

Intelligence Artificielle

Algorithmes pour jeux

Emmanuel ADAM

Université Polytechnique des Hauts-De-France



UPHF/INSA HdF

- 1 Présentation des jeux
- 2 Algorithme MiniMax - NegaMax
 - Evaluation des coups en MiniMax
 - Exemple de MiniMax
 - Algorithme MiniMax
 - Algorithme NegaMax
 - Exemple de NegaMax
- 3 Algorithme Alpha - Beta
 - Algorithme $\alpha - \beta$
- 4 Evaluation
- 5 Exemple sur Othello
 - Adaptation
- 6 Monte-Carlo

Jeux à deux joueurs

Caractéristiques des jeux

- Les deux adversaires (O et H) jouent à tour de rôle,
- La situation globale du jeu est connue de chacun des joueurs,
- La chance n'intervient pas,
- Les jeux sont dits "à somme nulle" : les gains d'un joueur représentent exactement les pertes de l'autre joueur.

Algorithme MiniMax - NegaMax

Algorithme MiniMax - NegaMax

- Complexité : dimension de l'espace d'états
 - Morpion : facteur de branchement ≈ 3 , nb de demi-coups = 9 au plus
dimension $\approx 3^9 = 19683$
 - Echec : facteur de branchement ≈ 35 , nb de demi-coups ≈ 30
dimension $\approx 35^{30} !!!$
- Nécessité d'une profondeur maximale de résolution
- Nécessité d'une fonction d'évaluation pour estimer les noeuds non feuilles
- Chaque joueur joue le coup de gain maximal pour lui, en sachant que, et en prenant en compte que, l'adversaire fera de même

Evaluation dans l'algorithme MiniMax - NegaMax

Evaluation

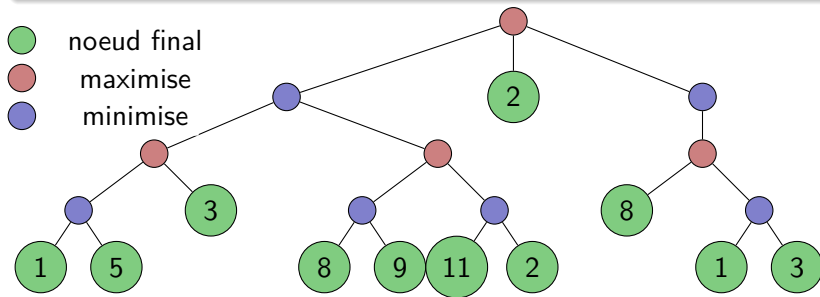
- Stratégie : **profondeur limitée**
- L'heuristique h évalue la qualité d'un noeud terminal (feuille ou de profondeur maximale)
- Comportement du joueur O_{rdi} :
 - O parcourt en profondeur et note les noeuds terminaux de niveau n d'une branche
 - O transmet les notes au noeud $n - 1$ et sélectionne :
 - le noeud de valeur maximale si le trait appartient à O (étape maximisante)
 - le noeud de valeur minimale si le trait appartient à H (étape minimisante)
 - O transmet la note du noeud $n - 1$ au noeud $n - 2$ et note les autres noeuds de niveau $n - 1$
 - O poursuit sa notation pour pouvoir choisir parmi les noeuds de niveau 1

Exemple de MiniMax 1/4

Evaluation

Arbre présentant l'espace d'états pour O.

