Intelligence Artificielle

Algorithmes pour jeux

Emmanuel ADAM

Université Polytechnique des Hauts-De-France

UPHF/INSA HdF

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

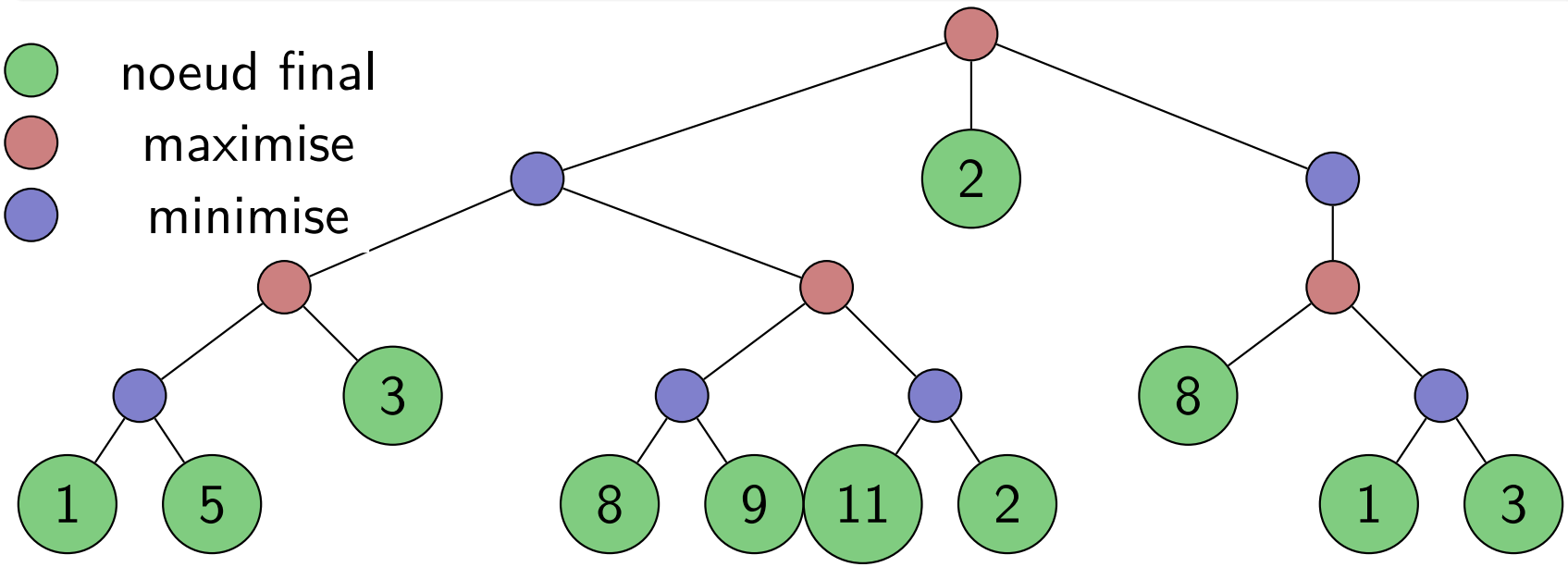
- Stratégie : profondeur limitée
- L'heuristique h évalue la qualité d'un noeud terminal (feuille ou de profondeur maximale)
- Comportement du joueur O_{rdi} :
  - O parcourt en profondeur et note les noeuds terminaux de niveau n d'une branche
  - O transmet les notes au noeud $n-1$ et sélectionne :
    - 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

