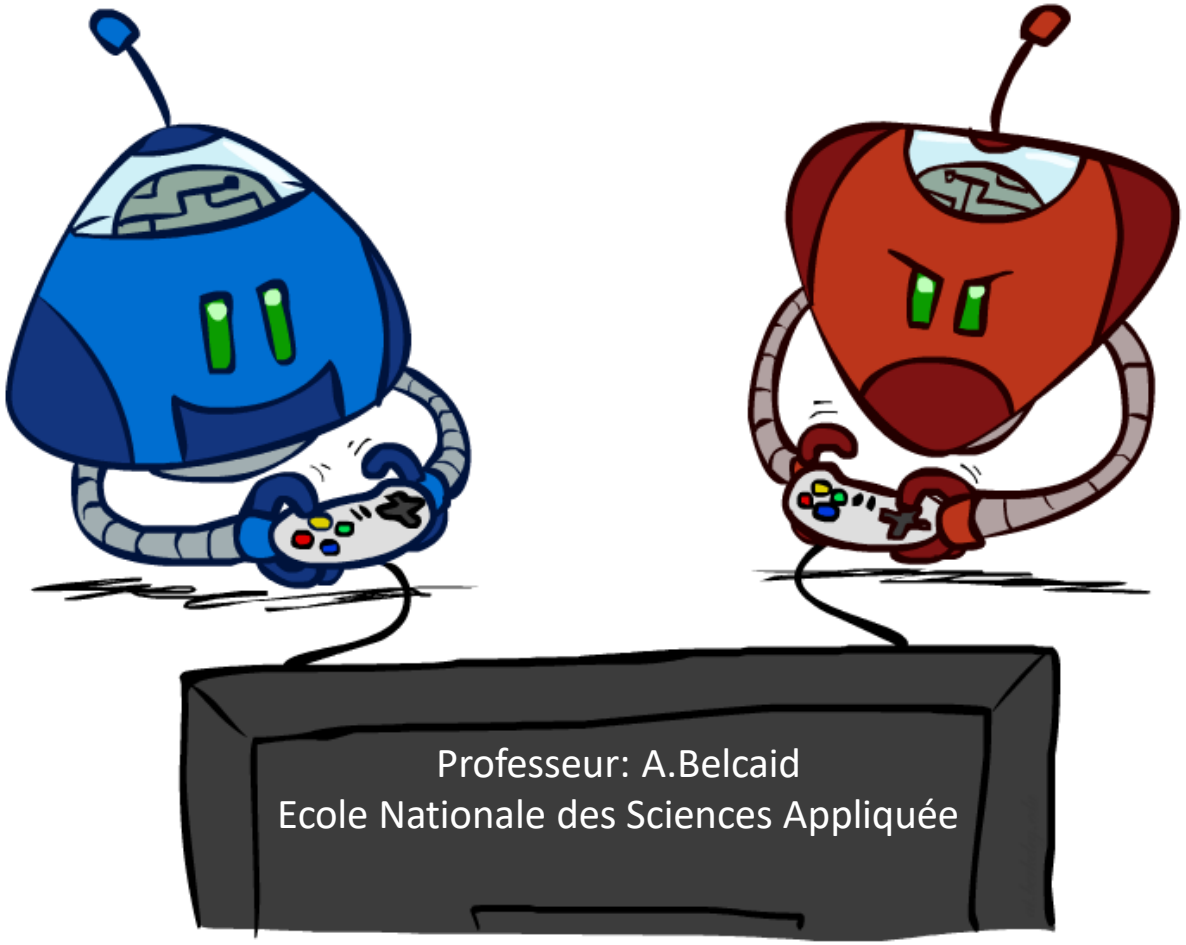
Intelligence artificielle

Recherche en situation adverse

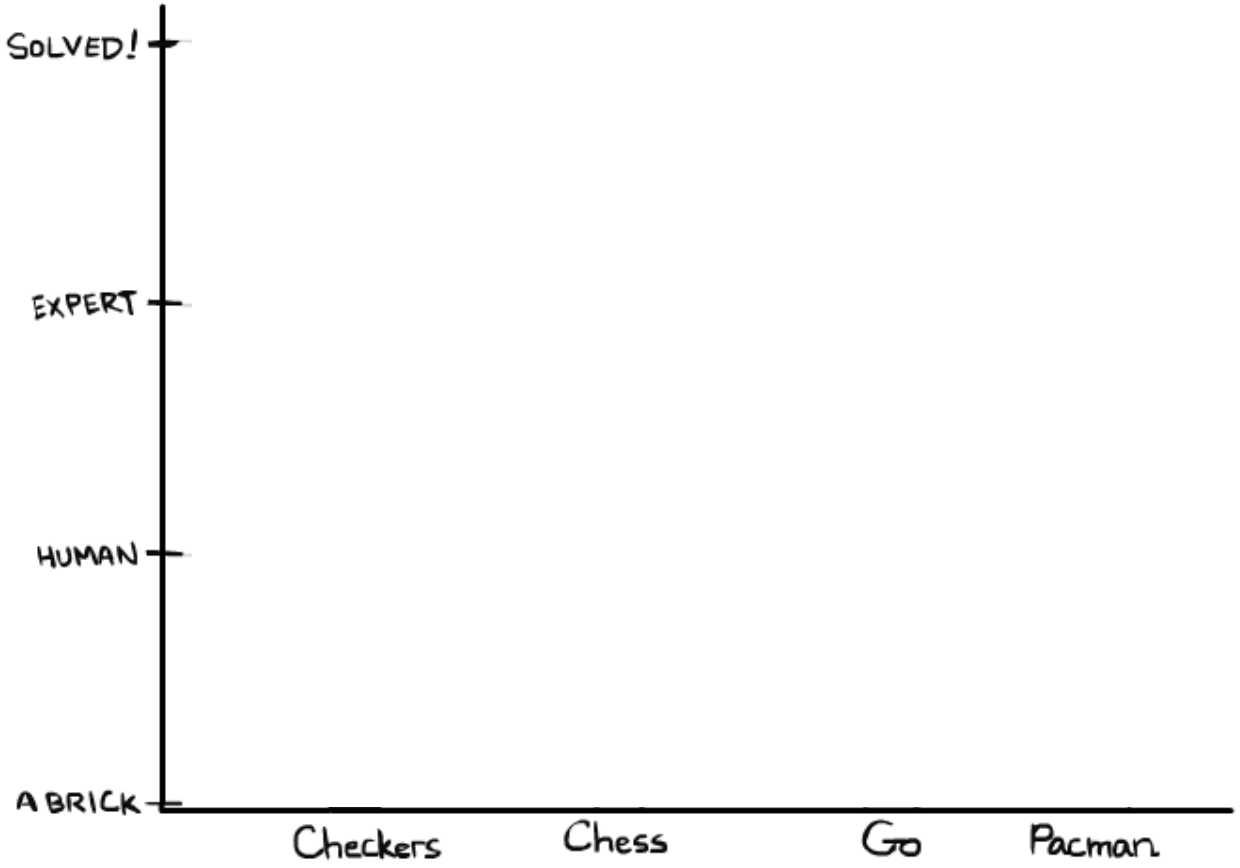


Professeur: A.Belcaid
Ecole Nationale des Sciences Appliquée

