Algorithme Mini-Max

1 Introduction

1.1 Le cadre

On va s'intéresser à des jeux à deux joueurs, Alice et Bob, qui jouent alternativement. Le type de jeux que l'on considère sont des jeux à « information totale » : chaque joueur a une vision complète de l'état du jeu à chaque instant.

Par exemple les jeux de cartes ne rentrent pas dans ce cadre (on ne connaît pas le jeu de l'adversaire) mais le jeu d'échecs, le jeu de go, les jeux tels que « *puissance quatre* » rentrent dans ce cadre.

1.2 Notion d'heuristique

On suppose que l'on dispose d'une fonction h à valeur réelle qui à toute position pos du jeu associe un réel de manière que :

- plus $h(pos)$ est grand, meilleure est la position pour Alice ;
- plus $h(pos)$ est petit, meilleure est la position pour Bob.

Une telle fonction s'appelle une *heuristique*.

1.3 Un exemple d'heuristique : « puissance 4 »

On peut associer à chaque case du jeu le nombre d'alignements de quatre pion réalisables qui contiennent la case considérées, nombre qui est donnée par le tableau suivant.

3	4	5	7	5	4	3
4	6	8	10	8	6	4
5	8	11	13	11	8	5
5	8	11	13	11	8	5
4	6	8	10	8	6	4
3	4	5	7	5	4	3

Une heuristique possible est alors la fonction h qui somme positivement les nombres associés aux cases occupées par Alice et négativement celles occupées par Bob.

Par exemple, supposons que la position du jeu soit :

			A	B		
			B	A		
		B	A	A	B	

L'heuristique de cette position pos est alors :

$$h(pos) = 13 + 7 + 5 + 8 - 4 - 5 - 10 - 11 = 3$$

2 Algorithme Mini-Max

2.1 Introduction

On va suivre l'exemple du puissance 4 (bien que pas super-pertinent, puisque toutes les positions peuvent être déterminées sans trop de souci). Lorsqu'Alice ou Bob doit jouer, chacun a plusieurs possibilités (de 1 à 7). Pour choisir le coup à jouer, une idée est de calculer l'heuristique de chacune des nouvelles positions pos^+ que l'on peut obtenir et choisir la plus grande (pour Alice) ou la plus petite (pour Bob).

Par exemple, dans la configuration suivante, c'est à Alice de jouer et elle a 7 possibilités.

			4	5		
		3	B	A	6	
1	2	B	A	A	B	7

La possibilité 4 donnera la plus grande heuristique : $1 + 13 = 14$.

Choix d'Alice	1	2	3	4	5	6	7
Nouvelle heuristique	4	5	9	14	12	7	4

Mais Alice, qui est rusée, peut aussi tenir compte du coup que Bob jouera après ! Elle peut donc calculer, en fonction de son coup, toutes les heuristiques possibles que Bob pourra obtenir et choisir la plus grande possible (rappelons qu'une heuristique petite est à l'avantage de Bob).

On peut encore aller plus loin : Alice peut chercher à calculer la meilleure heuristique qu'elle obtiendra après le coup de Bob. Et on peut ainsi continuer, mais le nombre de calculs à effectuer devient de plus en plus important (croissance exponentielle).

Pour calculer le meilleur coup d'Alice en fonction de l'heuristique considérée, il suffit donc de calculer toutes les heuristiques atteignables en n coups où n est un entier choisi à partir de la position où Alice joue.

- Si n est pair, Bob aura joué le dernier coup et on attribue alors le minimum des valeurs des heuristiques obtenues après ces n étapes : meilleur choix de Bob.
- Si n est impair, c'est Alice qui aura joué le dernier coup. On attribue alors la valeur maximale des heuristiques obtenues après ces n étapes : meilleur choix d'Alice.

2.2 Implémentation

On va définir deux fonctions mutuellement récursives `MaxiMin(pos,n)` et `MiniMax(pos,n)`, qui s'adaptent à toute situation de jeux similaire. La première cherchant à maximiser l'heuristique après n coup en partant de la position `pos`, en supposant que l'adversaire joue au mieux, la seconde cherchant à minimiser l'heuristique après n coup en partant de la position `pos`.

- Pour calculer `MaxiMin(pos,n)` on calcule pour chaque position `files` atteignable à partir de la position `pos` la valeur de l'heuristique de toutes les positions `MiniMax(files,n-1)` et on prend la plus grande.
- Pour calculer `MiniMax(pos,n)` on calcule pour chaque position `files` atteignable à partir de la position `pos` la valeur de l'heuristique de toutes les positions `MaxiMin(files,n-1)` et on prend la plus petite.

On suppose définies les fonctions suivantes :

- `h(pos)` qui donne l'heuristique d'une position.
- `files(pos)` qui renvoie la liste de toutes les positions du jeu atteignables à partir de la position `pos`.

On suppose aussi que l'on a importé de la bibliothèque Numpy la commande `inf` qui représente $+\infty$.

```
from numpy import inf as infini

def MiniMax(pos,n):
    if n==0 or files(pos)==[]:
        return h(p)
    mini=infini
    for f in files(pos):
        s=MaxiMin(f,n-1)
        if s<mini :
            mini=s
    return mini

def MaxiMin(pos,n):
    if n==0 or files(pos)==[]:
        return h(pos)
    maxi=-infini
    for f in files(pos):
        s=MaxiMin(f,n-1)
        if s>maxi :
            maxi=s
    return maxi
```