Comment faire jouer un programme à un jeu de société?

Auteur: Julien David

Dans ce cours, on s'intéresse aux jeux où:

- deux adversaires s'affrontent,
- il ne peut y avoir qu'un seul gagnant ou une seule gagnante,
- les adversaires jouent tour à tour, une seule fois par tour.

Commençons par un jeu facile

Problème Le jeu de Nim

On dispose n allumettes devant deux adversaires.

- tour à tour, les adversaires prennent 1, 2, ou 3 allumettes.
- la personne qui ramasse la dernière allumette a perdu.

L'espace des possibles

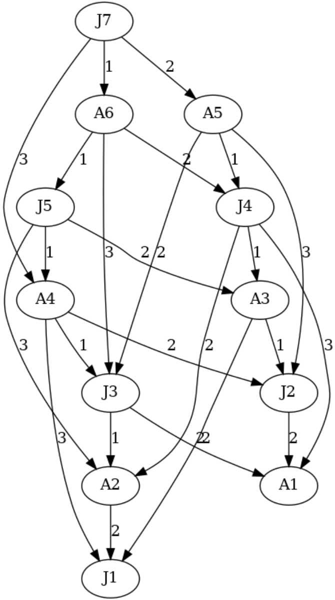
Chaque jeu possède un ensemble de configurations possibles, que l'on note $\mathcal{C}$.

Explication Espace des possibles au jeu de nim

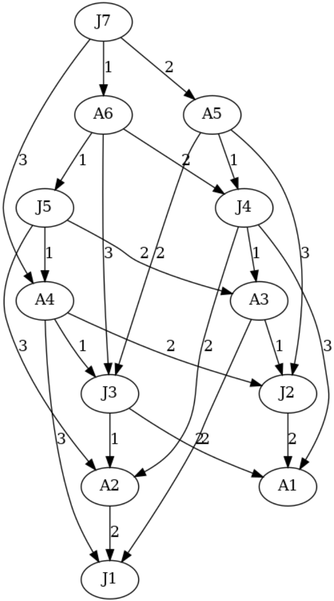
- Pour le jeu de nim, on part avec n allumettes et une joueuse désignée.
- Pour chaque valeur de n, chacun des deux adversaires peut arriver à cette configuration[a].
- Il existe au plus $2n$ configurations possibles.

[a] à exception près mais elles deviennent négligeables quand n grandit

Graphe des parties possibles: exemple avec $n = 7$



Graphe des parties possibles: exemple avec $n = 7$



- La joueuse va chercher un chemin qui mène vers $A1$.
- Un jeu peut donc se résumer à des problèmes de chemins dans des graphes.

Graphe des parties possibles: Construction?

Problème Construction du graphe?

- Comment construire efficacement ce graphe?
- Peut-il tenir en mémoire?
- la méthode est-elle générique ou spécifique au jeu?

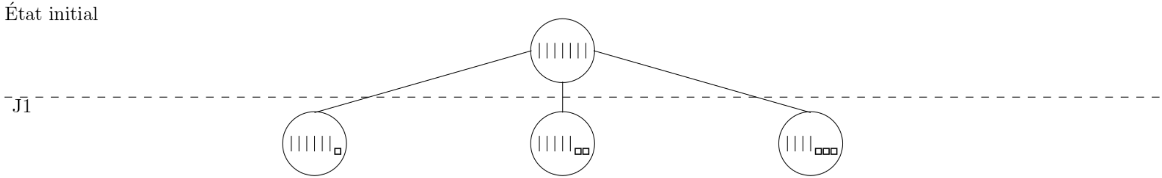
Arbre des parties possibles

Explication | Méthode générique

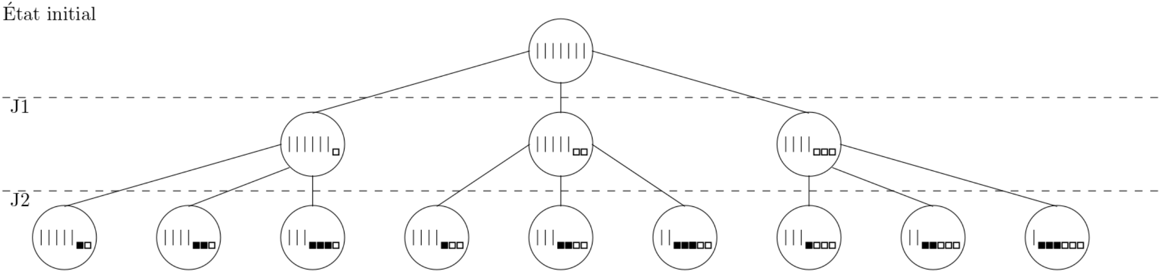
- si, à partir d'une configuration légale c du jeu, on connaît tous les coups autorisés,
- on peut calculer toutes les nouvelles configurations atteignables à partir de c.
- La structure ainsi obtenue est un arbre[a].