- O cherche à maximiser ses gains
- sachant que H cherchera à lui les minimiser

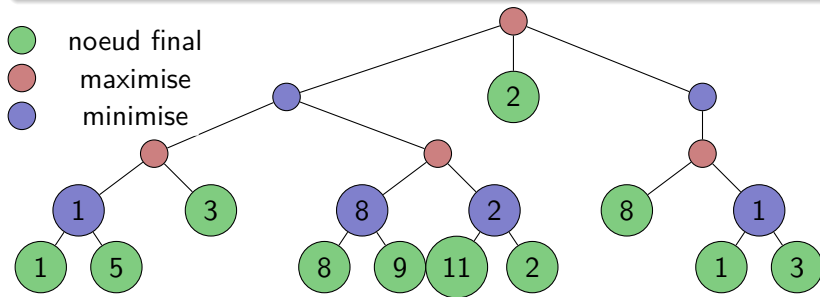


Exemple de MiniMax 2/4

Evaluation

Arbre présentant l'espace d'états pour O.

- O cherche à maximiser ses gains
- sachant que H cherchera à lui les minimiser

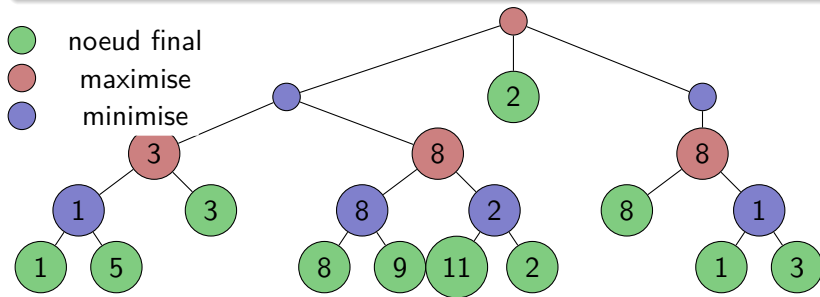


Exemple de MiniMax 3/4

Evaluation

Arbre présentant l'espace d'états pour O.

- O cherche à maximiser ses gains
- sachant que H cherchera à lui les minimiser

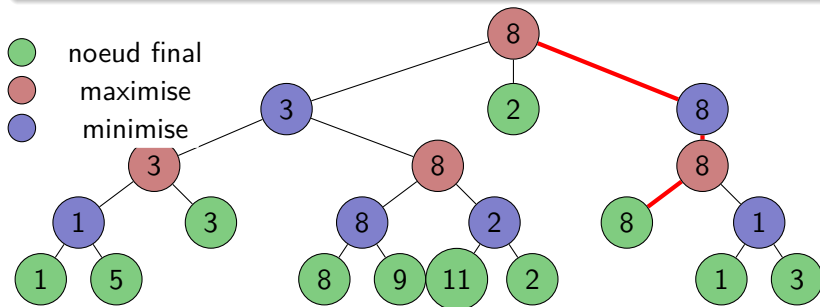


Exemple de MiniMax 4/4

Evaluation

Arbre présentant l'espace d'états pour O.

- O cherche à maximiser ses gains
- sachant que H cherchera à lui les minimiser



Exemple d'algorithme MiniMax I

```
procedure MINIMAX(s : situation)
    minimax; valSousArbre : entier ;

    if ESTFEUILLE(s) then
        RETOURNER(h(s))
    end if

    if ESTMAX(s) then
        minimax  $\leftarrow -\infty$ 
        for all s'  $\in$  s.successeurs do
            valSousArbre  $\leftarrow$  MINIMAX(s')
            if (minimax < valSousArbre) then
                minimax  $\leftarrow$  valSousArbre
            end if
        end for
        RETOURNER(minimax)
    end if
```

Exemple d'algorithme MiniMax II

```
if ESTMIN(s) then
   $minimax \leftarrow +\infty$ 
  for all  $s' \in s.successeurs$  do
     $valSousArbre \leftarrow \text{MINIMAX}(s')$ 
    if ( $minimax > valSousArbre$ ) then
       $minimax \leftarrow valSousArbre$ 
    end if
  end for
  RETOURNER( $minimax$ )
end if
end procedure
```

Exemple d'algorithme NegaMax I

L'Algorithme du NegaMax est simplifié :

