Lycée Chateaubriand
Informatique MP 2002/03
Mathieu Giraud

TP Caml n° 7
Exercices et stratégies min-max

1 Quatre exercices

Comme au dernier TP, ces petits exercices sont indépendants.

1.1 Exponentiation rapide

a. Ecrire une une fonction `puissance x n` qui calcule x^n avec x flottant et n entier strictement positif en seulement $\mathcal{O}(\log n)$ multiplications.

1.2 Calcul de π

On peut calculer des approximations successives de π par la méthode de Salamin :

$$\pi_n = \frac{4a_n^2}{1 - 2\sum_{i=1}^{n} 2^i (a_i^2 - b_i^2)}$$

$$a_0 = 1 \qquad b_0 = \frac{1}{\sqrt{2}} \qquad a_{n+1} = \frac{a_n + b_n}{2} \qquad b_{n+1} = \sqrt{a_n b_n}$$

b. Écrire la fonction `salamin n` qui calcule π_n.

1.3 Méthode de Newton

c. La méthode de Newton (ou "méthode des tangentes") permet d'approximer les solutions d'une équation du type $f(x) = 0$ où f est comme il faut. A partir d'une valeur initiale x_0, on définit la suite x_n par :

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

On arrête la récurrence lorsque $|x_{n+1} - x_n|$ passe en-dessous d'un seuil ϵ fixé.

d. Coder la fonction `newton : (float -> float) -> float -> float -> float` telle que l'appel `newton f epsilon x0` effectue la méthode décrite ci-dessus. On pourra tester cette fonction en calculant par exemple $\sqrt{2}$ par `newton (fun x -> x *. x -. 2.) 1e-6 1.`.

1.4 Listes

e. Écrire une fonction `au_moins_deux` retournant les éléments apparaissant au moins deux fois dans une liste donnée.

2 Problème : stratégies min-max

Considérons un jeu à deux joueurs où chaque joueur joue alternativement. L'état du jeu est représenté par une variable de type `position` (type qui sera `'a` dans vos fonctions). Les règles du jeu sont données par une fonction `coups : position -> position list` donnant les coups "autorisés" à partir d'un coup donné; on supppose pour cela qu'à chaque tour un nombre fini de coups est seulement possible. C'est le cas de la majorité des jeux de plateau à deux joueurs (go, échecs, puissance 4, morpion...)

On se donne une fonction `évalue : position -> float` évaluant chaque position. Cette *fonction d'évaluation* reflète les "chances" qu'un joueur a (ou estime qu'il a) de gagner la partie. Elle aura des valeurs particulières pour une victoire, une défaite, et éventuellement pour la partie nulle. Nous prendrons ici le point de vue d'un des joueurs (l'ordinateur), et les évaluations iront de `-.1.` (partie perdue) à `1.` (partie gagnée).

Chaque joueur veut amener l'évaluation de la partie à un score qui lui est le plus favorable possible, *quelque soit la réponse de l'adversaire*. C'est la méthode *minmax* : étant donné une position p et une fonction d'évaluation f, si le premier joueur joue un coup i qui l'amène en p_i, le second joueur va jouer le coup j tel que $f(p_{i,j})$ soit le minimum (étape minimisante *min*). Le premier joueur a donc intérêt a jouer le coup i_0 tel que le minimum $\min_j\ p_{i_0,j}$ soit le plus grand possible (étape maximisante *max*), c'est à dire trouver i_0 tel que

$$\min_j\ p_{i_0,j} = \max_i \min_j\ p_{i,j}$$

La méthode exposée ci-dessous s'effectue à une profondeur de *deux demi-coups*. On peut la généraliser en considérant une stratégie minmax de profondeur quelconque où l'évaluation de chaque position est en fait l'évaluaton rendue récursivement par la méthode minmax elle-même.

2.1 Préliminaires

f. Écrire une fonction `trouvemax` prenant une fonction d'évaluation `évalue : position -> float` et une liste de positions `l`. Cette fonction doit renvoyer un couple `(p,ep)` où `p` est un élément de `l` maximisant la fonction `évalue` et `ep` le maximum en question.

g. Écrire aussi `trouvemin`. On pourra, mais ce n'est pas obligé, modifier légèrement `trouvemax` en une fonction `trouveopt` prenant une fonction de comparaison `cmp : float -> float -> bool` en argument supplémentaire.

2.2 Stratégie min-max

h. Écrire une fonction `minmax` tel que `minmax coups evalue pos n` effectue la stratégie minmax à profondeur `n` sur la position de départ `pos` et renvoie une meilleure solution `(p,ep)`. On pourra coder deux fonctions auxiliaires mutuellement récursives `maximise pos n` et `minimise pos n`.

2.3 Application : le jeu des allumettes

On va aplliquer la stratégie précédente au jeu 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 voir que la méthode minmax permet de la trouver.

Les positions seront ici un couple (`type position == bool * int`). La valeur booléenne indique le joueur à qui c'est le tour de jouer, et l'entier le nombre d'allumettes restant sur la table. Par exemple la position `(false, 5)` signifie que c'est au premier joueur de jouer et qu'il reste cinq allumettes.

i. Écrire la fonction représentant les coups possibles `coups`.

j. Écrire une fonction d'évaluation `évalue`.

k. Tester votre programme `minmax`.

2.4 Élaguage $\alpha\beta$

Une astuce permet de diminuer le nombre de branches explorées par la stratégie minmax. Dans notre exemple, on sait dès l'évaluation du premier fils du nœud entouré que le minimum sera inférieur à `0.1` : on peut donc éviter de parcourir les autres fils. Plus formellement, on maintient deux seuils, α et β, tel que

- pour un nœud *min*, α est égal à la plus grande valeur alors connue de ses ancêtres *max*.
- pour un nœud *max*, β est égal à la plus petite valeur alors connue de ses ancêtres *min*.

Lorsque, dans le parcours de l'arbre, il y a remise en cause de la valeur d'un nœud au delà du seuil courant, il est inutile d'explorer la descendance de ce nœud.

1.* Modifier les fonctions précédentes pour prendre en compte l'élaguage $\alpha\beta$.

Sources

- Exercices : sources diverses (E3A 1999, Luc Maranget...)
- Problème : TP de Jean-Christophe Filliâtre et polycopié de Jean-Marc Alliot et Thomas Schiex.