Jeux

M1 IDD 2019–2020 *Représentation des connaissances et raisonnement*

Stéphane Airiau

LAMSADE

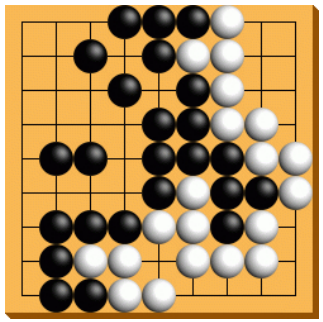
DAUPHINE
UNIVERSITÉ PARIS

Types de jeux

On s'intéresse pour le moment aux jeux :

- à deux joueurs
- où les joueurs jouent un *seul* coup en *alternance*
- où chaque joueur peut observer le coup de son adversaire
- à somme constante (si un joueur gagne, l'autre perd)

exemples : les échecs, le jeu de go



Modélisation

- s_0 état initial
- `joueur(s)` : définit quel joueur doit effectuer une action
- `actions(s)` : donne les coups légaux dans l'état `s`
- `resultat(s,a)` : fonction de transition qui définit le résultat de l'action `a` prise dans l'état `s`
- `terminal?(s)` retourne si le jeu est terminé
- `utilité(s)` : retourne une valeur numérique à chaque joueur lorsque le jeu est terminé

Par exemple pour les échecs :

- le gagnant reçoit un score de +1
- le perdant reçoit 0
- en cas d'égalité, chaque joueur reçoit $\frac{1}{2}$

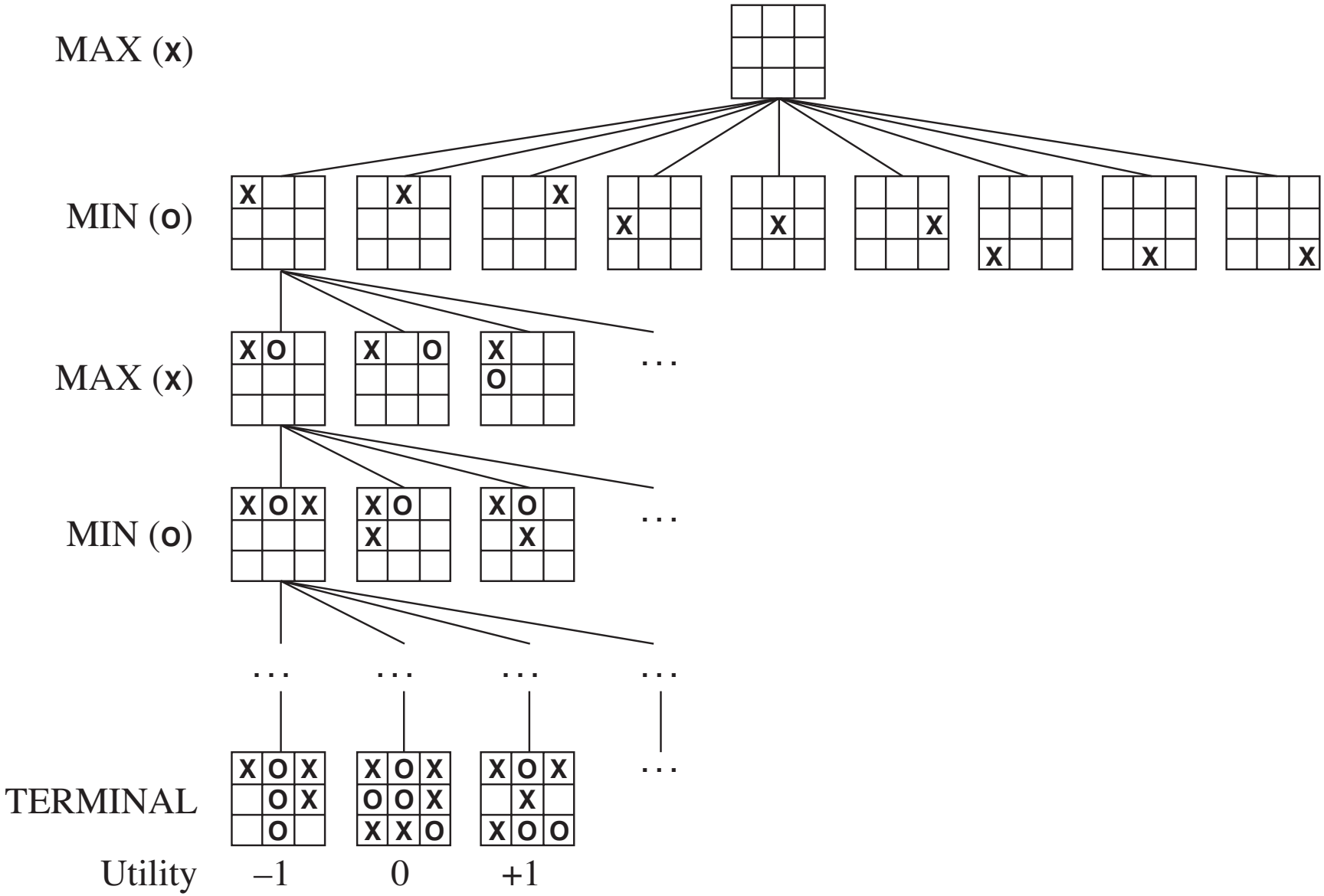
On parle de jeu à somme constante
(les joueurs ont des objectifs complètement contraires)