O prend le gain minimal de H en prenant l'opposé de son NegaMax et inversement

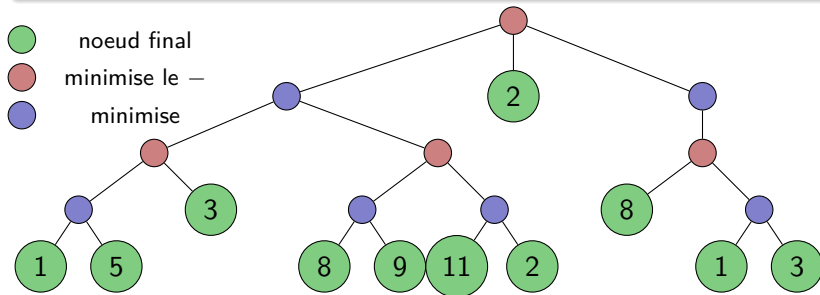
```
procedure NEGAMAX(s : situation)
    minimax; valSousArbre : entier ;
    if ESTFEUILLE(s) then
        RETOURNER(h(s))
    end if
    minimax  $\leftarrow -\infty$ 
    for all s'  $\in$  s.successeurs do
        valSousArbre  $\leftarrow$  - NEGAMAX(s')
        if (minimax < valSousArbre) then
            minimax  $\leftarrow$  valSousArbre
        end if
    end for
    RETOURNER(minimax)
end procedure
```

Exemple de NegaMax 1/4

Evaluation

Arbre présentant l'espace d'états pour O.

- O cherche à minimiser le négatif des gains possibles sachant que H cherchera à lui les minimiser

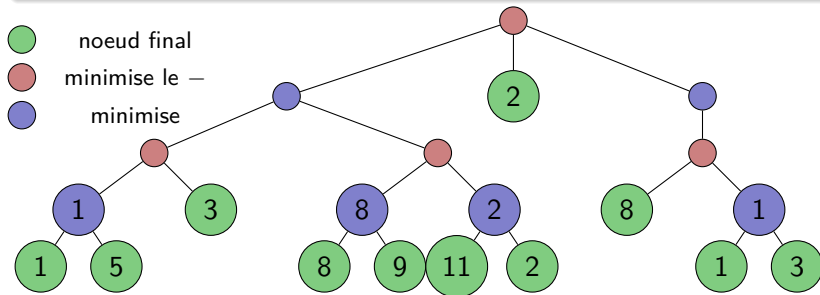


Exemple de NegaMax 2/4

Evaluation

Arbre présentant l'espace d'états pour O.

- O cherche à minimiser le négatif des gains possibles sachant que H cherchera à lui les minimiser

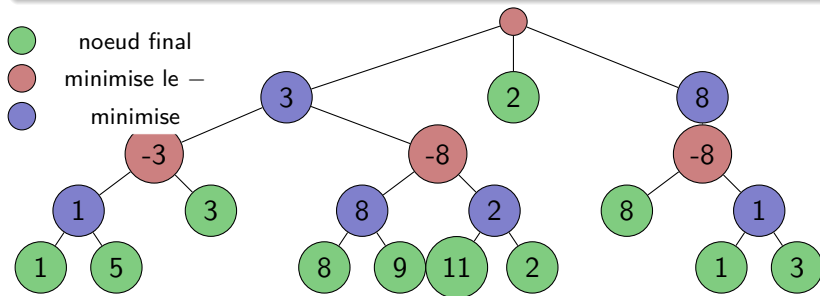


Exemple de NegaMax 3/4

Evaluation

Arbre présentant l'espace d'états pour O.

