EXPLORATION ET HEURISTIQUES

For progress there is no cure.

John Von Neumann [**von_neumann_can_1955**]

À la fin de ce chapitre, je sais :

- ☞ expliquer la notion d'heuristique
- ☞ appliquer les algorithmes A* et minimax à des cas simples

A Au-delà des jeux d'accessibilité, les heuristiques

Une stratégie est connue pour les jeux d'accessibilité, mais, s'ils permettent de briller dans les salons, une fois la stratégie connue, la lassitude s'installe. De nombreux autres jeux existent, comme les échecs ou le go. Pour ces jeux, il est impensable de dessiner l'arbre de jeu, la combinatoire nous indique que le nombre de parties possibles est bien trop grand. Le nombre de Shannon est une tentative pour estimer le nombre de parties différentes [1] possibles. Il vaut 10^{123} pour les échecs ce qui est plus grand que le nombre d'atomes dans l'univers observable…

S'il nous faut donc renoncer à explorer ces arbres de jeux d'une manière exhaustive, rien ne nous empêche néanmoins de les explorer localement. Si on utilise un arbre de jeu exhaustif, la partie se finit lorsque la position est une feuille à laquelle est associé un gain ou un score. Lorsqu'on explore localement un arbre de jeu, les feuilles sont parfois absentes voire encore très loin de la position. C'est pourquoi le développement de ces algorithmes s'appuie sur des **heuristiques** capables d'estimer le gain d'une position sans disposer de l'intégralité de l'arbre de jeu.

■ **Définition 1 — Heuristique.** Une heuristique [a] est une approche de résolution de problème à partir de connaissances incomplètes. Une heuristique doit permettre d'aboutir en un temps limité à des solutions néanmoins acceptables mêmes si elles ne sont pas nécessairement optimales.

a. Je trouve en grec ancien.

1. qui ont un sens

B Explorer un arbre de jeu à somme nulle

> Tu sais que ce que tu peux espérer de mieux est d'éviter le pire.
>
> Italo Calvino, 1979

L'algorithme Minimax associé à une heuristique permet de développer des stratégies sans avoir à explorer tout l'arbre de jeu. Il découle naturellement du théorème du même nom démontré par Von Neumann en 1926.

L'algorithme 1 détaille la procédure Minimax. Le principe est le suivant : on construit un arbre de jeu incomplet comme celui-de la figure 1. La hauteur de l'arbre est un entier fixé qui limite la profondeur de l'exploration de l'algorithme. La complexité s'en trouve ainsi améliorée.

Chaque niveau de l'arbre est contrôlé par un seul joueur. Comme, pour les jeux d'accessibilité considérés, les gains de l'un sont les pertes de l'autre, on choisit de nommer les joueurs J_{max} en rouge et J_{min} en cyan. Le but du jeu est de maximiser le gain pour J_{max}, son score est donc positif, et, symétriquement, minimiser le gain pour J_{min} dont le score est donc négatif.

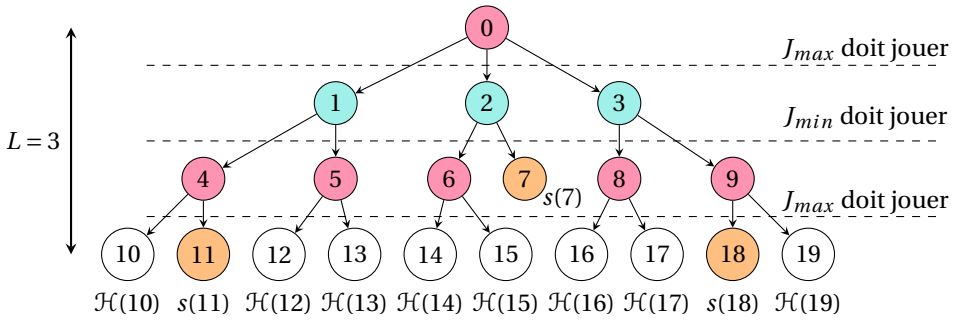

FIGURE 1 – Exemples d'arbre minimax. Chaque niveau est contrôlé par un seul joueur, J_{max} en rouge et J_{min} en cyan. La hauteur de l'arbre L est telle que seule une partie de l'arbre de jeu est accessible. Certaines feuilles sont visibles (en orange, c'est l'automne). L'arbre minimax s'achève donc parfois sur des nœuds internes pour lesquels on donne une estimation du gain (non coloré).