Représentation avec un arbre

On peut représenter une partie à l'aide d'un arbre :

- à chaque niveau de l'arbre, un joueur choisit un coup
- chaque joueur cherche à gagner
  ⇨ le jeu est à somme constante
  ⇨ un joueur cherche a maximiser son score
  ⇨ son adversaire cherche à le faire perdre ⇨ cherche à minimiser le score
- on se place du point de vue d'un des joueur
  - on appelle ce joueur `MAX` (il cherche à maximiser son score)
  - l'autre joueur est appelé `MIN` (il cherche à minimiser le score de `MAX`)

Exemple du morpion



le morpion

- combien y-t-il de configurations différentes ?

le morpion

- combien y-t-il de configurations différentes ?
- moins que 9 != 362,880 feuilles dans l'arbre

le morpion

- combien y-t-il de configurations différentes ?
- moins que 9 != 362,880 feuilles dans l'arbre
- certaines configurations vont se retrouver plusieurs fois dans l'arbre
  ⇨ table de transposition pour se rappeler qu'on a déjà vu cette configuration

le morpion

- combien y-t-il de configurations différentes ?
- moins que 9 != 362,880 feuilles dans l'arbre
- certaines configurations vont se retrouver plusieurs fois dans l'arbre
  ⇨ table de transposition pour se rappeler qu'on a déjà vu cette configuration
- généralement, les jeux sont trop gros pour raisonner sur l'arbre en entier
  ⇨ on peut le voir comme une abstraction théorique que l'on **n**'utilisera **pas** en pratique.

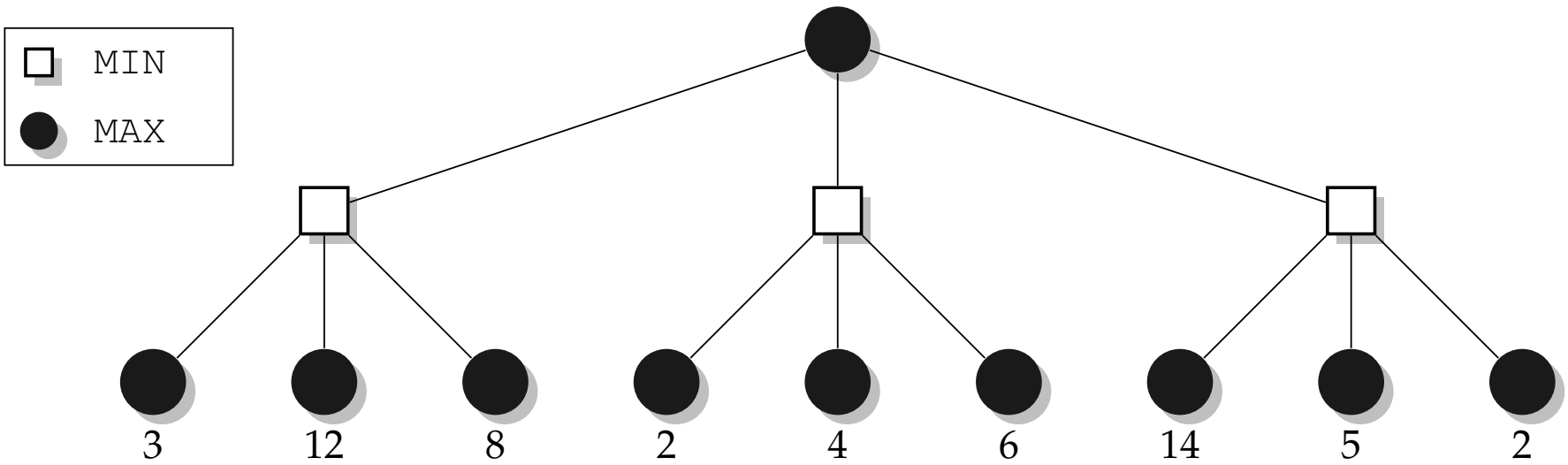
Résolution Optimale

- séquence de coups qui mènent à un état gagnant !
- maix `MIN` joue aussi !
  ⇨ une stratégie doit prendre en compte `MIN` et prévoir un coup pour chaque action possible de `MIN`
- on peut supposer que `MIN` joue de façon optimale aussi !