- O cherche à minimiser le négatif des gains possibles sachant que H cherchera à lui les minimiser

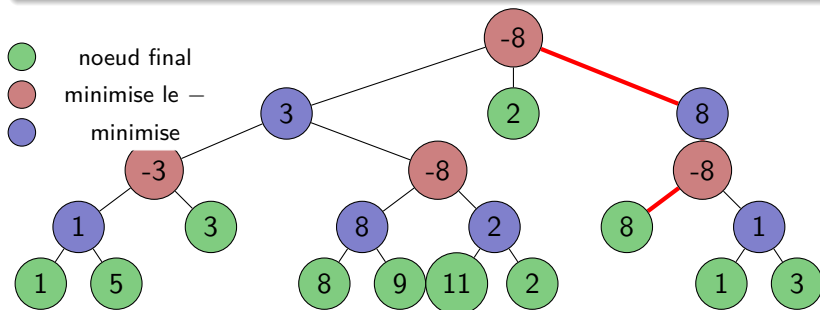


Exemple de NegaMax 4/4

Evaluation

Arbre présentant l'espace d'états pour O.

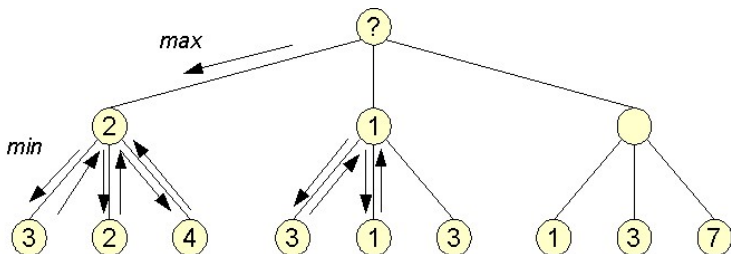
- O cherche à minimiser le négatif des gains possibles sachant que H cherchera à lui les minimiser



Algorithme Alpha - Beta

Algorithme Alpha - Beta

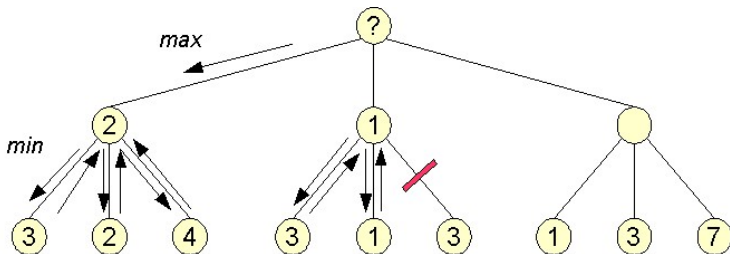
- Problème du MiniMax/NégaMax : Exploration complète de l'arbre, de l'espace d'états
- Possibilité d'élaguer des branches en fonction des découvertes



Algorithmme Alpha - Beta

Algorithmme Alpha - Beta

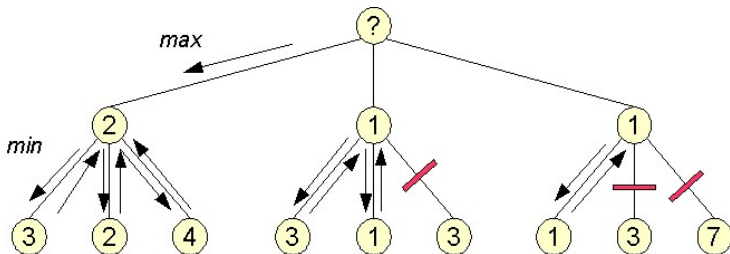
- Problème du MiniMax : Exploration complète de l'arbre, de l'espace d'états
- Possibilité d'élager des branches en fonction des découvertes



Algorithme Alpha - Beta

Algorithme Alpha - Beta

- Problème du MiniMax : Exploration complète de l'arbre, de l'espace d'états
- Possibilité d'élaguer des branches en fonction des découvertes



Alpha - Beta

α et β

On distingue deux seuils, appelés α (pour les noeuds Min) et β (pour les noeuds Max) :

- le seuil α , pour un noeud Min s , est égal à la plus grande valeur (déjà déterminée) de tous les noeuds Max ancêtres de s . Si la valeur de s devient inférieure ou égale à α , l'exploration de sa descendance peut être arrêtée ;
- le seuil β , pour un noeud Max s , est égal à la plus petite valeur (déjà déterminée) de tous les noeuds Min ancêtres de s . Si la valeur de s devient supérieure ou égale à β , l'exploration de sa descendance peut être arrêtée.
- α et β sont initialisés réciproquement à $-\infty$ et $+\infty$.