[a] il est possible dans certains cas de faire un graphe en utilisant la memoization

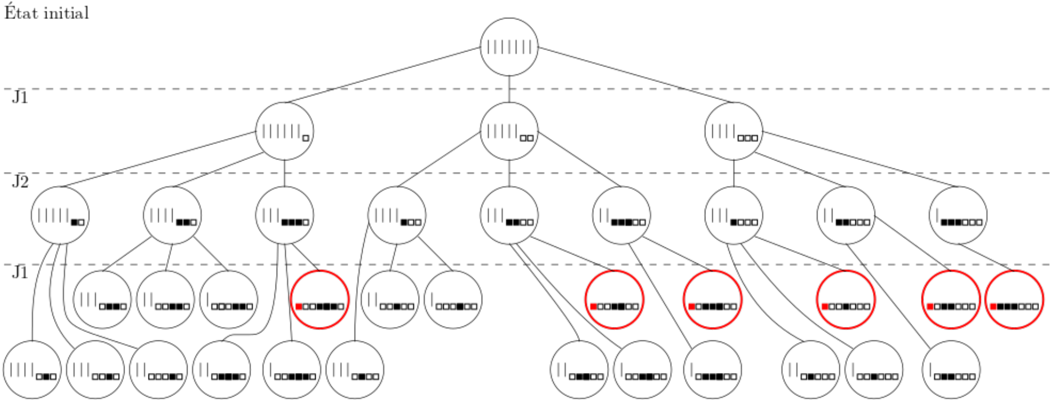
Arbres des parties possibles: exemple avec $n = 7$



Arbres des parties possibles: exemple avec $n = 7$



Arbres des parties possibles: exemple avec $n = 7$



L'espace des possibles

Chaque jeu possède un ensemble de configurations possibles, que l'on note $\mathcal{C}$.

Explication Nombre de parties possibles jeu de nim

- Pour le jeu de nim, on part avec n allumettes et on peut en retirer $1, 2$ ou 3
- Le nombre de parties possibles est donc donné par la formule

$$p(n) = \begin{cases} 1, & \text{si } n \in \{0, 1\} \\ 2, & \text{si } n = 2 \\ p(n-1) + p(n-2) + p(n-3), & \text{sinon} \end{cases}$$

L'espace des possibles

Chaque jeu possède un ensemble de configurations possibles, que l'on note $\mathcal{C}$.

Explication Nombre de parties possibles jeu de nim

- Pour le jeu de nim, on part avec n allumettes et on peut en retirer $1, 2$ ou 3
- Le nombre de parties possibles est donc donné par la formule

$$p(n) = \begin{cases} 1, \text{ si } n \in \{0, 1\} \\ 2, \text{ si } n = 2 \\ p(n-1) + p(n-2) + p(n-3), \text{ sinon} \end{cases}$$

- Premières valeurs:

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
p(n)	1	1	2	4	7	13	24	44	81	149	274	504	927	1705	3136

Aussi appellée **Suite de Tribonacci**

L'espace des possibles

Chaque jeu possède un ensemble de configurations possibles, que l'on note $\mathcal{C}$.

Explication — Nombre de noeuds dans l'arbre jeu de nim

- Pour le jeu de nim, on part avec n allumettes et on peut en retirer $1, 2$ ou 3
- Le nombre de noeuds est donné par la formule

$$s(n) = \begin{cases} 1, & \text{si } n = 0 \\ 2, & \text{si } n = 1 \\ 4, & \text{si } n = 2 \\ 1 + s(n-1) + s(n-2) + s(n-3), & \text{sinon} \end{cases}$$

L'espace des possibles

Chaque jeu possède un ensemble de configurations possibles, que l'on note $\mathcal{C}$.

Explication — Nombre de noeuds dans l'arbre jeu de nim

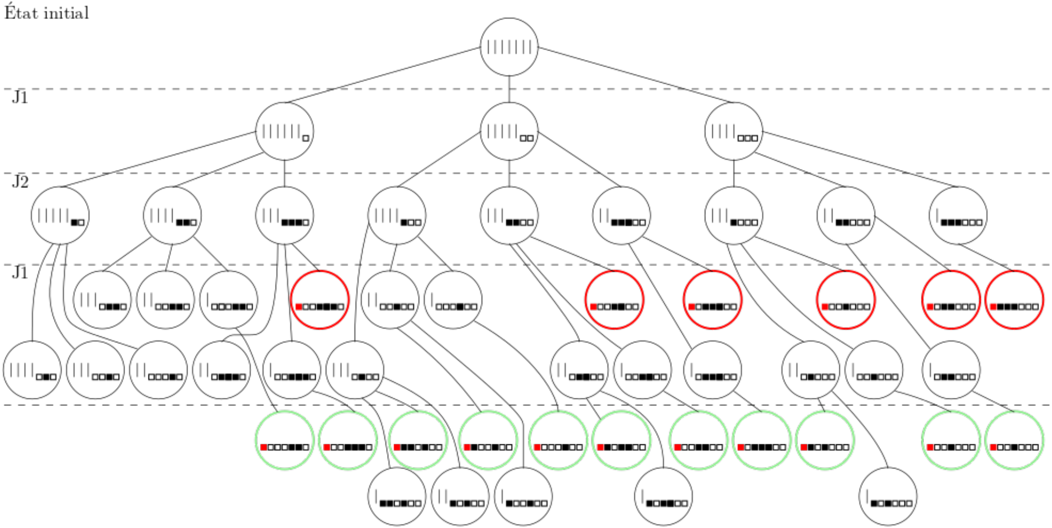
- Pour le jeu de nim, on part avec n allumettes et on peut en retirer $1, 2$ ou 3
- Le nombre de noeuds est donné par la formule

$$s(n) = \begin{cases} 1, & \text{si } n = 0 \\ 2, & \text{si } n = 1 \\ 4, & \text{si } n = 2 \\ 1 + s(n-1) + s(n-2) + s(n-3), & \text{sinon} \end{cases}$$

- Premières valeurs:

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
s(n)	1	2	4	8	15	28	52	96	177	326	600	1104	2031	3736	6872

Arbres des parties possibles: exemple avec $n = 7$



ça ressemble à du backtracking

Problème — mais ça n'est pas du backtracking classique

Ici, on doit tenir compte des coups joués par l'adversaire.