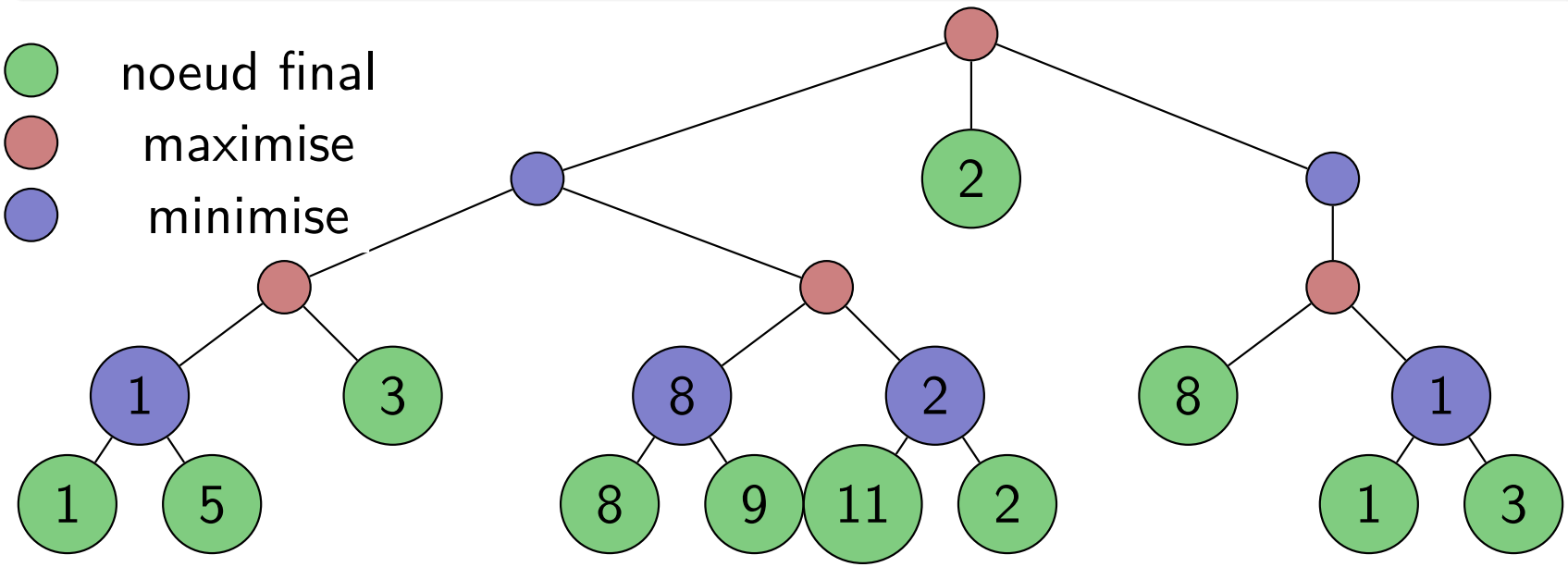


Exemple de MiniMax 2/4

Evaluation

Arbre présentant l'espace d'états pour O.

