Draw")
            break
    i = int(input("enter row number:"))
    j = int(input("enter column number :"))
    states[i][j] = 'X'
    print("Player move",states)
```

```
System move [['-', '-', '-'], ['-', '-', '-'], ['O', '-', '-']]
enter row number:1
enter column number :2
Player move [['-', '-', '-'], ['-', '-', 'X'], ['O', '-', '-']]
System move [['-', '-', '-'], ['O', '-', 'X'], ['O', '-', '-']]
enter row number:2
enter column number :1
Player move [['-', '-', '-'], ['O', '-', 'X'], ['O', 'X', '-']]
System move [['O', '-', '-'], ['O', '-', 'X'], ['O', 'X', '-']]
[['O', '-', '-'], ['O', '-', 'X'], ['O', 'X', '-']]
System won
```