- On veut jouer le coup qui permet d'obtenir le meilleur **score** possible.

ça ressemble à du backtracking

Problème mais ça n'est pas du backtracking classique

Ici, on doit tenir compte des coups joués par l'adversaire.

- On veut jouer le coup qui permet d'obtenir le meilleur **score** possible.
- On part du principe que l'adversaire va jouer le meilleur possible dans son interêt et donc jouer le coup qui lui assure également le meileure score possible.

ça ressemble à du backtracking

Problème mais ça n'est pas du backtracking classique

Ici, on doit tenir compte des coups joués par l'adversaire.

- On veut jouer le coup qui permet d'obtenir le meilleur **score** possible.
- On part du principe que l'adversaire va jouer le meilleur possible dans son interêt et donc jouer le coup qui lui assure également le meileure score possible.
- Le meilleur score de l'adversaire est le moins bon score pour nous.

ça ressemble à du backtracking

Problème mais ça n'est pas du backtracking classique

Ici, on doit tenir compte des coups joués par l'adversaire.

- On veut jouer le coup qui permet d'obtenir le meilleur **score** possible.
- On part du principe que l'adversaire va jouer le meilleur possible dans son interêt et donc jouer le coup qui lui assure également le meileure score possible.
- Le meilleur score de l'adversaire est le moins bon score pour nous.
- On veut donc maximiser le score parmi les scores minimum que l'adversaire va nous permettre d'atteindre: on parle d'un **minimax**, ou **MinMax**.

Nos besoins

Remarque De quoi a-t-on besoin?

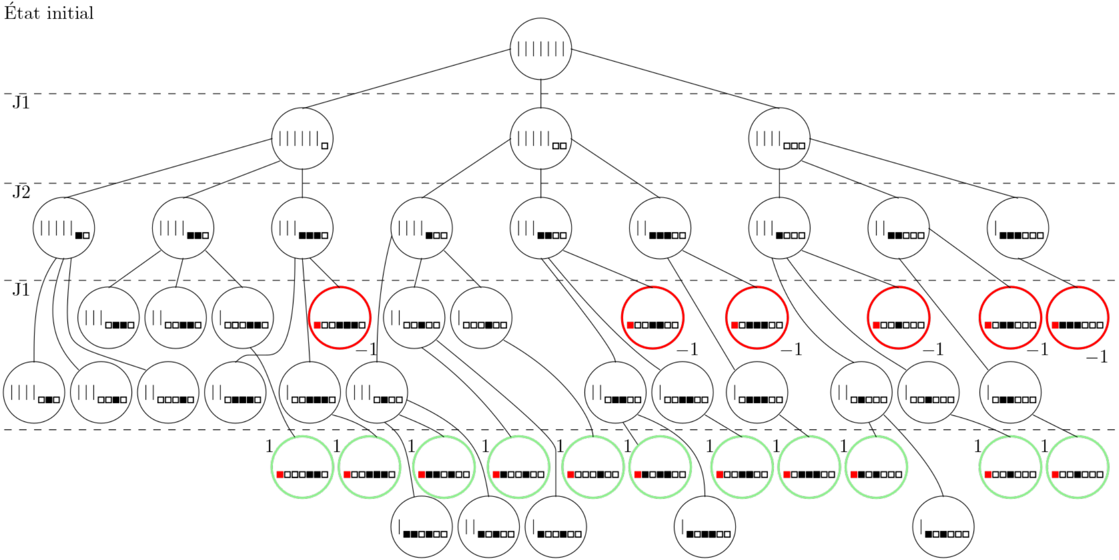
- Une fonction qui permet de déterminer un score
  - Dans un premier temps, on peut juste vouloir renvoyer 1 quand la Joueuse a gagné -1 quand c'est son Adversaire.