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

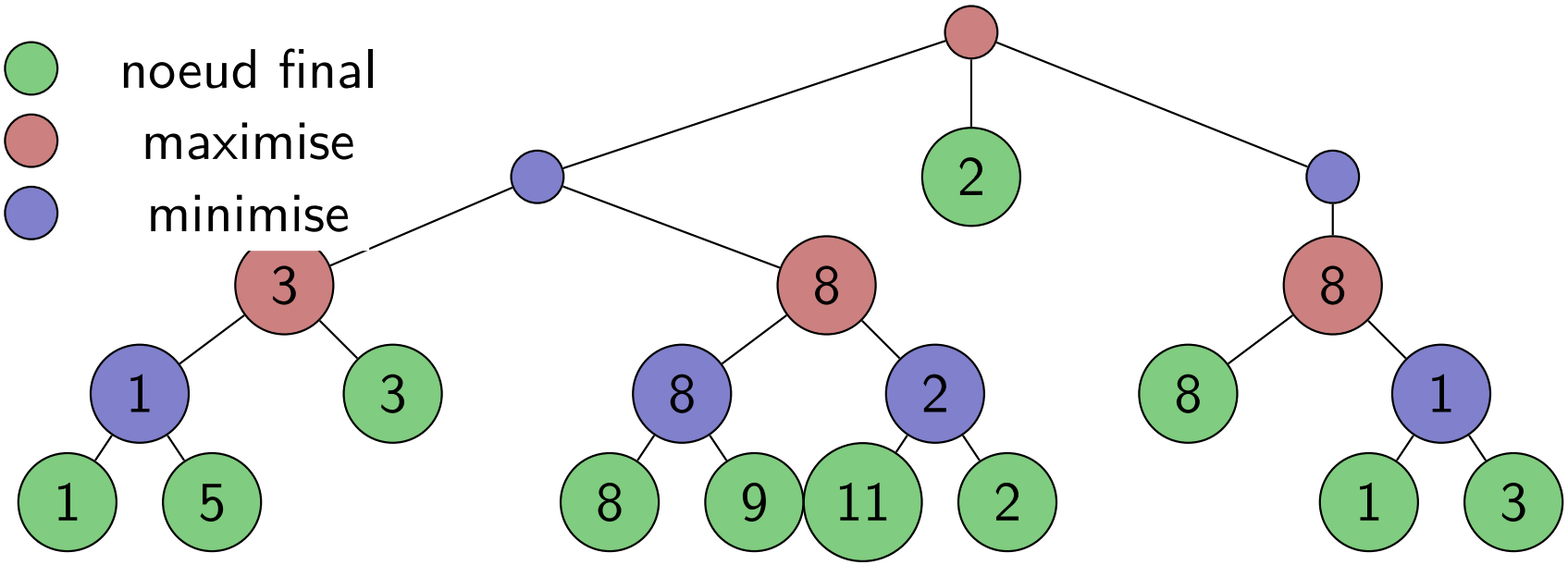


Exemple de MiniMax 3/4

Evaluation

Arbre présentant l'espace d'états pour O.