Exemple d'algorithme $\alpha - \beta$ I

```
procedure ALPHABETA(s : situation,  $\alpha$  : entier,  $\beta$  : entier)
  minimax; valSousArbre : entier;
  if ESTFEUILLE(s) then RETOURNER(h(s))
  end if

  if ESTMAX(s) then
    minimax  $\leftarrow \alpha$ 
    for all s'  $\in$  s.successeurs do
      valSousArbre  $\leftarrow$  ALPHABETA(s', minimax,  $\beta$ )
      if (minimax < valSousArbre) then
        minimax  $\leftarrow$  valSousArbre
      end if
      if (minimax  $\geq \beta$ ) then RETOURNER(minimax)
      end if
    end for
    RETOURNER(minimax)
  end if
```

Exemple d'algorithme $\alpha - \beta$ II

```
if ESTMIN(s) then  
    minimax  $\leftarrow \beta$   
    for all s'  $\in s.successeurs$  do  
        valSousArbre  $\leftarrow$  ALPHABETA(s',  $\alpha$ , minimax)  
        if (minimax  $>$  valSousArbre) then  
            minimax  $\leftarrow$  valSousArbre  
        end if  
        if (minimax  $\leq \alpha$ ) then RETOURNER(minimax)  
        end if  
    end for  
    RETOURNER(minimax)  
end if  
end procedure
```

Exemple d'algorithme $\alpha - \beta$ simplifié

Voici l'algorithme $\alpha - \beta$ simplifié sur base du NegaMax :

```
procedure ALPHABETANEGA( $s$  : situation,  $\alpha$  : entier,  $\beta$  : entier)
  if ESTFEUILLE( $s$ ) then
    RETOURNER( $h(s)$ )
  else
    for all  $s' \in s.successeurs$  do
       $\beta \leftarrow \max(\alpha, -ALPHABETANEGA(s', -\beta, -\alpha))$ 
      if  $\alpha \geq \beta$  then
        RETOURNER( $\alpha$ )
      end if
       $\alpha \leftarrow \max(\alpha, \beta)$ 
    end for
    RETOURNER( $\alpha$ )
  end if
end procedure
```

Fonction d'évaluation

Heuristique

- Dans le cadre des jeux, l'heuristique dépend souvent des critères liés :
 - au matériel (jetons, ...);
 - à la mobilité (nombre de coups possibles);
 - à la position (des pièces, ...).
 - $h = p_{matériel} \times c_{matériel} + p_{mobilité} \times c_{mobilité} + p_{position} \times c_{position}$
- les pondérations peuvent évoluer en cours de partie

Exemple sur Othello

Othello

- Grille de 64 cases dont 4 sont occupées : le jeu prend fin au plus tard après 60 coups.

	A	B	C	D	E	F	G	H
1								
2								
3								
4				○	●			
5				●	○			
6								
7								
8								

Heuristique pour Othello

Heuristique pour Othello

- Reprenons les 3 critères définissant l'heuristique pour un jeu, ici pour une couleur :
 - le matériel = nombre de pions de la couleur,
 - la mobilité = nombre de cases jouables par la couleur,
 - la valeur d'une position = somme des valeurs des cases occupées par la couleur.
- On utilise une grille évaluant la valeur d'une case, adaptée au

jeu :

500	-150	30	10	10	30	-150	500
-150	-250	0	0	0	0	-250	-150
30	0	1	2	2	1	0	30
10	0	2	16	16	2	0	10
10	0	2	16	16	2	0	10
30	0	1	2	2	1	0	30
-150	-250	0	0	0	0	-250	-150
500	-150	30	10	10	30	-150	500

