TP 6
Puissance quatre et Minimmax

1 Introduction

L'objectif sera de programmer le jeu de puissance quatre et de proposé une stratégie qui optimise nos chances de gagner en utilisant l'algorithme du Minimax. L'idée générale est assez simple :

On anticipe plusieurs coups (on parle de profondeur), par exemple quatre coup et on cherhce parmi la configuration possible lmaquelle est la plus avantageuse. Cela suppose de construire une heuristique (c-a-d en algorithmique, une heuristique est une méthode de calcul qui fournit rapidement une solution réalisable, pas nécessairement optimale ou exacte, pour un problème d'optimisation difficile.) Une méthode consiste à attribuer à une position de jeu un nombre de sorte que plus le nombre est grand plus on est susceptible de gagner. On cherche donc une fonction notée h qui à chaque position du graphe de l'arène attribue un nombre. Par convention, si on est à une position gagnante, on note hppq " '8 et si on est à une position perdante, on note $h(p) = -\infty$ Par exemple :

- Aux échecs, on peut attribuer à chacune des pièces un nombre de points (classiquement 1 par pion, 3 par fou et cavalier, 5 par tour, 9 par dame et 0 pour le roi), et faire la différence entre ses points et ceux de l'adversaire. Plus vous avez un score élevé et plus vous avez de matériels que votre adversaire et donc vous êtes plus susceptibles de gagner.
- Au puissance 4, on peut attribuer comme valeur à chaque case, le nombre d'alignement de quatre pions possible passant par cette case. Plus une case peut être impliqué dans un grand nombre d'alignement de quatre pièces, plus si vous y placez un jeton, vous êtes susceptible que ce pion vous serve à gagner (en clair, mieux vaut jouer au centre que dans les coins). On peut alors faire la somme des valeurs des emplacements occupés par vos jetons à laquelle vous retirez la somme des valeurs des emplacements de votre adversaire.

évidemment, ces exemples de fonctions ne reflètent que très partiellement la réalité. On peut gagner une partie d'échecs avec moins de pièces que son adversaire si celles-ci sont bien placées. De même, à puissance 4, on peut très bien gagner avec un jeton dans le coin tandis que son adversaire qui a joué au milieu perd. Nous ne cherchons à décrire que partiellement la réalité.

Pour cette raison, on dit que la fonction h s'appelle une heuristique, elle va coder notre compréhension très partielle du jeu. Autrement dit, c'est une façon de coder nos préjugés sur le jeu en question.

2 Le jeu

Le plateau de jeu sera représenté par une liste de liste de 6×7 (6 lignes et 7 colonnes)

Les cases vide valent 0 et 1 pour un jeton du joueur 1 et 2 pour le joueurs 2. Je passe sur les explications du jeu (que vous connaissez).

Si le joueur 1 choisit de mettre un jeton dans la colonne i le jeton se place au dessus du plus haut des jetons de la colonne. Un joueur gagne s'il a quatre jetons alignés. :

```
T=np.array([[0 for k in range(7)] for i in range(6)])
```

1. Programmer une fonction `jouable(T)` qui renvoie `True` si on peut jouer (ajouter un jeton) et `False` sinon.
2. Programmer la fonction `position(T,col)` prenant comme paramètres la configuration `T` du jeu (la liste de liste de 1 2 et 0) et une colonne renvoyant la position `ligne,col` du jeton placé dans la colonne `col`.

3. Programmer une fonction `jeu(T,col,n_joueur)` prenant comme paramètres la configuration `T` du jeu (la liste de liste de 1 2 et 0) une colonne et le numéro du joueur renvoyant la nouvelle configuration T après que le joueur `n_joueur` est placer un jeton dans la colonne `col`.
4. Après avoir récupérer le programme de la fonction `victoire(T,col,n_joueur)`, expliquer brièvement ce que renvoie cette fonction.
5. Programmer la fonction `s(i,j)` renvoyant le score du pion de la ligne i, colonne j :

```
def s(i,j):
    assert i>=0 and j>=0 and i<=5 and j<=6
    bas=min(i+3,5)
    haut=max(0,i-3)
    droite=min(j+3,6)
    gauche=max(0,j-3)
    ..................
```

6. Programmer la matrice `S` de score de l'heuristique énoncer ci-dessus, où `S[i][j]` renvoie le score du jeton de la ligne i colonne j.
7. Programmer `h(T,joeur)`, la fonction heuristique ou fonction de score on choisit dans l'exemple.

3 Minimax

L'algorithme se programmera en récursif. On ne peut pas raisonnablement anticiper tous les coup et on choisira une profondeur p.

Le jeu étant fini, l'arbre associé l'est aussi. Le jeu se termine lorsqu'on aboutit à une feuille f. Pour toute feuille f, on note $\mathcal{S}(f) \in \mathbb{R} \cup \{\pm\infty\}$ le score du joueur Max si cette feuille est atteinte (rappel : le but du joueur Max est de maximiser cette quantité, Min de la minimiser).

On peut définir récursivement une extension de la fonction $\mathcal{S}$, notée $\mathcal{E}$, à tous les nœuds du graphe de la façon suivante :

$$\mathcal{E}(n) = \begin{cases} \mathcal{S}(n) & \text{si } n \text{ est une feuille} \\ \max\{\mathcal{E}(x) \mid x \text{ fils de } n\} & \text{si } n \text{ est contrôlé par le joueur Max ;} \\ \min\{\mathcal{E}(x) \mid x \text{ fils de } n\} & \text{si } n \text{ est contrôlé par le joueur Min.} \end{cases}$$

Il est facile de montrer par récurrence que :

— si, lorsque Max doit jouer en un noeud interne n, il joue un fils y de n tel que $\mathcal{E}(y) = \max\{\mathcal{E}(x) \mid x \text{ fils de } n\}$, alors son score à la fin de la partie vaut au moins $\mathcal{E}(n)$;

— si, lorsque Min doit jouer en un nœud interne n, il joue un fils y de n tel que $\mathcal{E}(y) = \min\{\mathcal{E}(x) \mid x \text{ fils de } n\}$, alors le score de Max à la fin de la partie vaut au plus $\mathcal{E}(n)$;

Stratégie optimale pour chaque joueur. Si chaque joueur joue en essayant de maximiser (resp. minimiser) la quantité $\mathcal{E}$ précédente, le score final du joueur Max à la fin de la partie sera $\mathcal{E}(r)$, avec r la racine.

Nous allons programmer deux fonctions `minimax()` et `maximin()` qui s'appelle mutuellement.

— La fonction `maximin(T,n,p)` qui renvoie le couple meilleur score (pour le joueur n) et colonne (à jouer) quand c'est au joueur n de jouer .

— la fonction `minimin(T,n,p)` qui renvoie le couple du plus petits des meilleurs scores possible (pour le joueur n) et colonne (à jouer) quand c'est au joueur n de jouer en fonction de ce que joue $3-n$.

Programmer les fonctions `maximin(T,n,p)` et `minimin(T,n,p)`. On complètera, en veillant à appeler la fonction `minimin` à l'interieur de `maximin` et respectivement.

```
def maximin(T,n,p):
    if p==0 or not jouable(T):
        return h(T,n),None
    m=-np.infty
    for c in range(7):

        if T[0][c]==0:
```

```
            Tc=np.copy(T)
            ........................
        return m,col
```