Nos besoins

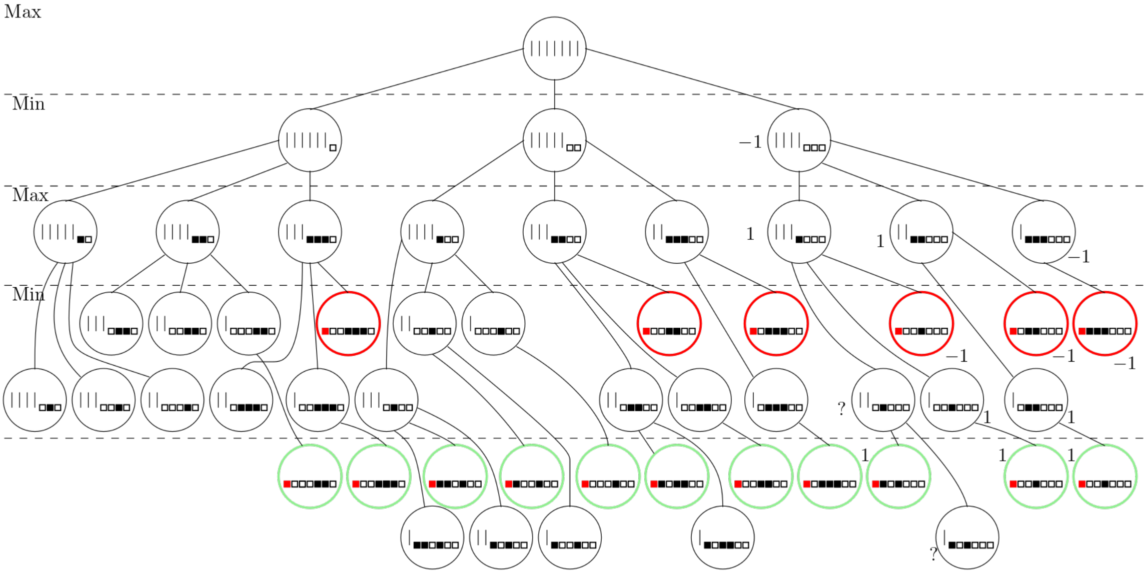
Remarque De quoi a-t-on besoin?

- Une fonction qui permet de déterminer un score
  - Dans un premier temps, on peut juste vouloir renvoyer 1 quand la Joueuse a gagné -1 quand c'est son Adversaire.
- savoir qui est en train de jouer.
  - on va marquer les noeuds de l'arbre comme étant ceux de la Joueuse ou de son Adversaire.
  - quand c'est à la joueuse de joueur, on cherche à maximiser le score.
  - quand c'est à l'adversaire, on cherche à le minimiser.

Arbres des parties possibles: exemple avec $n = 7$



Arbres des parties possibles: exemple avec $n = 7$



Le principe du minimax

Algorithme : $Minimax(C, score, J)$

Entrées : Une configuration de jeu $C \in \mathcal{C}$, une fonction $score : \mathcal{C} \mapsto \{-1, 0, 1\}$, $J \in \{JOUEUSE, ADVERSAIRE\}$
Sortie : Le meilleur coup à jouer et le score associé

```
si C ne contient plus de coups possibles alors
    Retourn (None, score(C));
fin
(res, meilleur) ← (None, faux_score);
pour tous les coups possibles c faire
    C' ← On joue le coup c dans C;
    (tmp, s) ← Minimax(C', score, changerJoueuse(J)) ;
    si J == JOUEUSE et s > meilleur alors
        (res, meilleur) ← (c, s);
    fin
    si J == ADVERSAIRE et s < meilleur alors
        (res, meilleur) ← (c, s);
    fin
fin
return (res, meilleur)
```

Le NégaMax

Du minimax au NegaMax

Remarque Simplification possible

- Si pour toute position légale possible p, on peut garantir que

$$score(p, JOUEUSE) = -score(p, ADVERSAIRE)$$

- Alors il et possible de simplifier l'écriture du Minimax, pour ne pas avoir besoin de tester le joueur en train de jouer.
- il suffit de toujours prendre le **max** de la **négation** du meilleur score de l'autre joueuse.
- **un NegaMax**

Negamax

Algorithme : $NegaMax(C, score, J)$

Entrées : Une configuration de jeu $C \in \mathcal{C}$, une fonction $score : \mathcal{C} \times \{JOUEUSE, ADVERSAIRE\} \mapsto \{-1, 0, 1\}$,
$J \in \{JOUEUSE, ADVERSAIRE\}$

Sortie : Le meilleur coup à jouer et le score associé

si C ne contient plus de coups possibles **alors**
    Retourn $(None, score(C, J))$;
fin
$(res, meilleur) \leftarrow (None, -2)$;
pour tous les coups possibles c **faire**
    $C' \leftarrow$ On joue le coup c dans C;
    $(tmp, s) \leftarrow NegaMax(C', score, changerJoueuse(J))$;
    si $-s > meilleur$ **alors**
        $(res, meilleur) \leftarrow (c, -s)$;
    fin
fin
return $(res, meilleur)$

Exercice

Algorithme : $NegaMax(C, score, J)$

Entrées : Une configuration de jeu $C \in \mathcal{C}$, une fonction $score : \mathcal{C} \times \{JOUEUSE, ADVERSAIRE\} \mapsto \{-1, 0, 1\}$, $J \in \{JOUEUSE, ADVERSAIRE\}$
Sortie : Le meilleur coup à jouer et le score associé

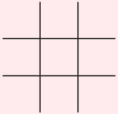
```
si C ne contient plus de coups possibles alors
    Retourn (None, score(C, J));
fin
(res, meilleur) ← (None, −2);
pour tous les coups possibles c faire
    C′ ← On joue le coup c dans C;
    (tmp, s) ←
      NegaMax(C′, score, changerJoueuse(J)) ;
    si −s > meilleur alors
        (res, meilleur) ← (c, −s);
    fin
fin
return (res, meilleur)
```

Question Quel est le résultat $n = 5$?

Appliquez l'algorithme du negamax sur le jeu de nim en supposant qu'on commence avec 5 allumettes.