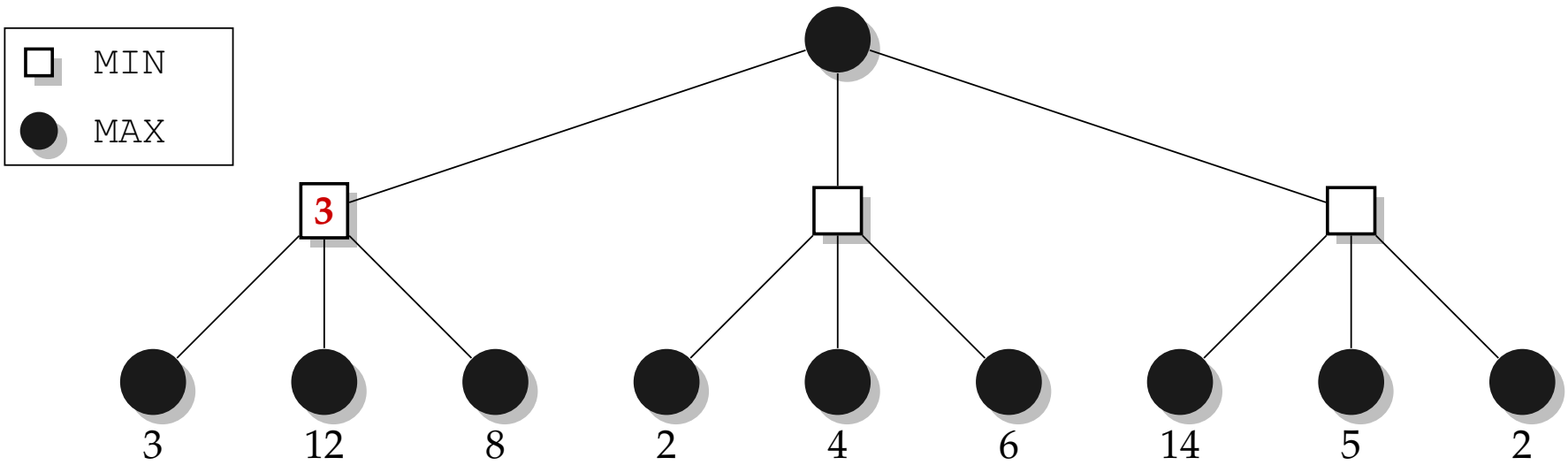
`MINIMAX(s)` =

- `utilité(s)` si `s` est un état final
- $\max_{a \in actions(s)}$ `MINIMAX(resultat(s,a))` si `MAX` joue
- $\min_{a \in actions(s)}$ `MINIMAX(resultat(s,a))` si `MIN` joue