■ **Définition 2 — Définition du score des joueurs pour une position donnée.** Le jeu associe à chaque feuille de l'arbre minimax un gain qui devient le score du joueur qui atteint cette feuille. Si la feuille est en position p alors on choisit de noter ce score $s(p) \in \mathbb{R}$. À partir du score associé aux feuilles, on peut définir récursivement le score maximal ou minimal associé à un nœuds internes p de l'arbre minimax comme suit :

$$\mathcal{S}(p) = \begin{cases} s(p) & \text{si } p \text{ est une feuille} \\ \max\{\mathcal{S}(f), f \text{ fils de } p\} & \text{si } p \text{ est contrôlé par } J_{max} \\ \min\{\mathcal{S}(f), f \text{ fils de } p\} & \text{si } p \text{ est contrôlé par } J_{min} \end{cases} \quad (1)$$

Ainsi, à la fin de la partie, si J_{max} en position p effectue les choix décrit ci-dessus, son score final sera au moins $\mathcal{S}(p)$. De même, le score de J_{min} en position p sera au plus $\mathcal{S}(p)$. Ceci peut se démontrer par récurrence.

Le score d'un joueur ne peut se calculer tel que décrit ci-dessus puisqu'on ne connaît pas l'intégralité de l'arbre de jeu. L'algorithme 1 suppose donc qu'on connaît une heuristique $\mathcal{H}$ pour estimer le score d'une position d'un nœud intermédiaire. Cette heuristique est une fonction de la position p dans l'arbre et sa valeur est un nombre réel $\mathcal{H}(p)$.

Algorithme 1 Minimax

Entrée : p un position dans l'arbre de jeu (un nœud de l'arbre)
Entrée : s la fonction de score sur les feuilles
Entrée : $\mathcal{H}$ l'heuristique de calcul du score pour un nœud interne
Entrée : L la profondeur maximale de l'arbre Minimax

```
1 : Fonction MINIMAX(p, s, H , L)
2 :     si p est une feuille alors
3 :         renvoyer s(p)
4 :     si L = 0 alors
5 :         renvoyer H(p)                          ▷ On arrête d'explorer, on estime
6 :     si p est contrôlé par J_max alors
7 :         M ← −∞
8 :         p_M un nœud vide
9 :         pour chaque fils f de p répéter
10 :            v ←MINIMAX(f,s,H, L − 1)            ▷ v est un score de J_min
11 :            si v > M alors
12 :                M ← v
13 :                p_M = f
14 :        renvoyer M, p_M          ▷ Valeur maximale trouvée et la racine de cette solution
15 :    sinon
16 :        m ← +∞
17 :        p_m un nœud vide
18 :        pour chaque fils f de p répéter
19 :            v ←MINIMAX(f,s,H, L − 1)            ▷ v est un score de J_max
20 :            si v < m alors
21 :                m ← v
22 :                p_m = f
23 :        renvoyer m, p_m          ▷ Valeur minimale trouvée et la racine de cette solution
```

■ **Exemple 1 — Calcul des scores sur un arbre Minimax.** La figure 2 superpose les scores calculés par l'algorithme Minimax sur chaque nœud. Le joueur J_{max} peut donc espérer au plus un score de 6 s'il se trouve en position 0.