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

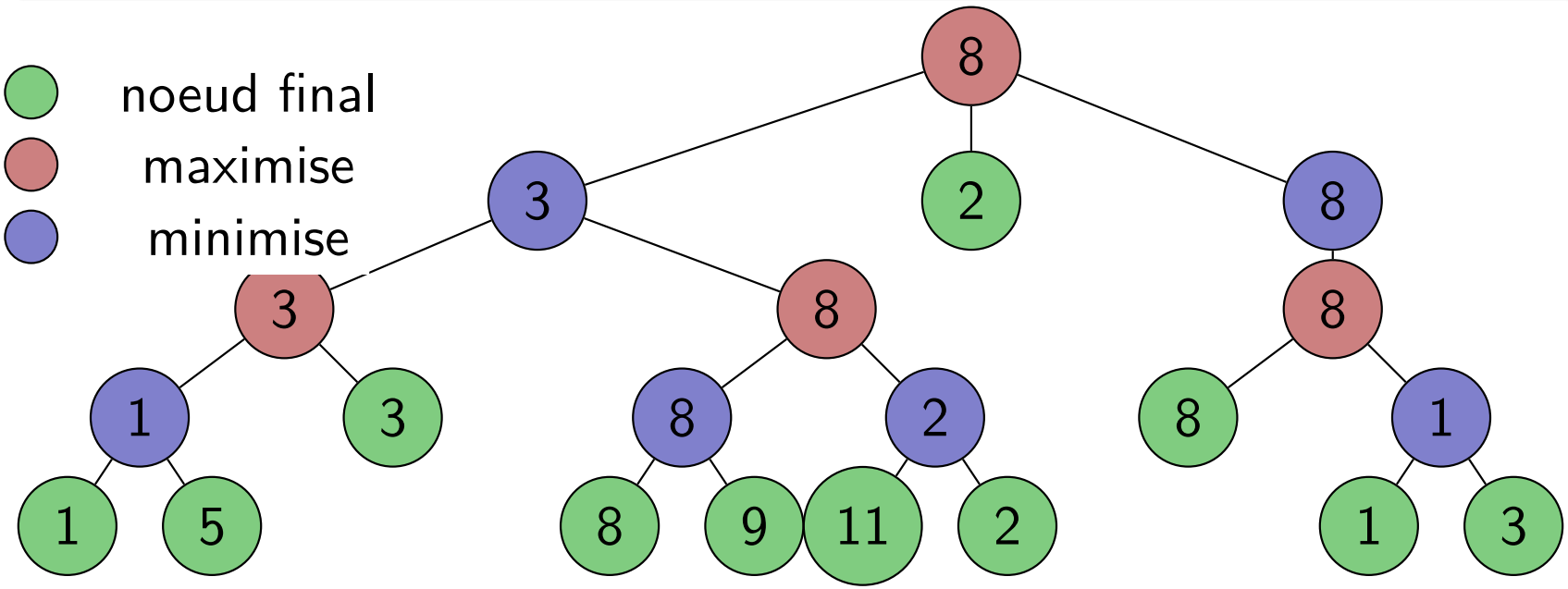


Exemple de MiniMax 4/4

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 ← −∞
        for all s′ ∈ s.successeurs do
            valSousArbre ← MINIMAX(s')
            if (minimax < valSousArbre) then
                minimax ← valSousArbre
            end if
        end for
        RETOURNER(minimax)
    end if
```

Exemple d'algorithme MiniMax II

```
if estMin(s) then
    minimax ← +∞
    for all s' ∈ s.successeurs do
        valSousArbre ← MiniMax(s')
        if (minimax > valSousArbre) then
            minimax ← valSousArbre
        end if
    end for
    retourner(minimax)
end if
end procedure
```

Exemple d'algorithme NegaMax I

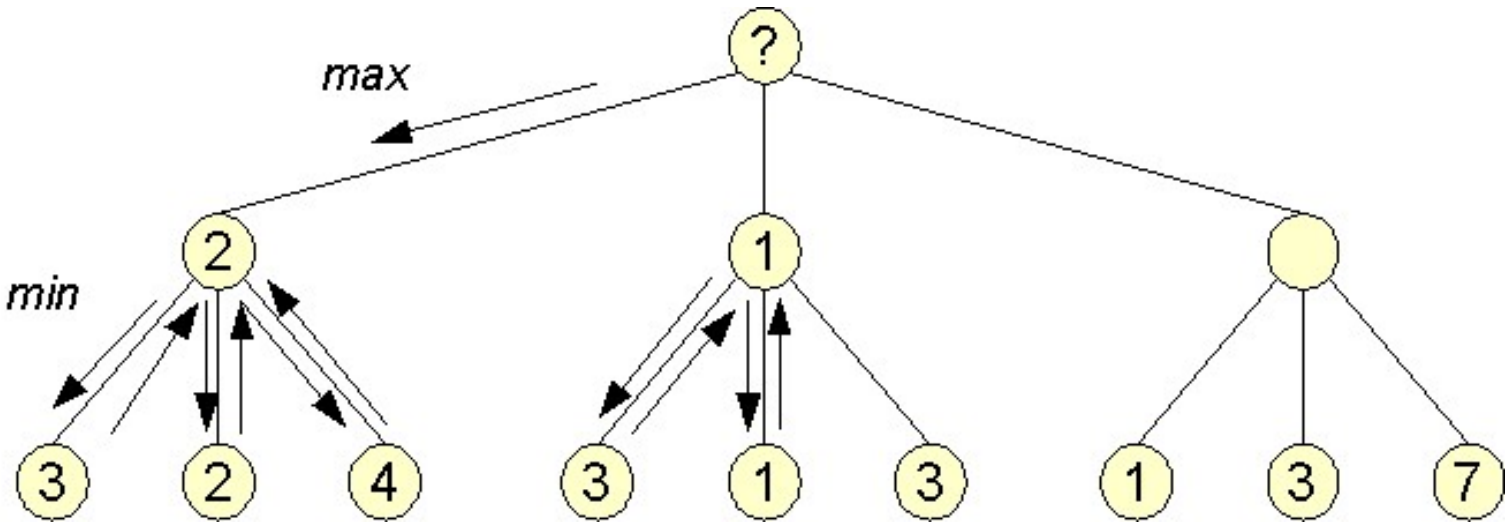
L'Algorithme du NegaMax est simplifié :
O prend le gain minimal de H en prenant l'oposé de son NegaMax et inversement

