Algorithmes pour les jeux
Stratégie min-max

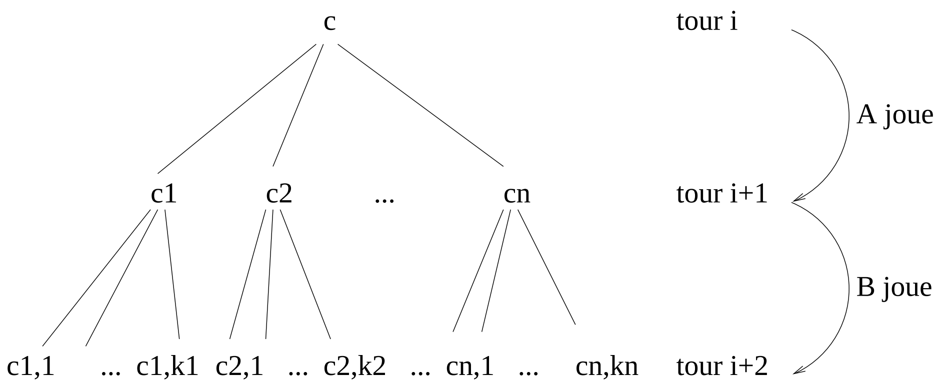
Considérons un jeu à deux joueurs, appelons-les A et B, où la configuration d'une partie est une donnée finie, et où chaque joueur n'a, à chaque tour, qu'un nombre fini de coups possibles. C'est le cas de la majorité des jeux de plateau à deux joueurs. On se propose ici de programmer un algorithme classique pour les jeux : *la stratégie min-max* (également appelée *alpha-beta*).

Principe

On appelle *configuration* une description complète de l'état d'une partie. Par exemple, au *puissance 4*, une configuration est une description de la grille et l'information "c'est à X de jouer". Aux échecs, une configuration est une description de l'échiquier, l'information "c'est aux blancs de jouer" (ou aux noirs), tels pions ont avancé de 2 cases initialement, mais aussi toutes les configurations précédentes (pour les répétitions de position), éventuellement le temps de jeu de chaque joueur, etc.

On suppose que l'on dispose également d'une *fonction d'évaluation*, qui associe à toute configuration une valeur réelle, avec des valeurs particulières pour la défaite, la victoire, et la partie nulle éventuellement. Comme son nom l'indique, cette fonction évalue la position d'un joueur : plus cette valeur est grande et plus le joueur est en bonne position. Il est clair que chaque joueur essaie de maximiser sa fonction d'évaluation et de minimiser celle de l'autre joueur.

La stratégie min-max est basée sur cette idée. Supposons que nous sommes dans une configuration c et que c'est au joueur A de jouer. Soit f la fonction d'évaluation de A. L'arbre des coups possibles à partir de c est représenté ci-dessous :



Si A joue le coup i, qui le mène en position c_i, alors B cherchera à minimiser la fonction d'évaluation de A au coup suivant, c'est-à-dire choisira la configuration $c_{i,j}$ qui minimise f. A a donc intérêt à choisir le coup i qui maximise ce minimum. C'est la stratégie min-max. A choisit donc de jouer le coup i_0 tel que

$$\min_{1\leq j\leq k_{i_o}} f(c_{i_0,j}) = \max_{1\leq i\leq n} \min_{1\leq j\leq k_i} f(c_{i,j})$$

On peut itérer plus loin cette méthode, en prenant comme évaluation sur la configuration $c_{i,j}$ non pas la fonction f mais l'évaluation rendue par la méthode min-max elle-même. L'itérer n fois, c'est prévoir $2n$ coups à l'avance.

1 Préliminaires

- ☐ Ecrire une fonction `minimize` qui prend une fonction de comparaison `comp` (de type `'a -> 'a -> bool`), une fonction d'évaluation `f` (de type `'b -> 'a`), une liste `l` d'objets de type `'b` et qui rend le couple `(x,v)` où `x` est un élément de `l` minimisant la fonction `f` pour la relation `comp`, et `v` la valeur de `(f x)`.
- ☐ En déduire deux fonctions `min_list` et `max_list` qui prennent une fonction `f` de type `'a -> float` et une liste `l` de type `'a list` et qui rende un couple `(x,v)` qui minimise `f`.

2 Stratégie min-max

Soit `'a` le type des configurations du jeu. Soit `eval` la fonction d'évaluation, de type `'a -> float`. On suppose que cette fonction rend `0.0` pour une défaite, `1.0` pour une victoire, et un réel de $]0,1[$ sinon. On suppose également qu'il n'y a pas de partie nulle. Soit `suiv` une fonction, de type `'a -> 'a list` rendant, pour une configuration donnée, la liste des configurations suivantes possibles, c'est-à-dire la liste des coups possibles.

- ☐ Ecrire une fonction `minmax` prenant en arguments `eval`, `suiv` et une configuration `c` et rendant le coup à jouer par la stratégie min-max, pour un seul niveau d'analyse (c'est-à-dire 2 coups).
- ☐ Ecrire une fonction `minmax_n` prenant les même arguments, et un entier `n`, et rendant le coup à jouer pour une analyse sur 2`n` coups.

3 Application : le jeu des allumettes

On va appliquer la stratégie précédente au jeu bien connu des allumettes : Initialement, il y a un certain nombre d'allumettes sur la table. Chaque joueur, à son tour, prend 1, 2 ou 3 allumettes. Celui qui prend la dernière allumette a perdu. Il y a une stratégie gagnante pour ce jeu, et nous allons avoir que la méthode min-max permet de la trouver.

On choisit comme type des configurations le couple d'un booléen, qui indique quel joueur doit jouer, et une entier qui indique combien il reste d'allumettes sur la table.

```
type configuration == bool * int
```

- ☐ Ecrire une fonction `affiche_configuration` affichant une configuration. Par exemple, la configuration `(true,13)` pourra être affichée :

```
au joueur 1 : |||||||||||||
```

- ☐ Ecrire une fonction `suiv` rendant, pour une configuration, la liste des configurations suivantes possibles.
- ☐ Ecrire une fonction `eval` d'évaluation d'une configuration.
- ☐ Ecrire une fonction `joue` qui associe à une configuration la configuration suivante, obtenue avec la stratégie min-max appliquée à une profondeur suffisante pour trouver la stratégie gagnante (si elle existe).
- ☐ En déduire enfin un programme où l'ordinateur fait jouer les deux joueurs, et un programme où vous affrontez l'ordinateur.

S'il vous reste du temps...

S'il vous reste du temps, vous pouvez appliquer la méthode min-max à d'autres jeux. Voici quelques possibilités :

- le puissance 4 ;
- le morpion (prendre une petite grille) ;
- la reine contre les pions : sur un échiquier, les pions noirs affrontent la reine blanche. La reine gagne si elle croque tous les pions, et les pions gagnent s'ils croquent la reine ou si l'un d'eux arrive à la promotion.