Projet : Puissance 4 Contre la Machine

Contre-rendu 2

Lors de cette deuxième séance, comme prévu, j'ai tenté de faire la version alpha-beta de l'algorithme : j'ai échoué. Dans certaines situations, le code fonctionne et arrive à retourner un score plus ou moins proche du résultat attendu en une quantité de temps tout à fait satisfaisante, mais dans d'autres cas le code tourne trop longtemps. Ce n'est pas un problème de récurrence infinie, auquel cas le code n'aurait juste pas tourné, mais plutôt, comme au code précédent, l'algorithme traite trop de cas.

Après quelques tests, j'en suis arrivé à la conclusion que non seulement la fonction negamax est mal codée et j'ai mal implémenté l'algorithme alpha-beta, mais aussi ma fonction isWInningMove est peut-être mal faite et ne traite pas tous les cas de victoire pour les deux joueurs. Voici le code :

```
import time
import math
histo = []
columns = [0,0,0,0,0,0,0]
alpha = -42
beta = 42


def canPlay(column):
    if columns[column] > 6:
        return False
    return True


def play(column):
    if canPlay(column):
        columns[column] += 1
        histo.append([columns[column],column])

def playList(plays):
    for i in range(len(plays)):
        columns[int(plays[i]) - 1] += 1
        histo.append([columns[int(plays[i]) - 1], int(plays[i]) - 1])

def isWinningMove(column):                          #Soucis avec ça
    if canPlay(column):
        histoTempo = histo
        histoTempo.append([columns[column],column])
        columnsTempo = columns
        columnsTempo[column - 1] += 1

        histoTempoOrdi = []
        for i in range(len(histoTempo) // 2):
            histoTempoOrdi.append(histoTempo[2*i])
        pos = [columnsTempo[column - 1],column]
        if checkLine(pos, histoTempoOrdi) or checkColumn(pos, histoTempoOrdi) or checkDiago(pos, histoTempoOrdi):
            return True
        return False
```

```python
def nbMoves(histo):
    return len(histo)

def checkLine(pos, histo): #Prend en paramètre la position du jeton testé et l'historique des coups du joueur concerné uniquement
    line = pos[0]
    column = pos[1]
    if [line, column + 1] in histo and [line, column + 2] in histo and [line, column + 3] in histo:
        return True
    elif [line, column - 1] in histo and [line, column - 2] in histo and [line, column - 3] in histo:
        return True
    return False

def checkColumn(pos, histo):    #Prend en paramètre la position du jeton testé et l'historique des coups du joueur concerné uniquement
    line = pos[0]
    column = pos[1]
    if [line - 1, column] in histo and [line - 2, column] in histo and [line - 3, column] in histo:
        return True
    return False

def checkDiago(pos, histo): #Prend en paramètre la position du jeton testé et l'historique des coups du joueur concerné uniquement
    line = pos[0]
    column = pos[1]
    if [line + 1, column - 1] in histo and [line + 2, column - 2] in histo and [line + 3, column - 3] in histo:
        return True
    elif [line - 1, column - 1] in histo and [line - 2, column - 2] in histo and [line - 3, column - 3] in histo:
        return True
    return False


print("[" + str(alpha) + ";" + str(beta) + "]")
```

```python
def negaMax(histo, columns, alpha, beta):                   #Renvoie le score de chaque plateau possible

    histoTempo = []
    columnsTempo = []
    print("[" + str(alpha) + ";" + str(beta) + "]")

    for i in range(len(histo)):
        histoTempo.append(histo[i])
    for i in range(len(columns)):
        columnsTempo.append(columns[i])

    if nbMoves(histo) >= 42:                        #Vérifie si le plateau est un ex aequo
        return 0

    for i in range(0,7):                        #Vérifie si l'ordi peut gagner au prochain tour
        if canPlay(i) and isWinningMove(i):
            return (21 - nbMoves(histoTempo)//2)

    maxi = (21 - nbMoves(histoTempo)//2)            #On majore le score
    print(maxi)


    if beta > maxi :                            #Adapte l'intervalle alpha-beta selon les valeurs de max
        beta = maxi
        if alpha >= beta :                      # Si l'intervalle est vide, on renvoit beta
            return beta

    bestScore = -42

    for i in range(0,7):
        if canPlay(i):
            columnsTempo[i] += 1                        #On simule qu'on joue dans la colonne i
            histoTempo.append([columns[i],i])
            print("[" + str(alpha) + ";" + str(beta) + "]")
            score = -negaMax(histoTempo, columnsTempo, -beta, -alpha)           #A chaque fois qu'on change de joueur, le score change de signe

            if (score >= beta):
```

```python
                return score
            if (score > alpha):
                alpha = score

    return alpha
###
playList("112233")

print(negaMax(histo, columns, alpha, beta))
```

Les print et les dernières lignes ne servent qu'à tester le code.

J'essayerai pendant ses vacances de reprendre la version précédente du code sans l'algorithme alpha-beta et de le réimplémenter en essayant d'éviter les erreurs que j'ai pu

faire à cet essai et j’espère pouvoir avoir un algorithme fonctionnel d’ici la fin de la séance prochaine.