```
procedure NegaMax(s : situation)
    minimax; valSousArbre : entier ;
    if estFeuille(s) then
        retourner(h(s))
    end if
    minimax ← −∞
    for all s' ∈ s.successeurs do
        valSousArbre ← − NegaMax(s')
        if (minimax < valSousArbre) then
            minimax ← valSousArbre
        end if
    end for
    retourner(minimax)
end procedure
```

Algorithme Alpha - Beta

Algorithme Alpha - Beta

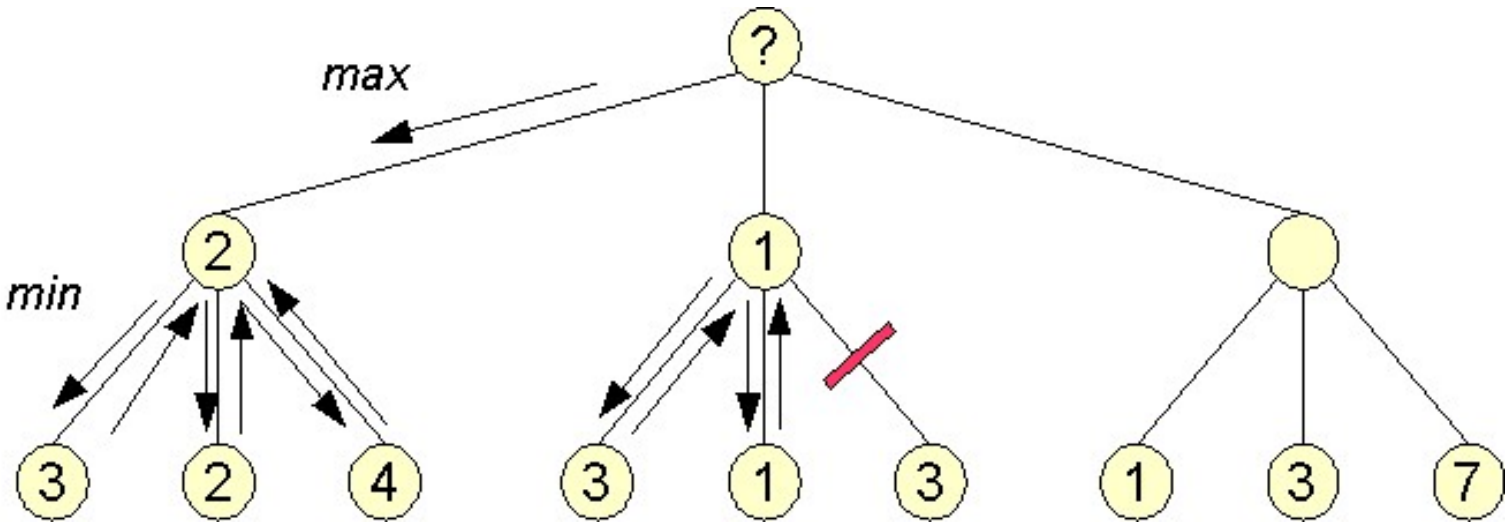
- Problème du MiniMax : Exploration complète de l'arbre, de l'espace d'états
- Possibilité d'élaguer des branches en fonction des découvertes



Algorithme Alpha - Beta

Algorithme Alpha - Beta

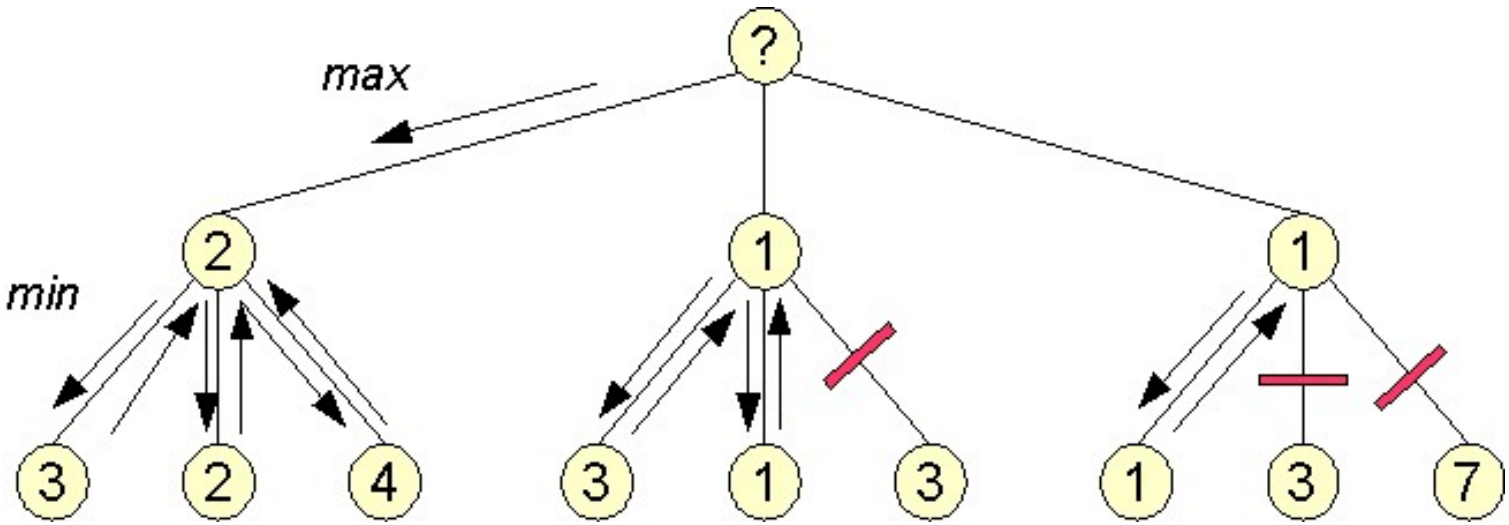
- Problème du MiniMax : Exploration complète de l'arbre, de l'espace d'états
- Possibilité d'élaguer des branches en fonction des découvertes



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, α : entier, β : entier)
    minimax; valSousArbre : entier ;
    if estFeuille(s) then retourner(h(s))
    end if

    if estMax(s) then
        minimax ← α
        for all s' ∈ s.successeurs do
            valSousArbre ← AlphaBeta(s', minimax, β)
            if (minimax < valSousArbre) then
                minimax ← valSousArbre
            end if
            if (minimax ≥ β) then retourner(minimax)
            end if
        end for
        retourner(minimax)
    end if
```