Stratégie pour Othello

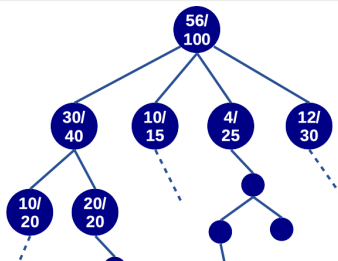
Stratégie

- Evolution de l'importance (du poids) des 3 critères en cours de jeu
 - debut : mobilité et position favorisées ($p_{mobilité}$ au maximum et $p_{position}$ important)
 - milieu de partie : importance de conquérir bords et points : $p_{position}$ au maximum
 - fin de partie : importance du nombre de jetons de la bonne couleur : $p_{matériel}$ au maximum
- Evolution du facteur de branchement : plus faible vers la fin
 - donc possibilité d'augmenter la profondeur de calcul
- Possibilité d'évaluer le niveau du joueur adverse

Recherche arborescente Monte Carlo (MCTS)

Stratégie

- A partir d'une situation, jouer une partie jusqu'à une situation finale (gain, perte, partie nulle)
- Choix **quasi-aléatoire** des noeuds, guidé légèrement par les retours positifs précédents
 - → pas d'heuristique utilisée pour trier les noeuds et élaguer l'arbre
 - → l'IA joue contre elle-même un grand nombre de fois et mémorise les chemins statistiquement intéressants



Recherche arborescente Monte Carlo (MCTS)

Stratégie

- A chaque noeud on associe 2 valeurs : le nb de simulations effectuées passant par ce noeuds et le nb de simulations gagnantes atteintes par ce noeud
- Principe proche du Q-Learning :
 - Atteindre une feuille : par exploration ou par suivi des plus prometteurs
 - Création de noeuds sous la feuille
 - Créer une branche : simuler une partie complète à partir d'un noeud fils (tiré au hasard ou non)
 - Remontée des observations à partir du résultat issu de la branche créée

Recherche arborescente Monte Carlo (MCTS)

Efficacité

- Pas nécessaire pour jeux "simples" (dont l'arbre des situations peut être balayé, comme OXO, ...)
- Très utile pour jeux à hauts niveaux de liberté (dame, échec, go, ...)
- Style de jeu déduit de longue séries de parties simulées :
 - → le style est **propre** à l'IA, il n'imité pas celui d'experts humains

MCTS : algorithme

MCTS : Composition d'un noeud

Classe Noeud

feuille \leftarrow FAUX

terminal \leftarrow FAUX

ia \leftarrow FAUX

noeudsFils $\leftarrow \emptyset$

nbVisites \leftarrow 0

sommeGains \leftarrow 0 \triangleright somme de gains obtenus par ce noeuds ou ses descendants

parent $\leftarrow \emptyset$

\triangleright AJOUTER LES CHAMPS SPECIFIQUES AU JEU A GERER

\triangleright Vrai si noeud sans fils créé

\triangleright Vrai si noeud sans autre action possible

\triangleright Vrai si noeud joue par l'IA

\triangleright contient les noeuds fils directs

\triangleright nb de passages par ce noeud

\triangleright parent direct du noeud

MCTS : algorithme

MCTS : Choisir et jouer

procedure SELECTACTION

▷ à partir du noeud courant, descendre jusqu'à un noeud feuille

terminal \leftarrow *VRAI*

noeud \leftarrow *noeudCourant*

while *noeud.feuille* \neq *VRAI* **do**

noeud \leftarrow *noeud.select()*

end while

rollOut(*node*)

▷ jouer une partie jusqu'au bout

end procedure

MCTS : algorithme

MCTS : Choisir

procedure SELECT

- ▷ à partir du noeud courant, choisir le "meilleur" noeud fils
- ▷ privilégier à la fois le gain et l'exploration

