UNIVERSITÉ CAEN NORMANDIE

UFR des Sciences

Université de Caen Normandie
Département mathématiques-informatique
2022–2023
Bruno Zanuttini

L2 informatique

Notes de cours. Algorithme négamax

Unité SMINFL3A : Programmation Java orientée objet

---

1 Présentation générale

L'algorithme « négamax » est une version de l'algorithme « minmax », qui permet de sélectionner le meilleur coup à jouer dans une situation donnée, pour un jeu

- à deux joueurs, jouant tour à tour,
- à somme nulle, c'est-à-dire tel que le score d'un joueur est toujours l'opposé du score de son adversaire (p.ex. perte/gain, ou partie nulle/partie nulle, ou -30/+30),
- à information complète, c'est-à-dire tel que les deux joueurs ont toujours toutes les informations sur la situation courante (par opposition aux jeux de cartes, par exemple, où la main de l'adversaire est en général inconnue).

La notion de « meilleur coup » est au sens de la relation suivante : un coup est « meilleur » qu'un autre s'il garantit une meilleure issue pour le joueur courant lorsque les deux joueurs jouent leurs meilleurs coups (selon la même relation) jusqu'à la fin de la partie. Par exemple, si

- un coup c_1 permet de garantir au joueur courant une partie nulle, mais ne peut pas lui permettre de gagner,
- un coup c_2 permet au joueur courant de gagner dans tous les cas, sauf lorsque son adversaire joue une suite précise de coups qui amène le joueur courant à perdre,

alors c'est le coup c_1 qui est considéré comme « meilleur ». Il n'y a donc pas de notion de « pari » dans cette notion, ni de tirage aléatoire de stratégies probables de l'adversaire, par exemple.

Dans le cadre de ces notes de cours, nous allons nous limiter à la version « pure » de cet algorithme, qui analyse le jeu jusqu'aux situations terminales. Pour la plupart des jeux, lorsque l'ensemble des parties possibles est trop grand (échecs, dames, etc.), on utilise en réalité une fonction permettant de limiter la profondeur de recherche en associant une valeur à certaines situations non terminales, valeur cherchant à quantifier à quel point la situation est prometteuse.

Les algorithmes « minmax » et « négamax » sont conceptuellement les mêmes ; plus de ressources pourront être trouvées en recherchant « minmax ». La version « négamax » correspond à une façon élégante et simple d'écrire le pseudo-code (et donc d'implémenter l'algorithme). C'est cette version qui est présentée dans la suite ; on se limite par ailleurs aux jeux dont l'issue est ternaire (perte/gain/partie nulle), mais il est très facile de généraliser à des scores.

2 Principe

Note On pourra commencer par regarder l'exemple d'exécution présenté dans la partie 4.

L'algorithme se conçoit facilement de manière récursive. Son principe est d'affecter une valeur à une situation pour le joueur courant (devant jouer dans cette situation). La valeur indiquera le score (minimum) garanti pour le joueur courant à la fin de la partie, s'il joue son meilleur coup dans la situation courante.

On distingue deux cas.

Le cas de base, où s'arrête la récursion, est celui des situations terminales. Dans ce cas, la valeur associée à la situation est simplement le score du joueur courant ($+1$ s'il a gagné, 0 pour une partie nulle, et -1 s'il a perdu). En effet, étant donné qu'il n'y a aucun coup à jouer, le score minimum garanti est tout simplement celui obtenu.

Dans le cas général, la situation courante s n'est pas terminale, et le joueur courant j a donc un certain nombre de coups qu'il peut jouer. L'algorithme affecte alors à la situation s la valeur du meilleur coup ; pour cela, il énumère tous les coups disponibles, et pour chaque coup c :

1. simule le coup c dans la situation courante s, ce qui résulte en une nouvelle situation s' où c'est à l'adversaire j' de jouer (cet adversaire devient donc le joueur courant dans la situation s'),
2. effectue un appel récursif dans la situation s', ce qui donne donc une valeur v' pour s' (du point de vue de j'),
3. affecte au coup c l'opposé $-v'$ de cette valeur ; ceci reflète le fait que si le joueur courant joue le coup c dans la situation courante s, alors par la suite son adversaire j' pourra obtenir la valeur v' ce qui donnera donc une valeur $-v'$ pour j (le jeu étant à somme nulle, par hypothèse).

L'algorithme obtient ainsi une valeur pour chaque coup dans la situation courante s, et il affecte donc à s la meilleure parmi ces valeurs.

3 Pseudo-code

Les algorithmes 1 et 2 (page 3) présentent le pseudo-code correspondant à ce principe de fonctionnement. L'algorithme 2 consiste à calculer les valeurs des situations, tandis que l'algorithme 1 est l'appel principal, permettant de sélectionner le meilleur coup en fonction des valeurs retournées. Dans l'algorithme 1, bien noter le signe $-$ devant l'appel à `évaluer` à la 6$^{\text{e}}$ ligne.

4 Exemple d'exécution

Considérons le jeu du morpion, et la situation courante s suivante, où c'est au joueur X de jouer :

X	O	X
O	O	.
.	X	.