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

```
if estMax(s) then
    minimax ← β
    for all s' ∈ s.successeurs do
        valSousArbre ← AlphaBeta(s', α, minimax)
        if (minimax > valSousArbre) then
            minimax ← valSousArbre
        end if
        if (minimax ≤ α) 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, α : entier, β : 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(α)
      end if
      $\alpha \leftarrow max(\alpha, \beta)$
    end for
    retourner(α)
  end if
end procedure

Fonction d'évaluation

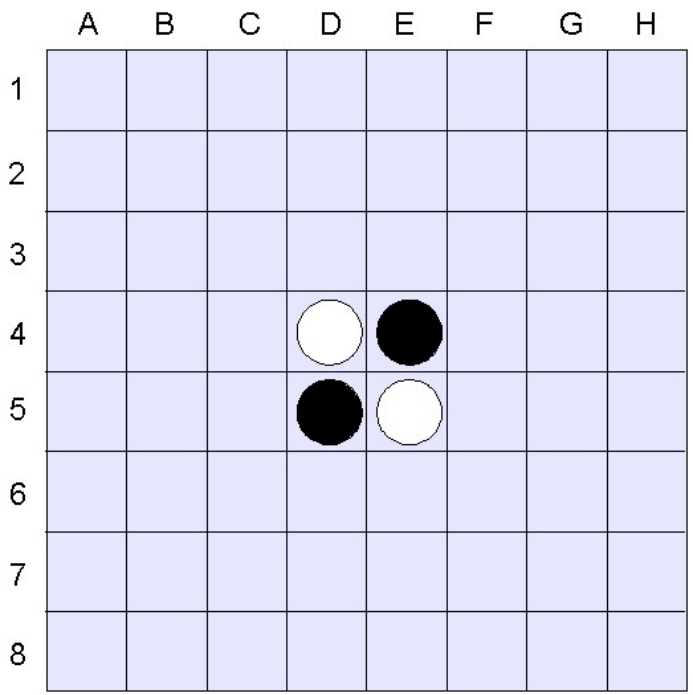
Heuristique

- 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, ...).
  - $h = p_{materiel} \times c_{materiel} + p_{mobilite} \times c_{mobilite} + p_{position} \times c_{position}$
- 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.



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

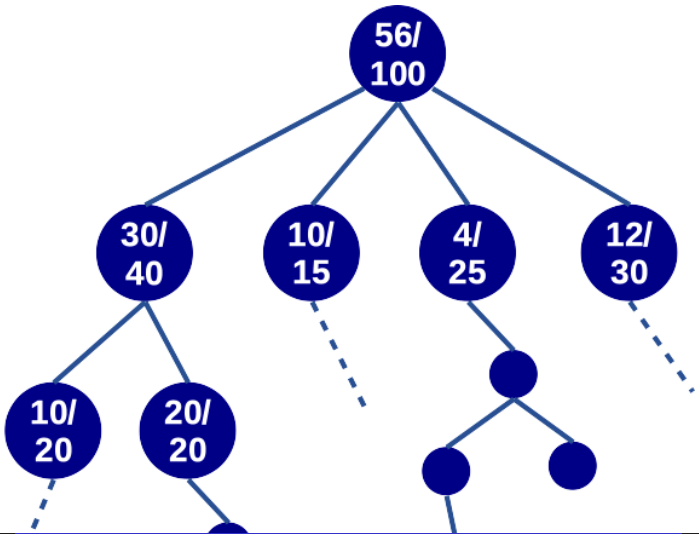
Stratégie

- Evolution de l'importance (du poids) des 3 critères en cours de jeu
  - debut : mobilité et position favorisées ($p_{mobilite}$ 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 : $p_{materiel}$ 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
  - $\rightarrow$ pas d'heuristique utilisée pour trier les noeuds et élaguer l'arbre
  - $\rightarrow$ 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, echec, go, ...)
- Style de jeu déduit de longue séries de parties simulées :
  - $\rightarrow$ le style est **propre** à l'IA, il n'imite pas celui d'experts humains

MCTS : algorithme

MCTS : Composition d'un noeud

Classe Noeud

feuille $\leftarrow$ *FAUX* ▷ *Vrai si noeud sans fils créé*

terminal $\leftarrow$ *FAUX* ▷ *Vrai si noeud sans autre action possible*

ia $\leftarrow$ *FAUX* ▷ *Vrai si noeud joue par l'IA*

noeudsFils $\leftarrow \emptyset$ ▷ *contient les noeuds fils directs*

nbVisites $\leftarrow 0$ ▷ *nb de passages par ce noeud*

sommeGains $\leftarrow 0$ ▷ *somme de gains obtenus par ce noeuds ou ses descendants*

parent $\leftarrow \emptyset$ ▷ *parent direct du noeud*

▷ *AJOUTER LES CHAMPS SPECIFIQUES AU JEU A GERER*

MCTS : algorithme

MCTS : Choisir et jouer

procedure SELECTACTION
▷ **à partir du noeud courant, descendre jusqu'à un noeud feuille**
terminal ← *VRAI*
noeud ← *noeudCourant*
while *noeud.feuille* ≠ *VRAI* **do**
noeud ← *noeud.select*()
end while
rollOut(*node*) ▷ **jouer une partie jusq'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**

$noeudChoisi \leftarrow \emptyset$

$meilleurValeur \leftarrow -\infty$

$c \leftarrow \sqrt{2}$

for all $fils \in noeudsFils$ **do**

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

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

else

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

end if

if $valeur \geq meilleurValeur$ **then**

$noeudChoisi \leftarrow child$

$meilleurValeur \leftarrow value$

end if

end for

$noeudChoisi.parent \leftarrow ceNoeud$

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
        ▷ verifier parmi ces fils s'il y a un noeud gagnant pour le joueur qui a la main
        fils ← noeudGagnant(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 ← *FAUX*
    ▷ créér les fils noeuds directement atteignable à partir de la situation courante
    noeudsFils ← *creerLesFils*()
    for all *fils* ∈ *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 ← *nbVisites* + 1
sommeGains ← *sommeGains* + *valeur*
if *parent* ≠ ∅ **then**
parent.*updateValue*(*valeur* × 0.9)
▷ *(ajout d'un petit coef de réduction pour priviégier 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'intervale $[0, 1]$
  - 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=-4.045982771735916 5 / 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
```