TP6 MPSI : Puissance 4

fabien.viger@ens.fr
30 Mars 2005

But du TP

On va jouer à Puissance 4 : il s'agit de créer un programme capable de jouer automatiquement, et plutôt fort. On se concentrera sur les algorithmes d'intelligence artificielle classiques, l'interface graphique étant fournie. Bien sûr, **à n'importe quel moment, toute question est la bienvenue.**

Préliminaire

Télécharger le programme d'interface graphique disponible sur la page Web, et le copier-coller dans Caml. À l'exécution, vous devriez voir une partie commencer entre deux joueurs "humains" (aller voir la fenêtre graphique).

Comme on va s'en servir tout au long de ce TP, résumons les rôles des différentes fonctions de l'interface :

- Le plateau est stocké dans une matrice de type `case vect vect`. Le type `case` est défini comme étant soit `vide`, soit `rouge`, soit `bleu`. Donc, si `p` est le nom de notre matrice, `p.(1)` donnera la 2^e colonne du plateau, sous forme de `case vect`, et donc `p.(1).(3)` donnera le type de pion présent dans la 2^e colonne, à la 4^e case en partant du bas.
- La fonction `eval` est la **fonction d'évaluation** de la situation codée par le plateau. Elle prend donc en argument une couleur, un plateau, et renvoie un entier qui sera positif si la situation semble favorable et négatif si elle semble défavorable à la couleur concernée. Si l'entier est au-dessus de 500, cela veut dire qu'on a gagné, et s'il est en-dessous de -500 c'est qu'on a perdu. Attention, `eval` ne marche pas dans des situations aberrantes (du style rouge et bleu ont tous les deux gagné).
- `joue` prend en argument une *colonne* du plateau, un type de pion (rouge ou bleu) et joue dedans, attention il faut que la colonne ne soit pas remplie sinon elle plante.
- `dejoue` prend en argument une colonne et enlève le pion le plus haut (il faut qu'il y ait au moins un pion sinon elle plante)
- `possible` prend en argument un plateau et un numero de colonne et renvoie `true` s'il est possible de jouer dedans.

- `dessine` prend un plateau et le dessine.
- `fin_partie` affiche des informations de fin de partie (elle prend aussi un plateau en argument).

À ce stade, il est important de tout bien comprendre. Essayer :

```
let p = make_matrix 7 6 vide in
 joue p.(1) rouge;
 joue p.(2) bleu;
 joue p.(2) rouge;
 dejoue p.(2);
 dessine p;;
```

Regarder le code de la fonction `main` (tout à la fin du code de l'interface). Essayer de comprendre comment elle marche, sachant que l'argument `j1` de `main` doit être une fonction "joueuse" qui prend en argument un plateau, et renvoie l'entier codant le numéro (entre 0 et 6) de colonne où rouge veut jouer. De même pour `j2`, qui est le joueur bleu.

Remarquer que `humain rouge` renvoie une fonction qui justement, attends qu'on clique dans une colonne pour renvoyer le numéro correspondant. Ainsi, on peut lancer un puissance 4 en mode "deux joueurs humains" avec `main (humain rouge) (humain bleu);;`.

Question 1 : Un premier adversaire

Programmer une fonction `ordi1` qui prenne une couleur `c` et un plateau `p` en argument, et détermine le coup à jouer pour le joueur de couleur `c` qui va optimiser l'évaluation de sa situation : pour le joueur rouge, il s'agit de maximiser `eval`, et pour le bleu, de la minimiser. Ne pas hésiter à utiliser les fonctions `eval`, `joue`, `dejoue` et `possible`, et à me poser des questions si nécessaire.

Essayer de jouer contre lui
- en commençant : `main (humain rouge) (ordi bleu);;`
- en le laissant commencer : `main (ordi rouge) (humain bleu);;`

Question 2 : Un autre, deux fois plus fort !

Faire un adversaire qui réfléchit à profondeur 2 : il va jouer le coup tel que dans le pire des cas (si son adversaire joue le mieux possible), sa situation soit la meilleure possible. Attention, si un coup le fait gagner tout de suite, il le joue sans réfléchir (sinon, comme il anticipe à deux coups, et qu'il se pourrait qu'après un coup gagnant, l'adversaire fasse lui aussi un puissance 4, et dans ce cas la fonction d'évaluation ne marche plus).

On peut essayer de les faire jouer ensemble :

```
main (ordi1 rouge) (ordi2 bleu);;
main (ordi2 rouge) (ordi1 bleu);;
```

Question 3 : Profondeur...

Programmer *récursivement* un adversaire `ordi3` qui réfléchit à une profondeur n... Attention, il faut détecter les situations gagnantes immédiates.

Essayer alors de jouer contre lui, à diverses profondeurs. Par exemple :

```
main (humain rouge) (ordi3 bleu 10);;
```

Il est plus fort, hein ?

Question 4 : Alpha-Beta

cf explication au tableau.

Une fois implémenté, essayer de voir jusqu'à quelle profondeur on peut le faire tourner en temps acceptable (quelques secondes). Jouer contre lui.

Question 5 : Encore mieux ?

Améliorer la fonction d'évaluation.