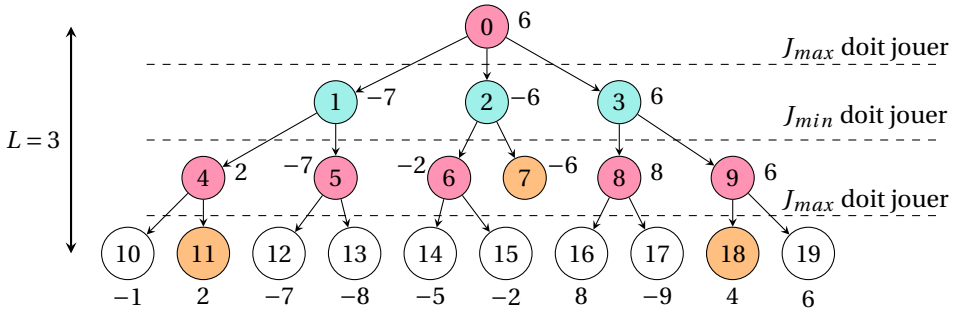

FIGURE 2 – Exemples d'arbre minimax complété avec les scores. Chaque niveau est contrôlé par un seul joueur, J_{max} en rouge et J_{min} en cyan.

C Élagage $\alpha\beta$ sur un arbre Minimax --→ HORS PROGRAMME

Selon les situations considérées, limiter la profondeur d'exploration de l'arbre de jeux peut s'avérer être insuffisant pour réduire la complexité du problème. L'élagage $\alpha\beta$ est une technique pour ne pas explorer certaines branches de l'arbre qui n'ont pas besoin de l'être. Le principe est détaillé sur la figure 3.

On distingue deux types d'élagage possible, les types α et β comme le montre les figures 4 et 5. Pour améliorer la complexité de l'algorithme Minimax, on peut choisir d'élaguer comme le montre l'algorithme 2.

D A* pour trouver un chemin

Si le jeu que l'on considère ne peut pas être modélisé selon un arbre Minimax, alors la situation n'est pas désespérée car il nous reste encore les graphes. Il est toujours possible de modéliser l'évolution d'un jeu comme une succession d'état, chaque coup joué permettant de passer d'un état à un autre. Développer une stratégie dans un jeu modélisé par un graphe à état revient donc à trouver un chemin d'un sommet à un autre dans un graphe. Or, nous sommes déjà bien outillés pour faire cela.

L'algorithme A*[2] est un algorithme couteau suisse qui peut être considéré comme un algorithme de Dijkstra muni d'une heuristique : là où Dijkstra ne tient compte que du coût du chemin déjà parcouru, A* considère ce coût et une heuristique qui l'informe sur le reste du

2. prononcer *A étoile* ou *A star*

Algorithme 2 Minimax avec élagage $\alpha\beta$

Entrée : p un position dans l'arbre de jeu (un nœud de l'arbre)
Entrée : s la fonction de score sur les feuilles
Entrée : $\mathcal{H}$ l'heuristique de calcul du score pour un nœud interne
Entrée : L la profondeur maximale de l'arbre Minimax
Entrée : α le niveau de coupure α ▷ $-\infty$ au démarrage
Entrée : β le niveau de coupure β ▷ $+\infty$ au démarrage

```
1 : Fonction MINIMAX_αβ(p, s, H , L, α, β)
2 :     si p est une feuille alors
3 :         renvoyer s(p)
4 :     si L = 0 alors
5 :         renvoyer H(p)                          ▷ On arrête d'explorer, on estime
6 :     si p est contrôlé par J_max alors
7 :         M ← −∞
8 :         p_M un nœud vide
9 :         pour chaque fils f de p répéter
10 :            v ←MINIMAX_αβ(f,s,H, L − 1, α, β)          ▷ v est un score de J_min
11 :            si v > M alors
12 :                M ← v
13 :                p_M = f
14 :            si v ⩾ β alors
15 :                renvoyer M                             ▷ Élagage de type β
16 :            α ← max(α, M)                  ▷ Mise à jour du niveau de l'élagage
17 :        renvoyer M, p_M          ▷ Valeur maximale trouvée et la racine de cette solution
18 :    sinon
19 :        m ← +∞
20 :        p_m un nœud vide
21 :        pour chaque fils f de p répéter
22 :            v ←MINIMAX_αβ(f,s,H, L − 1, α, β)          ▷ v est un score de J_max
23 :            si v < m alors
24 :                m ← v
25 :                p_m = f
26 :            si v ⩽ α alors
27 :                renvoyer m                             ▷ Élagage de type α
28 :            β ← min(β, m)                  ▷ Mise à jour du niveau de l'élagage
29 :        renvoyer m, p_m          ▷ Valeur minimale trouvée et la racine de cette solution
```