Autre exemple facile

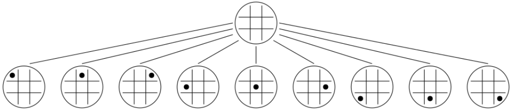
Problème Le morpion



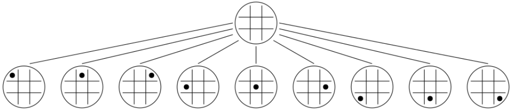
L'objectif est d'aligner trois CROIX, ou trois ROND, dans un des sens suivant:

- horizontal
- vertical
- diagonal

Espace des possibles



Espace des possibles



Remarque Difficile de tout dessiner...

$9 \times 8 = 72$ états possibles pour le deuxième coup.

Configurations possibles

Chaque jeu possède un ensemble de configurations possibles, que l'on note $\mathcal{C}$.

Explication | Configurations possibles au morpion

- Pour le morpion, chaque case peut être égale à VIDE, ROND ou CROIX
- La différence entre le nombre de ROND et de CROIX est au plus de 1
- Le nombre de configurations est inférieur à

$$\sum_{i=1}^{4} \binom{9}{i, i, 9-2i} + \sum_{i=1}^{5} \binom{9}{i, i-1, 9-2i+1} = 6045$$

- C'est une surestimation: certaines configurations ne sont pas légales

X	X	X
O	O	O

Parties possibles

Explication Parties possibles au morpion

- Pour le morpion, les 9 positions vont être jouées dans n'importe quel ordre
- Le nombre de parties est inférieur à $9! = 362880$
- C'est une surestimation: certaines parties s'arrêtent avant d'avoir joué 9 coups.

Parties possibles

Explication Taille de l'arbre au morpion

- le nombre de noeuds au niveau i de l'arbre est un arrangement de i parmi 9.
- Le nombre de noeuds est inférieur à $\sum_{i=0}^{9} \frac{9!}{(9-i)!} = 986410$
- C'est une surestimation: certaines parties s'arrêtent avant d'avoir joué 9 coups, on a en réalité besoin de calculer 576285 noeuds.

L'espace des possibles

Chaque jeu possède un ensemble de configurations possibles, que l'on note $\mathcal{C}$.

Explication — Espace des possibles du jeu d'échec

- En 1950, John Von Neumann estime le nombre de partie d'échecs à 10^{120}.
- les estimations récentes sont plus proches 10^{123}.
- le nombre de configurations légales est estimé à $4,8 \times 10^{44}$

Jeu de go

Explication | Jeu de Go

- Le nombre exact configurations légales est (Tromp, 2016): $2081681993819799846994786333448627702865224538$ $8453054842563945682092741961273801537852564845$ $1698519643907259916015628128546089888314427129$ $715319317557736620397247064840935$ soit supérieur à 10^{170}

Jeu de go

Explication | Jeu de Go

- Le nombre exact configurations légales est (Tromp, 2016):
  $208168199381979984699478633344862770286522453884530548425639456820927419612738015378525648451698519643907259916015628128546089888314427129715319317557736620397247064840935$
  soit supérieur à 10^{170}
- Le nombre de parties est supérieur à $10^{10^{100}}$.

Élagage

Definition L'élagage

- L'élagage signifie littéralement couper certaines branches d'un arbre afin d'orienter ou limiter son développement.
- L'idée est donc de ne pas calculer, voir recalculer, toutes les configurations d'un jeu afin de prendre une décision.

Élagage par profondeur bornée

Definition — L'élagage par profondeur bornée

- Il est possible de limiter le nombre de coups que l'on veut prédire à l'avance.
- Par exemple, si on décide de prédire 4 coups à l'avance, l'arbre sera au maximum de hauteur 4.

Explication — Avantage

- C'est donc rapide à calculer.
- On peut créer des niveaux de difficultés dans le jeu en réglant la hauteur de l'arbre.

Élagage par profondeur bornée

Definition L'élagage par profondeur bornée

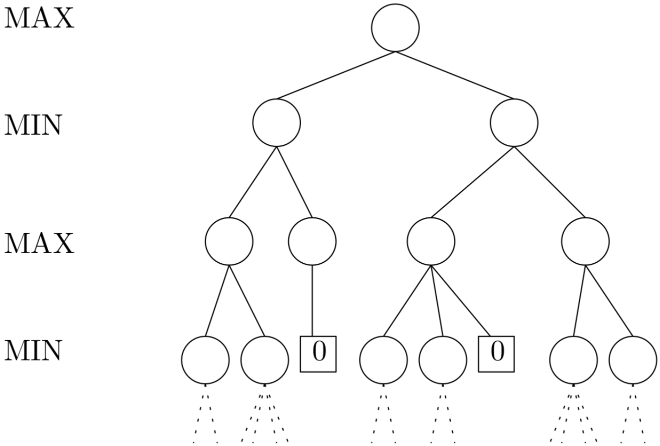
- Il est possible de limiter le nombre de coups que l'on veut prédire à l'avance.
- Par exemple, si on décide de prédire 4 coups à l'avance, l'arbre sera au maximum de hauteur 4.

Problème

- Il faut écrire des fonctions de score plus complexe, qui nécessitent de bien connaître le jeu.