$\text{noeudChoisi} \leftarrow \emptyset$

$\text{meilleurValeur} \leftarrow -\infty$

$c \leftarrow \sqrt{2}$

for all $\text{fils} \in \text{noeudsFils}$ **do**

▷ *hasard()* retourne un nb au hasard entre 0 et 1

if $\text{fils.nbVisites} = 0$ **then** $\text{valeur} \leftarrow \text{hasard}() + c \times \sqrt{\log(\text{nbVisites} + 1)}$
else

$\text{valeur} \leftarrow \frac{\text{fils.sommeGains}}{\text{fils.nbVisites}} + c \times \sqrt{\frac{\log(\text{nbVisites} + 1)}{\text{fils.nbVisites}}}$

end if

if $\text{valeur} \geq \text{meilleurValeur}$ **then**

$\text{noeudChoisi} \leftarrow \text{child}$

$\text{meilleurValeur} \leftarrow \text{valeur}$

end if

end for

$\text{noeudChoisi.parent} \leftarrow \text{ceNoeud}$

return noeudChoisi

end procedure

MCTS : algorithme

MCTS : Jouer

```

procedure ROLLOUT(noeud)
    ▷ continuer une partie jusqu'à une victoire ou un match nul
    fils ← ∅
    if noeud.terminal ≠ VRAI then
        noeud.expand() ▷ générer les noeuds suivants directs
        ▷ vérifier parmi ces fils s'il y a un noeud gagnant pour le joueur qui a la main
        fils ← noeud.Gagnant(noeud.ia)
        if fils = ∅ then ▷ s'il n'y en a pas, en prendre un au hasard
            fils ← noeud.noeudsFils[hasard(nb(noeud.noeudsFils))]
        end if
        if fils.terminal ≠ VRAI then ▷ si le fils n'est pas terminal, poursuivre le jeu
            rollOut(fils)
        else ▷ sinon remonter la valeur vers la racine
            fils.updateValue(fils.sommeGains)
        end if
    end if
end procedure
  
```

MCTS : algorithme

MCTS : Créer des fils directs

```
procedure EXPAND
  feuille  $\leftarrow$  FAUX
  ▷ créer les fils noeuds directement atteignable à partir de la situation courante
  noeudsFils  $\leftarrow$  creerLesFils()
  for all fils  $\in$  noeudsFils do
    fils.value()
    ▷ valuer le fils
  end for
end procedure
```

MCTS : algorithme

MCTS : Remonter les valeurs

procedure UPDATEVALUE(*valeur*)

- ▷ **incremente le nb de visites et cumule la valeur au total**
- ▷ **demande aux ancêtres de faire de même**

$nbVisites \leftarrow nbVisites + 1$

$sommeGains \leftarrow sommeGains + valeur$

if $parent \neq \emptyset$ **then**

$parent.updateValue(valeur \times 0.9)$

- ▷ *(ajout d'un petit coef de réduction pour privilégier les victoires rapides)*

end if

end procedure

Adaptation de la méthode (MCTS)

Ajout des règles du jeu

- Seules les fonctions suivantes sont dépendantes du jeu :
 - **"creerLesFils()"** : crée des noeuds sous le noeud courant selon les actions possibles (*place un jeton, en retire, pose une carte, ... en fonction de la règle du jeu*).
un noeud avec $ia = VRAI$ crée des fils avec $ia = FAUX$ et inversement
 - **"value()"** : Evalue chaque noeud s'il est terminal (exemple $gagnant \leftarrow 1$, $perdant \leftarrow 0$, $partieNulle \leftarrow 0$)
 - **"noeudGagnant(typeJoueur)"** : vérifie si le noeud est un noeud terminal gagnant pour le joueur 'typeJoueur'... (4 pions alignés, roi pris, ...)