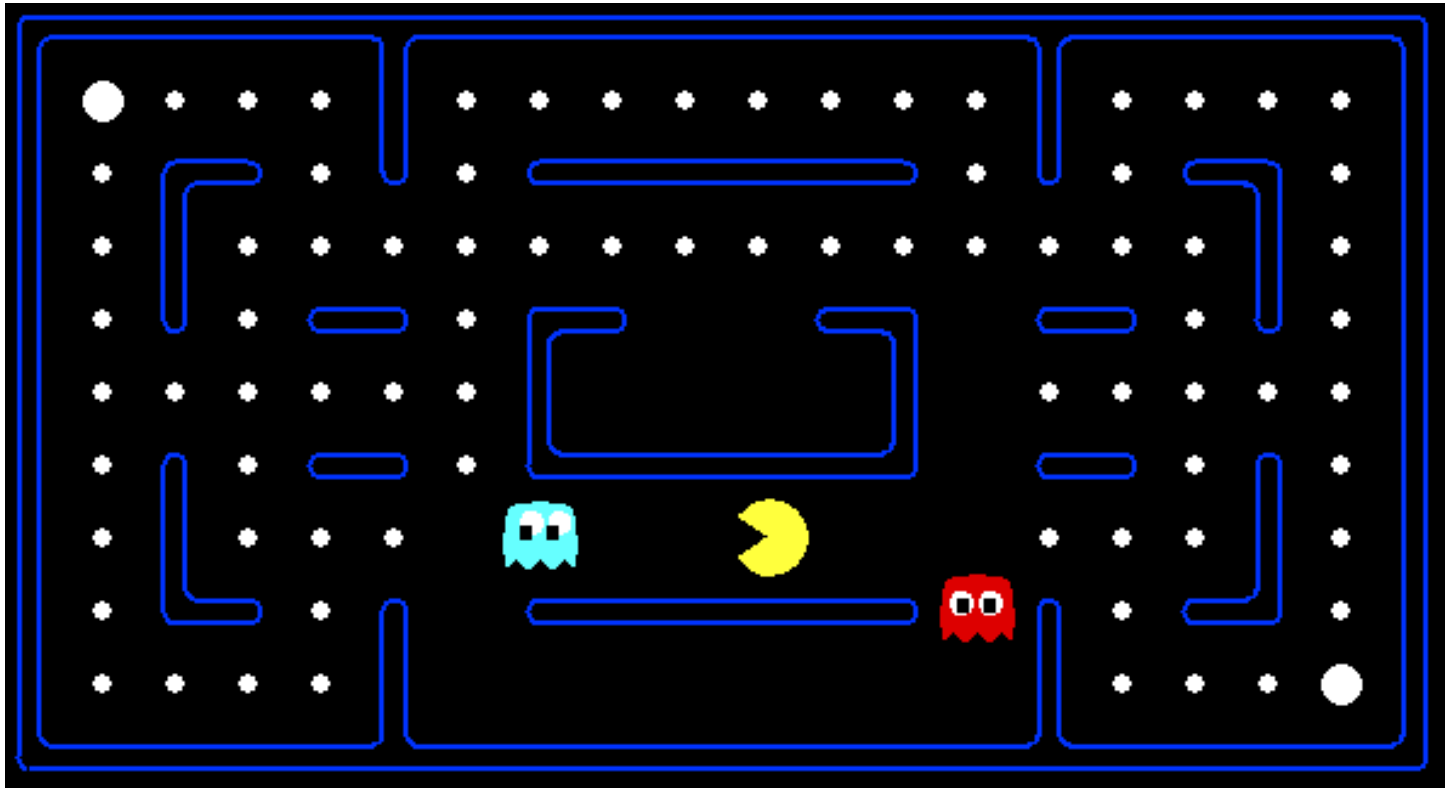
[Les slides sont crées par Dan Klein et Pieter Abbeel pour CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.]

Game Playing State-of-the-Art