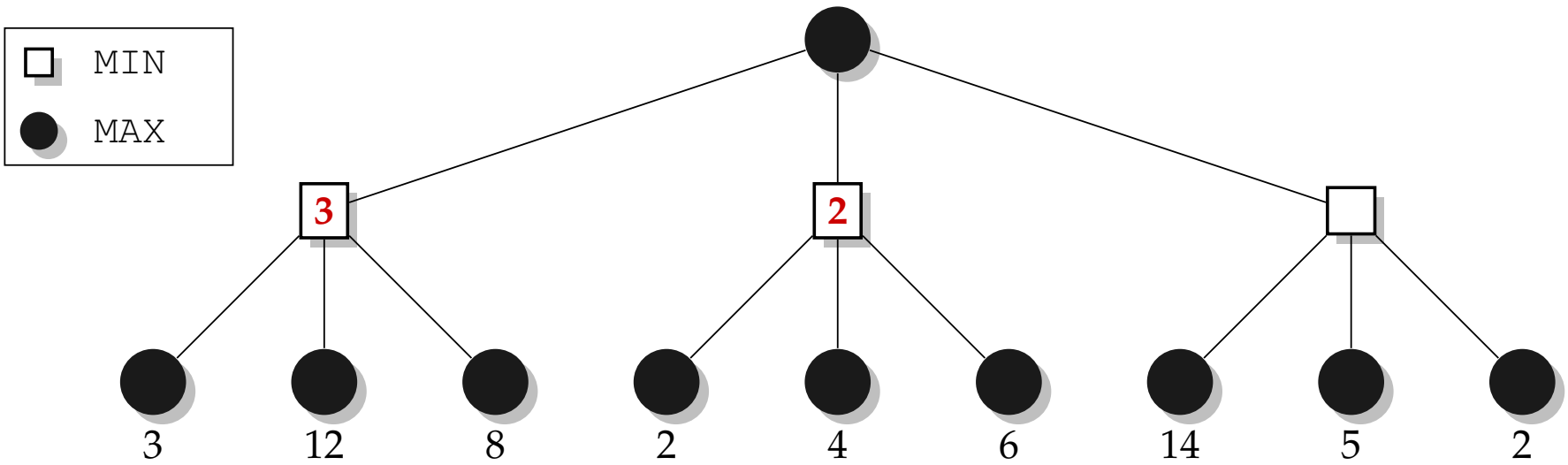
Exemple



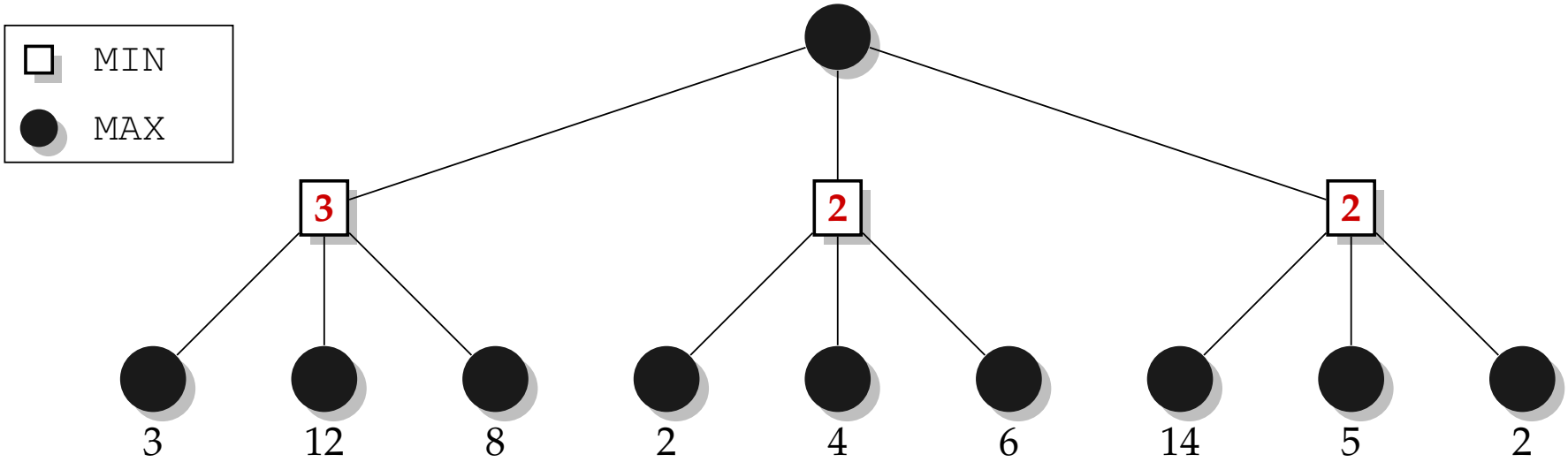
Exemple



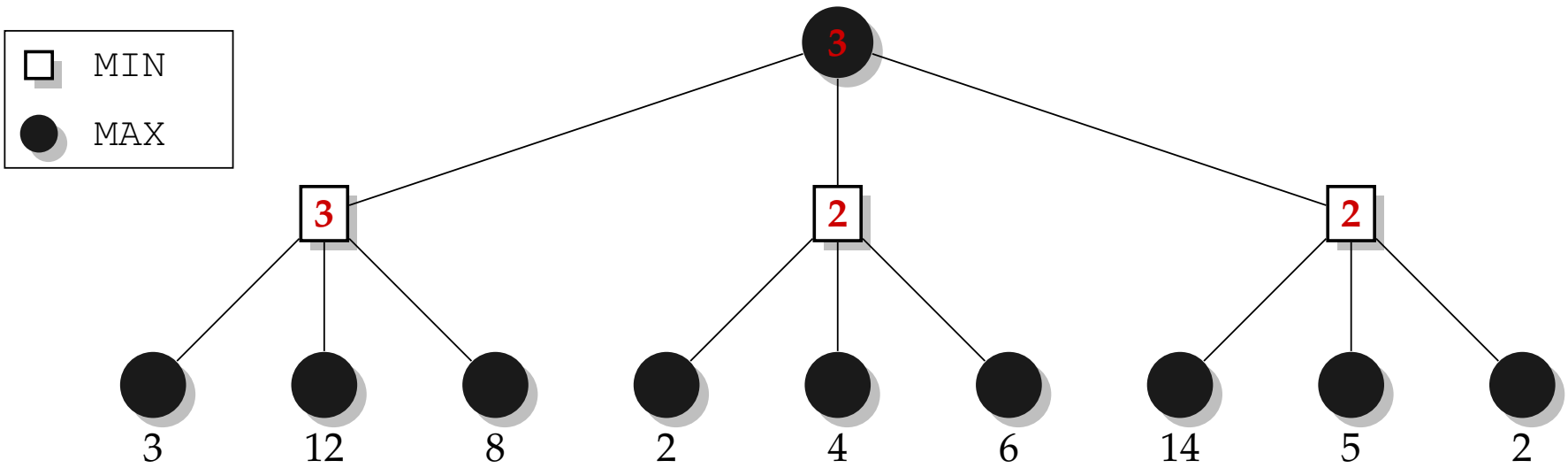
Exemple



Exemple



Exemple



Algorithmes

```
1 function MINIMAX_DECISION(s) returns an action
2 arg max_{a∈actions(s)} MIN_VALUE(result(s,a))
```

```
1 function MIN_VALUE(s) returns a utility value
2     if terminal?(s) then return utility(s)
3     v ← ∞
4     for each a ∈ actions(s) do
5         v ← min{v, MAX_VALUE(result(a,s))}
6     return v
```