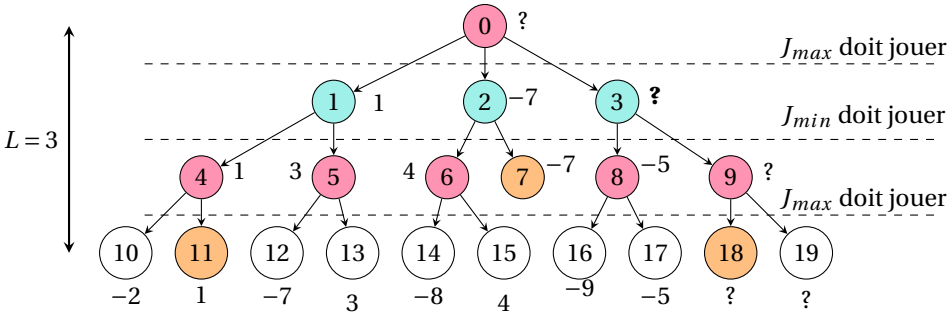

FIGURE 3 – L'élagage $\alpha\beta$ permet sur cet exemple d'éviter l'exploration complète du sous-arbre du nœud numéro 3. En effet, comme c'est un nœud de J_{min}, que le premier nœud du niveau a un score de 1 et que le score calculé du nœud 8 vaut -5, alors on comprend que J_{min} remontera au moins -5 et que J_{max} ne choisira pas ce nœud numéro 3 car ce n'est pas le maximum du niveau.

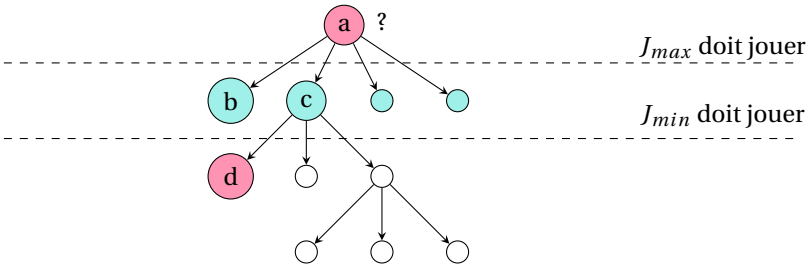

FIGURE 4 – L'élagage α : supposons que $\mathcal{S}(b)$ ait déjà été calculé par l'algorithme Minimax. Si $\mathcal{S}(b) \geqslant \mathcal{S}(d)$, alors il est inutile d'explorer le sous-arbre c : J_{max} ne choisira pas le nœud c, il lui préfèrera b.

chemin à parcourir. Il faut bien remarquer que le chemin qu'il reste à parcourir n'est pas nécessairement déjà exploré : il est rarement possible d'explorer tout le graphe à état d'un jeu. Si l'heuristique pour évaluer le reste du chemin à parcourir est bien choisie, alors A* converge aussi vite voire plus vite que Dijkstra[**unswmechatronics_dijkstras_2013**].

■ **Définition 3 — Heuristique admissible.** Une heuristique $\mathcal{H}$ est admissible si pour tout sommet du graphe d'état, $\mathcal{H}(s)$ est une borne inférieure de la plus courte distance séparant le sommet de départ du sommet d'arrivée.

■ **Définition 4 — Heuristique cohérente.** Une heuristique $\mathcal{H}$ est cohérente si pour tout arc (s, p) du graphe d'état $G = (V, E, w)$, $\mathcal{H}(s) \leqslant \mathcal{H}(p) + w(s, p)$.