Élagage par profondeur bornée

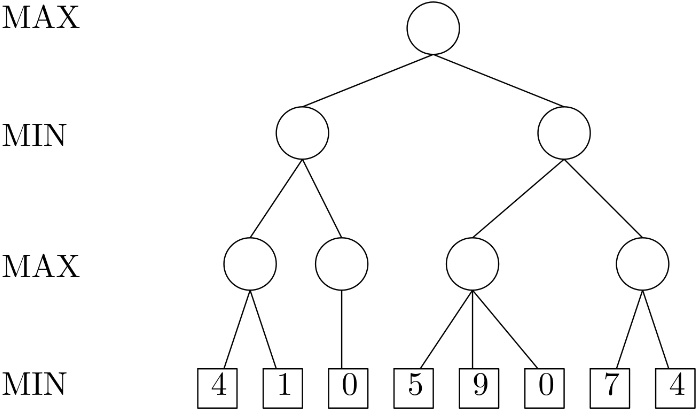
On imagine un jeu imaginaire où on possède une fonction de score efficace.



On décide de d'élaguer à prodondeur 3

Élagage par profondeur bornée

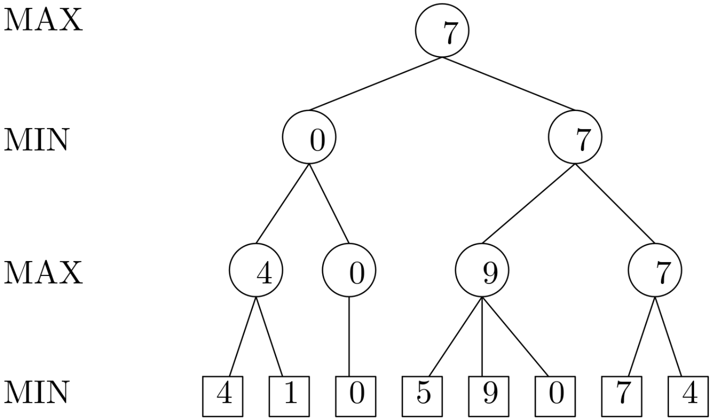
On imagine un jeu imaginaire où on possède une fonction de score efficace.



On évalue tous les noeuds de cette profondeur à l'aide de la fonction score.

Élagage par profondeur bornée

On imagine un jeu imaginaire où on possède une fonction de score efficace.



On applique le minimax

Minimax avec élagage

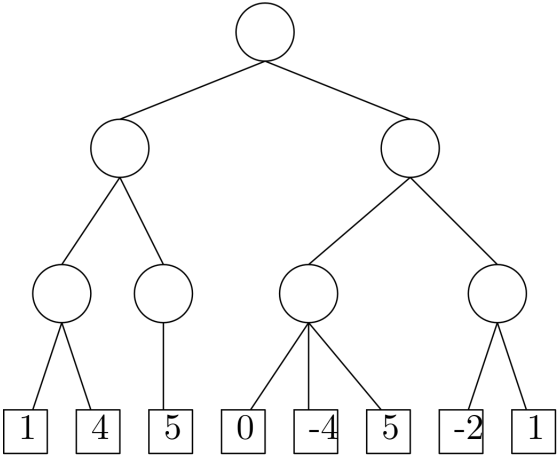
Algorithme : $Minimax(C, score, J)$

Entrées : Une configuration de jeu $C \in \mathcal{C}$, une fonction $score : \mathcal{C} \mapsto \{-1, 0, 1\}$, $J \in \{JOUEUSE, ADVERSAIRE\}$, une profondeur p
Sortie : Le meilleur coup à jouer et le score associé

si C ne contient plus de coups possibles ou p vaut 0 **alors**
  Retourn $(None, score(C))$;
fin
$(res, meilleur) \leftarrow (None, faux\_score)$;
pour tous les coups possibles c **faire**
  $C' \leftarrow$ On joue le coup c dans C;
  $(tmp, s) \leftarrow Minimax(C', score, changerJoueuse(J), p - 1)$;
  si $J == JOUEUSE$ et $s > meilleur$ **alors**
    $(res, meilleur) \leftarrow (c, s)$;
  fin
  si $J == ADVERSAIRE$ et $s < meilleur$ **alors**
    $(res, meilleur) \leftarrow (c, s)$;
  fin
fin
return $(res, meilleur)$

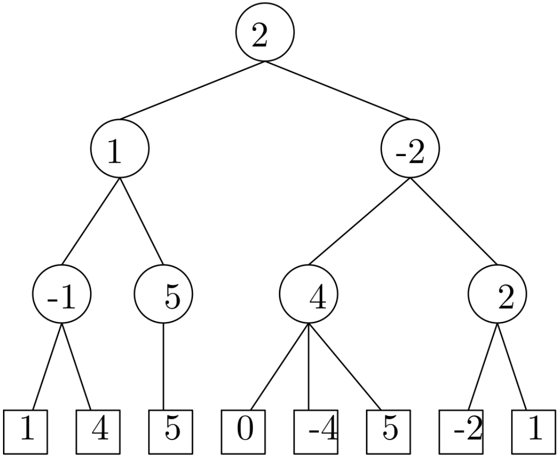
Élagage par profondeur bornée: négamax

On imagine un jeu imaginaire où on possède une fonction de score efficace.



Élagage par profondeur bornée: négamax

On imagine un jeu imaginaire où on possède une fonction de score efficace.