Utilisation de la méthode (MCTS)

Utilisation

- A partir du noeud vide initial, cas où l'IA joue en 1^{er} :
 - ① générer *nb* parties (*nb* = 10000 par exemple)
 - ② choisir le meilleur fils, et annoncer avoir joué son action
 - ③ si l'IA n'a pas gagné,
demander le jeu de l'adversaire
 - ④ récupérer le noeud correspondant à l'action choisie par
l'adversaire
si le noeud n'a pas été généré, lancer 'expand' à partir du
noeud courant pour le générer
 - ⑤ si l'Humain.e n'a pas gagné, retour au point 1

Recherche arborescente Monte Carlo (MCTS)

Remarques

- Code aussi générique que $\alpha - \beta$
- Valuation très rapide :
 - ne value que les noeuds terminaux,
 - pas d'heuristiques utilisés pour noeuds intermédiaires
- valuation pas nécessaire dans l'intervalle $[0, 1]$
 - dans l'exemple suivant, une perte a une valeur de -100, un gain, une valeur de 1, un match nul une valeur de 0

Exemple de Puissance 4 par MCTS I

noeud 0, level = 0, action 0

+-----+

+-----+

+-----+

+-----+

+-----+

+-----+

valeur=-219.81816045777066 / 309.0 visites

je choisis l'action 3

noeud 4, level = 1, action 3

+-----+

+-----+

+-----+

+-----+

+-----+

+__1__+

valeur=-209.43378760640923 / 287.0 visites

Exemple de Puissance 4 par MCTS II

Votre choix ?

5

votre jeu

noeud 524, level = 2, action 5

+ _ _ _ _ _ +

+ _ _ _ _ _ +

+ _ _ _ _ _ +

+ _ _ _ _ _ +

+ _ _ _ _ _ +

+ _ _ 1 2 _ +

valeur=-19.32626184202228 / 22.0 visites

je choisis l'action 4

noeud 530, level = 3, action 4

+ _ _ _ _ _ +

+ _ _ _ _ _ +

+ _ _ _ _ _ +

+ _ _ _ _ _ +

+ _ _ _ _ _ +

Exemple de Puissance 4 par MCTS III

```
+__112_+
valeur=-40.36342008438456 / 48.0 visites
```

```
-----
```

```
Votre choix ?
```

```
5
```

```
votre jeu
```

```
noeud 8567, level = 4, action 5
```

```
+______+
```

```
+______+
```

```
+______+
```

```
+______+
```

```
+____2_+
```

```
+__112_+
```

```
valeur=-4.0459827717359165 / 4.0 visites
```

```
-----
```

```
je choisis l'action 5
```

```
noeud 9470, level = 5, action 5
```

```
+______+
```

Exemple de Puissance 4 par MCTS IV

```

+-----+
+-----+
+-----1_+
+-----2_+
+___112_+
valeur=-43.782350665866815 / 48.0 visites

```

 Votre choix ?

6
 votre jeu
 noeud 9478, level = 6, action 6

```

+-----+
+-----+
+-----+
+-----1_+
+-----2_+
+___1122+
valeur=-10.882096454731442 / 8.0 visites

```

Exemple de Puissance 4 par MCTS V

```

je choisis l'action 1
noeud 42044, level = 7, action 1
+-----+
+-----+
+-----+
+-----1_+
+-----2_+
+_1_1122+
valeur=-11.718483007390398 / 16.0 visites

```

Votre choix ?

0

votre jeu

```

noeud 43511, level = 8, action 0

```

```

+-----+
+-----+
+-----+
+-----1_+
+-----2_+

```

Exemple de Puissance 4 par MCTS VI

```

+21_1122+
valeur=-2.2876792454961 / 1.0 visites
-----
je choisis l'action 2
noeud 46755, level = 9, action 2
+_____+
+_____+
+_____+
+_____1_+
+_____2_+
+2111122+
valeur=1.0 / 0.0 visites
-----
fin du jeu
j'ai gagne

```