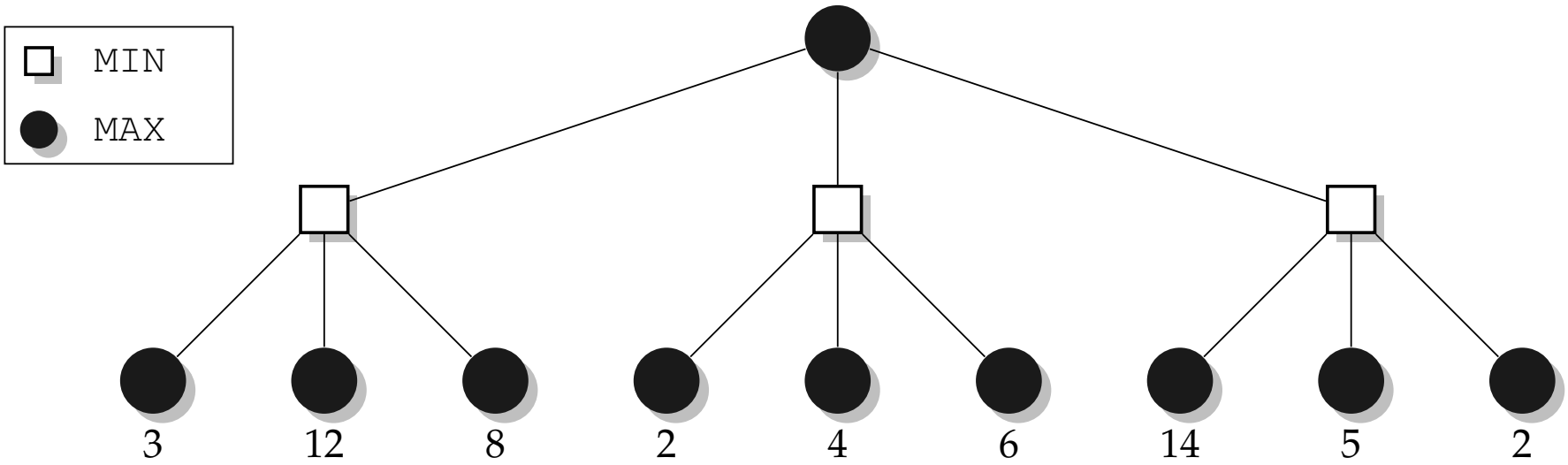
```
1 function MAX_VALUE(s) returns a utility value
2     if terminal?(s) then return utility(s)
3     v ← −∞
4     for each a ∈ actions(s) do
5         v ← max{v, MIN_VALUE(result(a,s))}
6     return v
```

Jeux à plus de deux joueurs

- l'algorithme peut marcher
- cependant, les autres joueurs n'ont pas toujours le même but (de minimiser le score du même joueur)
- il faut analyser plus finement le jeu
  ⇨ théorie des jeux (économie, Academy Awards)

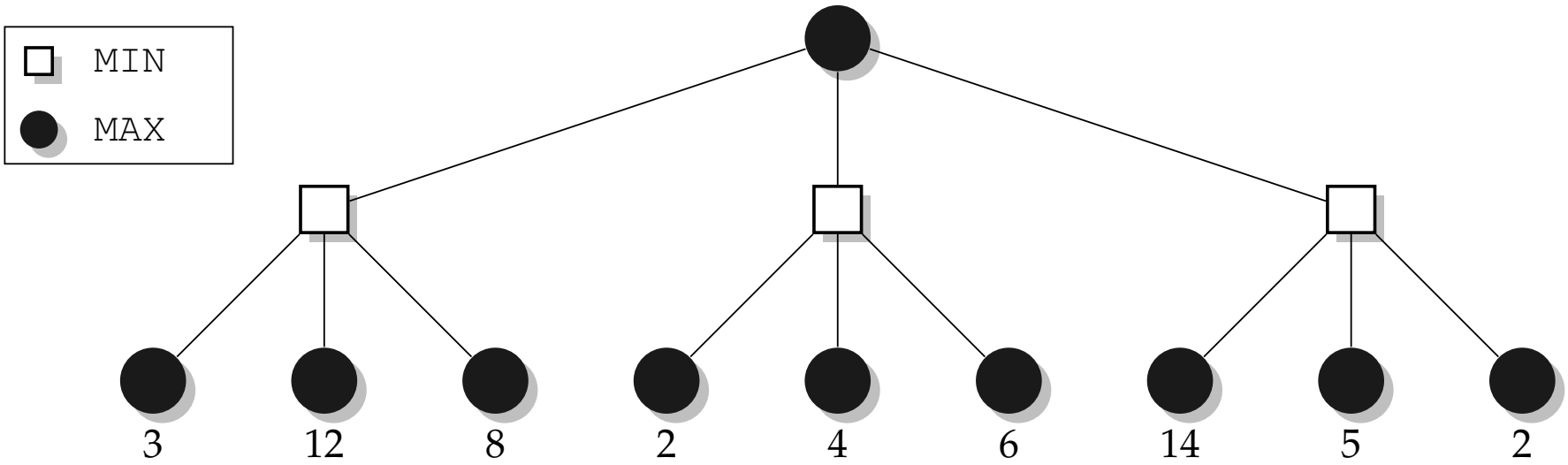
Elagage

Peut-on éviter de visiter des sous arbres entiers
sans risque de manquer une solution optimale ?