Negamax

Algorithme : $NegaMax(C, score, J)$

Entrées : Une configuration de jeu $C \in \mathcal{C}$, une fonction $score : \mathcal{C} \times \{JOUEUSE, ADVERSAIRE\} \mapsto \{-1, 0, 1\}$,
$J \in \{JOUEUSE, ADVERSAIRE\}$, une profondeur p

Sortie : Le meilleur coup à jouer et le score associé

si C ne contient plus de coups possibles ou p vaut 0 **alors**
    Retourn $(None, score(C, J))$;
fin
$(res, meilleur) \leftarrow (None, -2)$;
pour tous les coups possibles c **faire**
    $C' \leftarrow$ On joue le coup c dans C;
    $(tmp, s) \leftarrow NegaMax(C', score, changerJoueuse(J), p - 1)$;
    si $-s > meilleur$ **alors**
        $(res, meilleur) \leftarrow (c, -s)$;
    fin
fin
return $(res, meilleur)$

Élagage $\alpha\beta$

L'élagage $\alpha\beta$

Explication Comment ça marche?

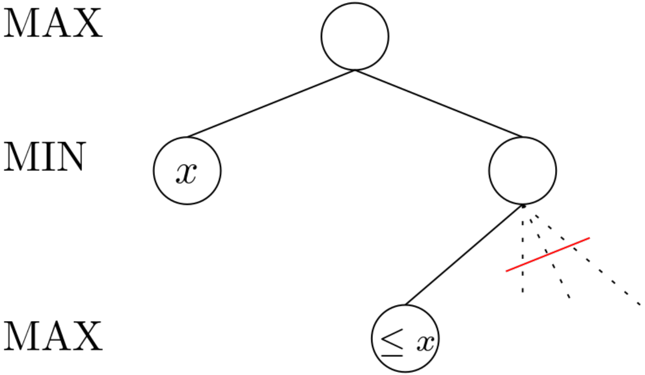
- Quand on éxécute l'algorithme du minimax, cela revient à faire un parcours préfixe de l'arbre des parties possibles.
- L'idée est de se servir des notions de Min et Max pour éviter que calculer des sous-arbres

L'élagage $\alpha\beta$

Explication | Coupure α

- Si sur un noeud **Max**, on a déjà calculé le score x d'un des enfants.
- Les enfants suivants sont donc des noeuds **Min** et les petits-enfants des **Max**.
- Si l'un des petits-enfants est de valeur $y \leq x$, on sait que la valeur renvoyée par son parent (un noeud **Min**) est inférieure ou égale à y.
- Or on a déjà une meilleure solution x, il est donc inutile de continuer cette partie du calcul.

L'élagage $\alpha\beta$



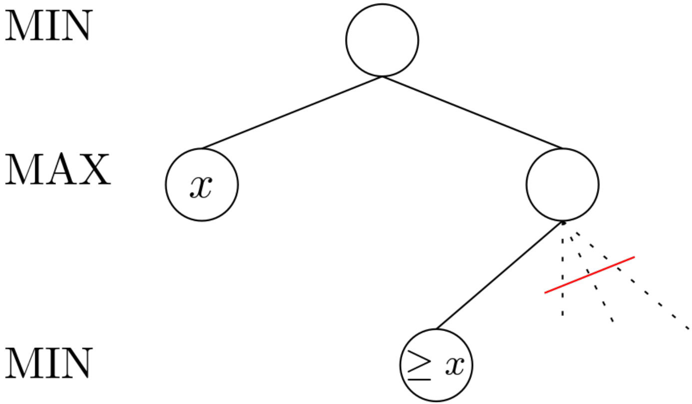
Coupure α

L'élagage $\alpha\beta$

Explication | Coupure β

- Si sur un noeud **Min**, on a déjà calculé le score x d'un des enfants.
- Les enfants suivants sont donc des noeuds **Max** et les petits-enfants des **Min**.
- Si l'un des petits-enfants est de valeur $y \geq x$, on sait que la valeur renvoyée par son parent (un noeud **Max**) est supérieure ou égale à y.
- Or on a déjà une meilleure solution x, il est donc inutile de continuer cette partie du calcul.

L'élagage $\alpha\beta$



Coupure β

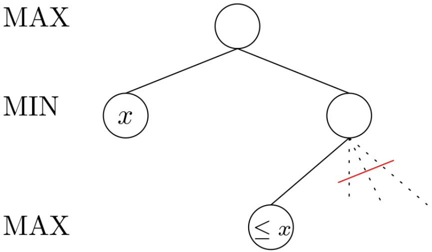
L'élagage $\alpha\beta$ pour le minimax

Algorithme : $Minimax(C, score, J)$

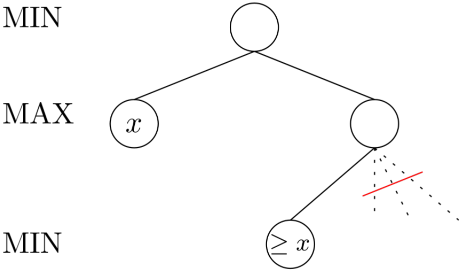
Entrées : Une configuration de jeu $C \in \mathcal{C}$, une fonction $score : \mathcal{C} \mapsto \{-1, 0, 1\}$, $J \in \{JOUEUSE, ADVERSAIRE\}$, deux valeurs $\alpha < \beta$
Sortie : Le meilleur coup à jouer et le score associé