Pour affecter une valeur à la situation s, l'algorithme énumère les coups possibles pour le joueur X.

1. Considérons le premier coup c, consistant à jouer au milieu à droite. La situation résultante est la situation s' suivante, où c'est au joueur O de jouer :

X	O	X
O	O	X
.	X	.

   Ce n'est pas une situation terminale, donc l'algorithme effectue à nouveau des appels récursifs pour lui affecter une valeur. Les deux appels récursifs donnent deux situations :

   (a) $s''_1 =$

X	O	X
O	O	X
O	X	.

   qui n'est pas terminale, et où c'est au joueur X de jouer, avec un seul coup disponible ; l'appel récursif donne alors la situation $s''' =$

X	O	X
O	O	X
O	X	X

   où X gagne ; on a donc une valeur de -1 pour s''' (puisque le joueur courant y est O), et donc une valeur de $+1$ pour s''_1 ;

   (b) $s''_2 =$

X	O	X
O	O	X
.	X	O

   qui de même n'est pas terminale, et où le seul coup disponible pour le joueur courant X donne la situation

X	O	X
O	O	X
X	X	O

   qui est une partie nulle (valeur de 0 pour cette dernière situation, et donc de $-0 = 0$ pour s''_2).

   Pour la situation s', les deux coups se voient donc affecter une valeur de $-(+1) = -1$ et de $-0 = 0$, respectivement. La valeur affectée à la situation s', où le joueur courant est O, est donc $\max(-1, 0) = 0$. Ceci reflète le fait que dans la situation s', le joueur courant O a un coup garantissant une partie nulle (en l'occurrence, jouer en bas à droite).

Entrées : La situation courante s du jeu (supposée non terminale), le joueur courant j
Sortie : Le meilleur coup à exécuter pour j dans s
début
    meilleureValeur ← `null`
    meilleurCoup ← `null`
    pour *chaque coup c possible pour j dans s* **faire**
        s' ← situation résultant de l'exécution de c dans s pour j
        j' ← adversaire de j
        v ← − `évaluer`(s', j')
        si *meilleureValeur* $=$ *null ou* $v >$ *meilleureValeur* **alors**
            meilleureValeur ← v
            meilleurCoup ← c
        fin
    fin
    retourner *meilleurCoup*
fin

Algorithme 1 : Algorithme `négamax`

Entrées : La situation courante s du jeu, le joueur courant j
Sortie : La valeur de s pour j
début
    si s *est gagnante pour* j **alors**
        retourner $+1$
    sinon si s *est perdante pour* j **alors**
        retourner -1
    sinon si s *est nulle* **alors**
        retourner 0
    sinon
        `// s n'est pas une situation terminale`
        res ← `null`
        pour *chaque coup c possible pour j dans s* **faire**
            s' ← situation résultant de l'exécution de c dans s pour j
            j' ← adversaire de j
            v ← − `évaluer`(s', j')
            si *res* $=$ *null ou* $v >$ *res* **alors**
                res ← v
            fin
        fin
        retourner *res*
    fin
fin

Algorithme 2 : Algorithme `évaluer`

2. Considérons désormais le second coup c, consistant à jouer en bas à gauche. La situation résultante est la situation s' suivante, où c'est au joueur O de jouer :

X	O	X
O	O	.
X	X	.

L'algorithme effectue des appels récursifs et obtient les situations

X	O	X
O	O	O
X	X	.

et

X	O	X
O	O	.
X	X	O

, où le joueur courant est X. Dans la première situation, O a gagné (valeur de -1 pour X), et dans la seconde, un appel récursif avec le seul coup disponible pour X aboutit à une partie nulle (valeur de 0). Dans la situation s', où le joueur courant est O, les deux coups ont donc des valeurs de $+1$ et 0, respectivement, et la situation s' a donc une valeur de $+1$.

3. Enfin, considérons le troisième coup c, consistant à jouer en bas à droite. La situation résultante est la situation s' suivante, où c'est au joueur O de jouer :

X	O	X
O	O	.
.	X	X

L'algorithme effectue des appels récursifs et obtient les situations

X	O	X
O	O	O
.	X	X

et

X	O	X
O	O	.
O	X	X

, où le joueur courant est X. Dans la première situation, O a gagné (valeur de -1 pour le joueur courant X), et dans la seconde, le seul coup disponible pour X aboutit à sa victoire (valeur de $+1$). Dans la situation s', les deux coups ont donc des valeurs de $+1$ et -1, respectivement, pour le joueur courant O, et la situation s' a donc une valeur de $+1$.

Au final, dans la situation courante s, il y a trois coups pour le joueur courant X, aboutissant à des situations s' de valeurs respectives $0, +1, +1$. Dans s, les trois coups ont donc des valeurs respectives $0, -1, -1$, et donc s a une valeur de 0, correspondant au premier coup. L'algorithme recommande donc de jouer au milieu à droite. Les valeurs reflètent bien le fait que si X joue au milieu à droite, alors il s'assure (au moins) une partie nulle, quoi que O joue, tandis que s'il joue ailleurs, il permet à son adversaire de s'assurer une victoire.