Elagage

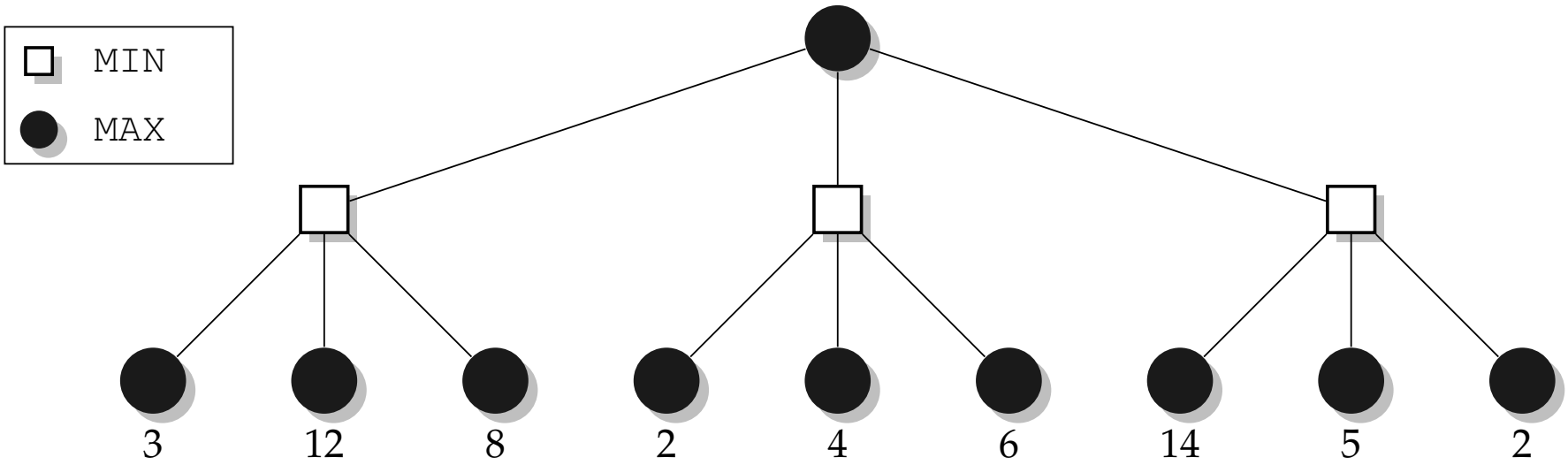
Peut-on éviter de visiter des sous arbres entiers
sans risque de manquer une solution optimale ?



$$\begin{aligned}\texttt{MINIMAX(racine)} &= \max\{\min\{3,12,8\},\min\{2,x,y\},\min\{14,5,2\}\}\\ &= \max\{3,z,2\} \text{ où } z = \min\{2,x,y\} \leqslant 2,\\ &= 3\end{aligned}$$

Elagage

Peut-on éviter de visiter des sous arbres entiers
sans risque de manquer une solution optimale ?

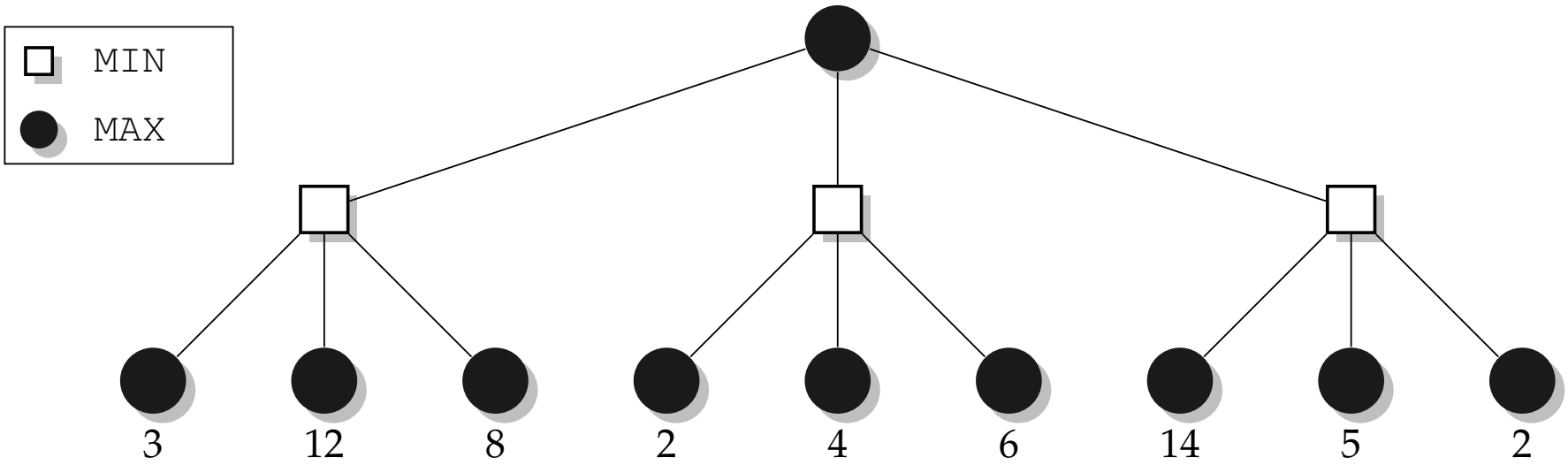


$$\begin{aligned}\texttt{MINIMAX(racine)} &= \max\{\min\{3,12,8\},\min\{2,x,y\},\min\{14,5,2\}\}\\ &= \max\{3,z,2\} \text{ où } z = \min\{2,x,y\} \leqslant 2,\\ &= 3\end{aligned}$$

⇨ Quelles que soient les valeurs de x et y, on connait la valeur de `MINIMAX(racine)` !

Elagage

Peut-on éviter de visiter des sous arbres entiers
sans risque de manquer une solution optimale ?