si C ne contient plus de coups possibles **alors**
    Retour $(None, score(C))$;
fin
$(res, meilleur) \leftarrow (None, faux\_score)$;
pour tous les coups possibles c **faire**
    $C' \leftarrow$ On joue le coup c dans C;
    $(tmp, s) \leftarrow AlphaBeta(C', score, changerJoueuse(J), \alpha, \beta)$;
    si $J == JOUEUSE$ **alors**
        si $s \geq \beta$ **alors**
            return (c, s)
        fin
        si $s > meilleur$ **alors**
            $(res, meilleur) \leftarrow (c, s)$;
        fin
        $\alpha \leftarrow max(\alpha, s)$;
    fin
    si $J == ADVERSAIRE$ **alors**
        si $s \leq \alpha$ **alors**
            return (c, s)
        fin
        si $s < meilleur$ **alors**
            $(res, meilleur) \leftarrow (c, s)$;
        fin
        $\beta \leftarrow min(\beta, s)$;
    fin
fin
return $(res, meilleur)$

L'élagage $\alpha\beta$



Coupure α

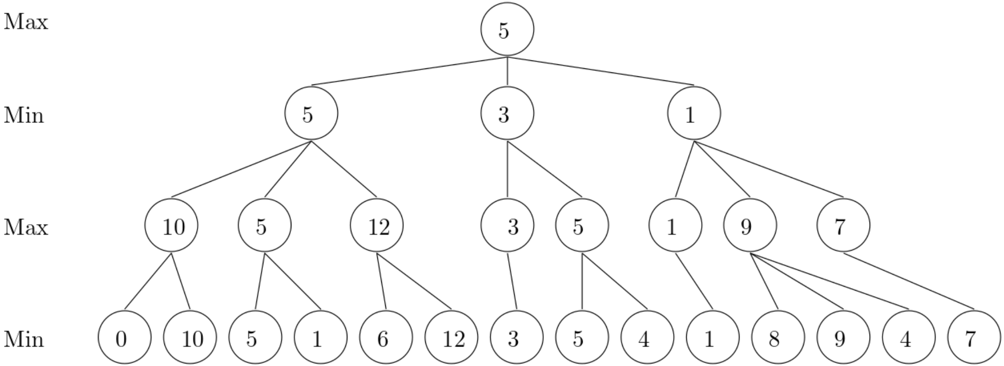


Coupure β

On effectue les coupures α sur les niveaux **Min** (l'adversaire) et les coupures β sur les niveaux **Max** (joueuse).

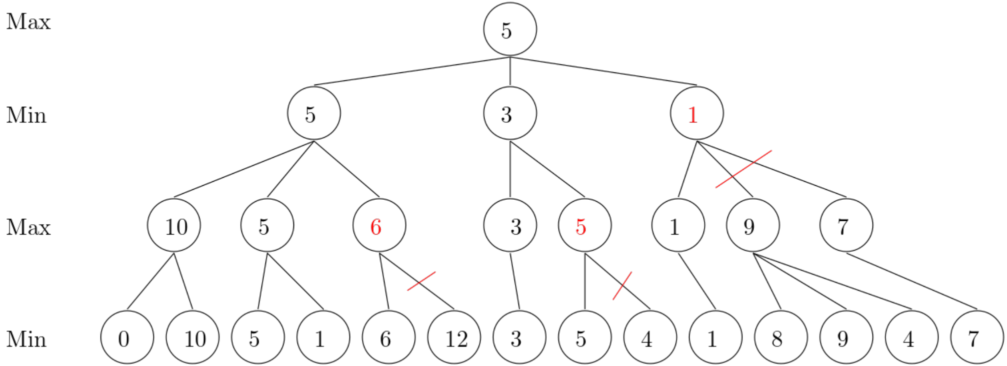
Exercice

Élaguez l'arbre suivant à avec l'élagage $\alpha\beta$



Exercice

Élaguez l'arbre suivant à avec l'élagage $\alpha\beta$



Élagage $\alpha\beta$ pour le Negamax

Algorithme : $AlphaBeta(C, score, J, \alpha, \beta)$

Entrées : Une configuration de jeu $C \in \mathcal{C}$, une fonction $score : \mathcal{C} \times \{JOUEUSE, ADVERSAIRE\} \mapsto \{-1, 0, 1\}$, $J \in \{JOUEUSE, ADVERSAIRE\}$, deux valeurs $\alpha < \beta$

Sortie : Le meilleur coup à jouer et le score associé

```
si C ne contient plus de coups possibles alors
    Retourn (None, score(C, J));
fin
(res, meilleur) ← (None, −2);
pour tous les coups possibles c faire
    C′ ← On joue le coup c dans C;
    (tmp, s) ← AlphaBeta(C′, score, changerJoueuse(J), −β, −α) ;
    si −s ≥ β alors
        return (c, −s)
    fin
    si −s > meilleur alors
        (res, meilleur) ← (c, −s);
    fin
    α = max(α, −s);
fin
return (res, meilleur)
```

Pour aller plus loin

- Théorie des jeux: `https://fr.wikipedia.org/wiki/Théorie_des_jeux`
- Équilibre de Nash: `https://fr.wikipedia.org/wiki/Équilibre_de_Nash`
- Jeux à somme nulle: `https://fr.wikipedia.org/wiki/Jeu_à_somme_nulle`