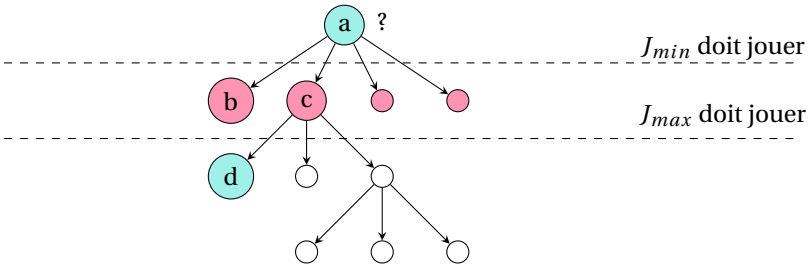

FIGURE 5 – L'élagage β : supposons que $\mathcal{S}(b)$ ait déjà été calculé par l'algorithme Minimax. Si $\mathcal{S}(b) \leqslant \mathcal{S}(d)$, alors il est inutile d'explorer le sous-arbre c : J_{min} ne choisira pas le nœud c, il lui préfèrera b.

■ **Définition 5 — Heuristique monotone.** Une heuristique $\mathcal{H}$ est monotone si l'estimation du coût **total** du chemin ne décroît pas lors du passage d'un sommet à ses successeurs. Pour un chemin $(s_0, s_1, \ldots, s_n)$, on $\forall 0 \leqslant i < j \leqslant n, c(s_j) \geqslant c(s_i)$.

E Explorer un graphe

Soit $G = (V, E, w)$ un graphe orienté pondéré. Soit d la fonction de distance utilisée par l'algorithme de Dijkstra (cf. algorithme **??**). A*, muni d'une fonction h permettant d'évaluer l'heuristique, calcule alors le coût total pour aller jusqu'à un sommet p comme suit :

$$c(p) = d(p) + h(p) \tag{2}$$

Le coût obtenu n'est pas nécessairement optimal, il dépend de l'heuristique.

Supposons que l'on cherche le chemin le plus court entre les sommets s_0 et p. Supposons que l'on connaisse un chemin optimal entre s_0 et un sommet s. Alors on peut écrire que le coût total vers le sommet p vaut :

$$\begin{aligned} c(p) &= d(p) + h(p) && (3)\\ &= d(s) + w(s,p) + h(p) && (4)\\ &= d(s) + h(s) + w(s,p) - h(s) + h(p) && (5)\\ &= c(s) + w(s,p) - h(s) + h(p) && (6) \end{aligned}$$

Ainsi, on peut voir l'algorithme A* comme un algorithme de Dijkstra muni :

- de la distance $\tilde{d} = c$,
- et de la pondération $\tilde{w}(s,p) = w(s,p) - h(s) + h(p)$.

L'algorithme 3 donne le détail de la procédure à suivre.

Algorithme 3 A*

Fonction ASTAR($G = (V, E, w)$, a) ▷ Sommet de départ a
  $\Delta \leftarrow a$
  $\Pi \leftarrow$
  $\tilde{d} \leftarrow$ l'ensemble des distances au sommet a
  $\forall s \in V, \tilde{d}[s] \leftarrow \tilde{w}(a, s)$ ▷ Le graphe est partiel, l'heuristique fait le reste
  tant que $\bar{\Delta}$ n'est pas vide **répéter** ▷ $\bar{\Delta}$: sommets dont la distance n'est pas connue
    Choisir u dans $\bar{\Delta}$ tel que $\tilde{d}[u] = \min\left(\tilde{d}[v], v \in \bar{\Delta}\right)$
    $\Delta = \Delta \cup \{u\}$
    pour $x \in \bar{\Delta}$ **répéter** ▷ Ou bien $x \in \mathcal{N}_G(u) \cap \bar{\Delta}$, pour tous les voisins de u dans $\bar{\Delta}$
      si $\tilde{d}[x] > \tilde{d}[u] + \tilde{w}(u, x)$ **alors**
        $\tilde{d}[x] \leftarrow \tilde{d}[u] + \tilde{w}(u, x)$
        $\Pi[x] \leftarrow u$ ▷ Pour garder la tracer du chemin le plus court
  renvoyer $\tilde{d}$, Π