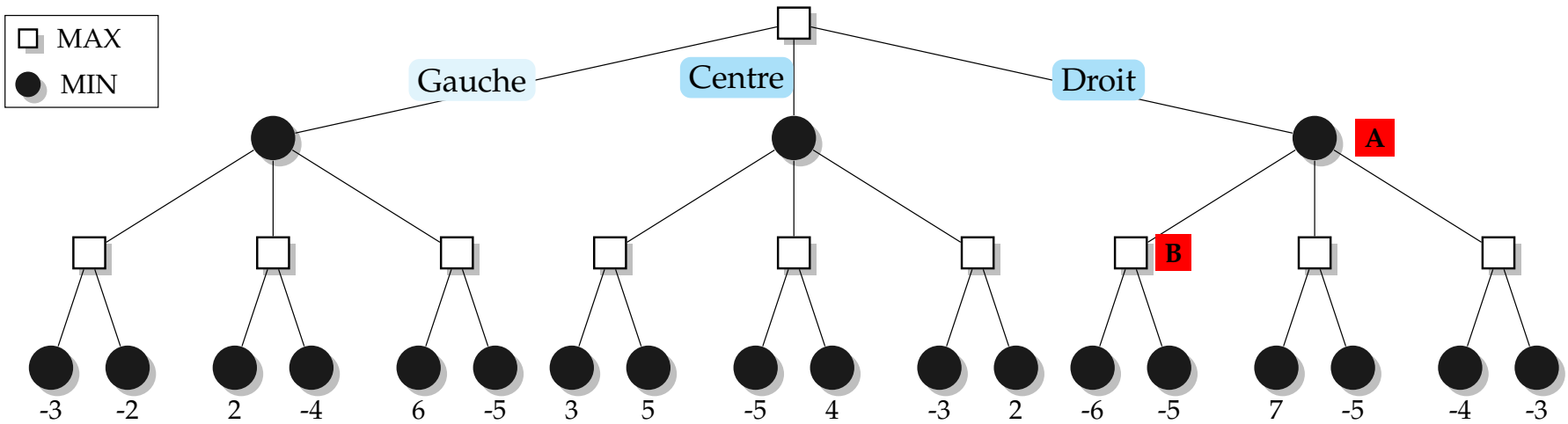


$$\texttt{MINIMAX(racine)} = \max\{\min\{3,12,8\},\min\{2,x,y\},\min\{14,5,2\}\}$$
$$= \max\{3,z,2\} \text{ où } z = \min\{2,x,y\} \leqslant 2,$$
$$= 3$$

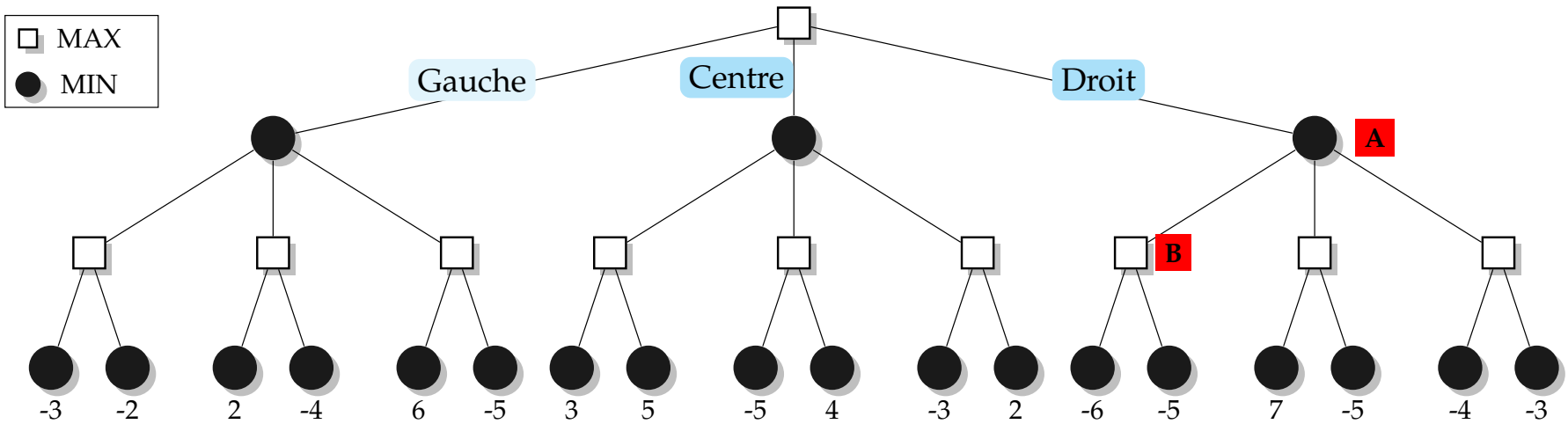
⇨ Quelles que soient les valeurs de x et y, on connait la valeur de `MINIMAX(racine)` !

