Projet : Puissance 4 Contre la Machine

Contre-rendu 1

Pour ma première séance, j'ai commencé par me documenter sur comment faire un algorithme de puissance 4 le plus compétent possible.

J'ai trouvé un site décrivant étape par étape la création d'un tel algorithme : http://blog.gamesolver.org/solving-connect-four/04-alphabeta/

Je m'en suis donc beaucoup servi et voilà comment cet algorithme fonctionne :

La première version de cet algorithme est un algorithme de type MinMax. Son but est d'attribuer un certain score à chaque organisation du plateau pour savoir combien de tour il manque à l'ordinateur ou au joueur avant de pouvoir gagner. J'ai donc suivi les instructions du site pour faire une première version de l'algorithme en Python, puisque je maitrise beaucoup plus ce langage que celui de l'IDE Arduino et qu'il m'est plus facile de tester mon code (je le traduirais dans l'IDE Arduino quand il fonctionnera), voici le code :

```python
histo = []
columns = [0,0,0,0,0,0,0]


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 isWinningMove(column):
    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
```

```python
def negaMax(histo, columns):                    #Renvoie le score de chaque plateau possible
    print(histo)
    histoTempo = []
    columnsTempo = []
    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 42 - nbMoves(histo)

    bestScore = -42                             #Initialise le pire score possible pour calculer le meilleur
    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])
            score = -negaMax(histoTempo, columnsTempo)          #A chaque fois qu'on change de joueur, le score change de signe
            if score > bestScore:                               #On garde en mémoire le meilleur score
                bestScore = score

    return bestScore



play(4)
play(4)
play(5)
play(5)
print(histo)
print(negaMax(histo, columns))
```

Les variables « histo » et « columns » stockent respectivement l'historique des coups joués en contenant les coordonnées de chaque jeton et le nombre de jetons dans chaque colonnes.

Les premières fonctions, canPlay, play et isWinningMove seront très utiles pour l'algorithme : la première vérifié si la colonne sur laquelle on joue ne serait pas pleine, la deuxième place un jeton tout en mettant à jour histo et columns et la dernière vérifie si un coup permet à l'ordinateur de gagner (inutile de vérifier pour le joueur puisque l'algorithme ne contrôle pas où le joueur joue).

Cette dernière fonction utilise les fonctions checkLine, checkColumn et checkDiago qui vérifient respectivement si un plateau contient une victoire en ligne, en colonne ou en diagonale.

Désormais, on peut faire la fonction cœur de l'algorithme, la fonction negaMax qui utilise le principe minMax : c'est une fonction récursive qui va attribuer à chaque plateau un certain score comme dit plus tôt. Ce score correspond soit à 0 (match nul), soit à la taille du plateau (6 lignes * 7 colonnes = 42) – le nombre de coups joués avant d'arriver à ce plateau. Ainsi, puisque la fonction est récursive, elle va pouvoir tester toutes les possibilités et aura donc pour but de retenir le meilleur score possible, en sachant que le meilleur score est le plus grand puisqu'il correspond au plateau gagnant avec le moins de coups possibles. Enfin, à chaque récursivité, on change le signe du score pour différencier un coup du joueur adverse et un coup de l'ordinateur puisque l'ordinateur ne veut surtout pas avantager le joueur.

Cet algorithme fonctionne, mais il a un problème majeur : il teste beaucoup trop de possibilités, et teste aussi certaines possibilités plusieurs fois, ce qui le rend beaucoup trop long pour fonctionner. Il fonctionne donc bien quand le plateau est déjà rempli puisqu'il aura moins de coups à calculer, mais sinon il est inutilisable.

Mon but à la prochaine séance sera donc de modifier cet algorithme. Le site propose d'en faire un algorithme alpha-beta : on limite le nombre de plateaux que l'algorithme traitera en le faisant traiter uniquement les plateaux avec un score appartenant à l'intervalle [alpha ; beta].