⇨élagage pour `MAX`

Elagage

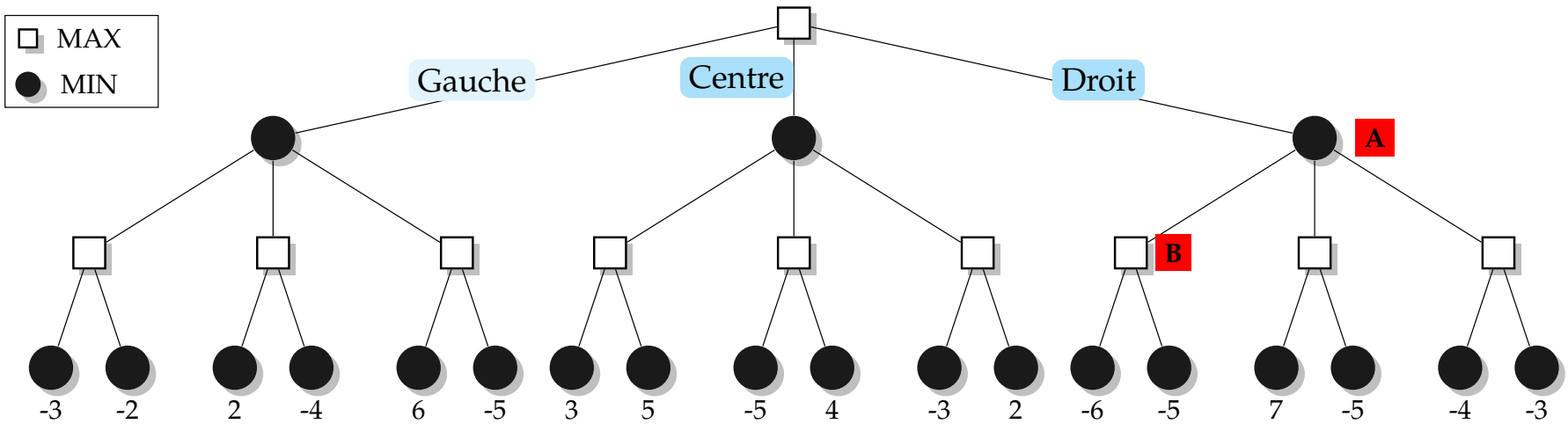


Elagage



- au moment d'examiner le noeud `A`, le joueur `MAX` a la garantie d'avoir au moins 2

Elagage



- au moment d'examiner le noeud `A`, le joueur `MAX` a la garantie d'avoir au moins 2
- après examen du sous-arbre sous `B`,
  on sait que si `MAX` joue l'action `Droit` au premier coup alors `MIN` aura au moins -5
  ⇨ on n'a pas besoin d'étudier les autres fils du noeud `A`
  ⇨ élagage pour `MIN`.

Elagage α-β

L'idée est donc de maintenir deux valeurs :

- α valeur de la meilleure (plus haute) valeur jusqu'ici pour `MAX`
- β valeur de la meilleure (plus basse) valeur jusqu'ici pour `MIN`

Le but est donc de mettre à jour α et β et d'élaguer la recherche quand la valeur du noeud considéré est pire que α pour `MAX` ou β pour `MIN`.

Algorithmes

```
1  function ALPHA_BETA(s) returns an action
2  v ← MAX_VALUE(s, −∞, +∞)
2  return a ∈ actions(s) with value v
```

```
1  function MIN_VALUE(s, α, β) returns a utility value
2      if terminal?(s) then return utility(s)
3      v ← +∞
4      for each a ∈ actions(s) do
5          v ← min{v, MAX_VALUE(result(a, s), α, β)}
6          if v ⩽ α then return v
7          β ← min{β, v}
8      return v
```

```
1  function MAX_VALUE(s, α, β) returns a utility value
2      if terminal?(s) then return utility(s)
3      v ← −∞
4      for each a ∈ actions(s) do
5          v ← max{v, MIN_VALUE(result(a, s), α, β)}
6          if v ⩾ β then return v
7          α ← max{α, v}
8      return v
```

Propriétés

- décision optimale si l'adversaire joue de façon optimale
  si l'adversaire ne joue pas parfaitement, on va faire mieux
  mais la solution ne sera peut être pas optimale dans ce cas
- complexité (temps) $\mathcal{O}(b^m)$ où b est le facteur de branchement et m la profondeur maximale
- complexité (mémoire) $\mathcal{O}(b \cdot m)$ (recherche en profondeur d'abord)

pour les échecs $b \approx 35$ et $m \approx 100$ pour des parties "raisonnables"
⇨ solution exacte impossible !

si on pouvait choisir optimalement l'ordre des actions à jouer pour élaguer au maximum, on a en général une complexité autour de $\mathcal{O}(b^{\frac{m}{2}})$
cela reste insuffisant pour les échecs !

Utiliser des fonctions d'évaluation si on ne peut pas finir la partie

- utiliser la connaissance du jeu pour bâtir une heuristique mesurant la qualité de l'état
  ex : donner des points pour les différentes pièces : 1 point pour pion, 3 pour une tour, etc..
  utiliser une somme pondérée comme fonction d'évaluation
- utiliser des critères plus complexes (les poids peuvent être en fonction de la présence de certaines combinaisons de pièces)
- utiliser des patterns de positions
- utiliser des simulations Monte Carlo pour évaluer les états

⇨ tant qu'il reste du temps, exécuter `Iterative-Deepening-Search`

il ne faudrait pas s'arrêter à des états où l'évaluation a des risques de changer drastiquement
l'effet d'horizon peut nous tromper (on peut imaginer une situation dans laquelle on est sûr de perdre, mais on peut repousser l'inévitable en faisant des actions qui ne vont rien changer)

Autres types de jeux

- jeux où toute l'information n'est pas observable (*observation imparfaite*)
  exemple : jeux de cartes (on ne connait pas les cartes de l'adversaire)
  autre exemple : bataille navale
- jeux avec un élément de chance (on doit lancer des dés)
  exemple : backgammon, monopoly
- jeux avec un élément de chance et avec information imparfaite
  pocker, scrabble, bridge, guerre nucléaire