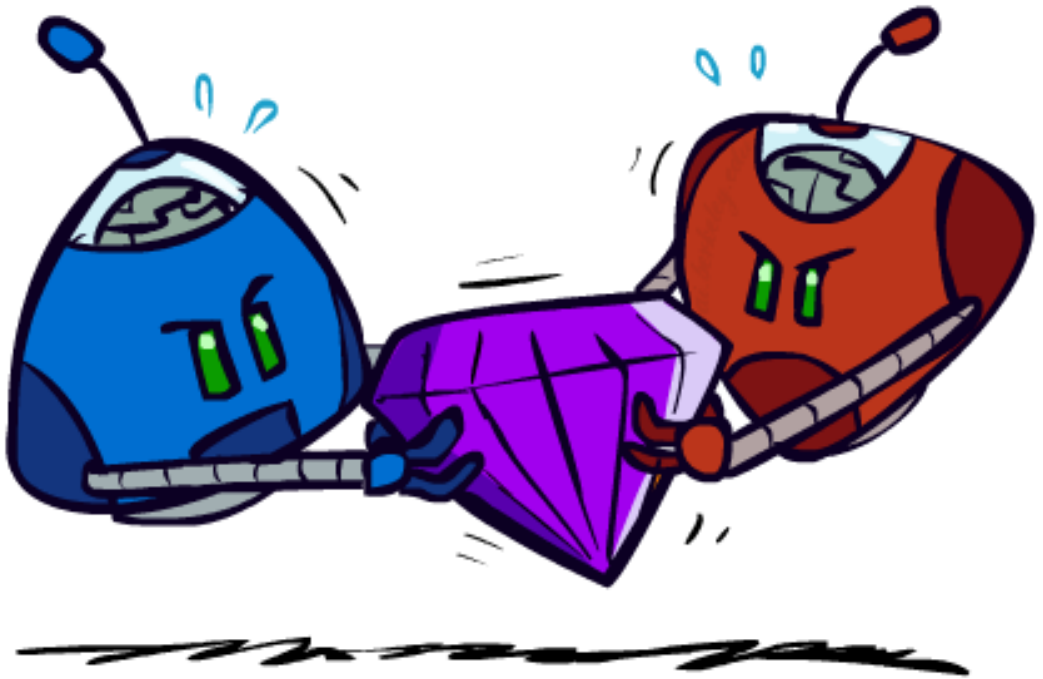
- **Dames:** 1950: Premier ordinateur. 1994: Premier champion: Chinook termine 40 ans du reigne du champion Marion Tinsley en utilisant un jeude fin de partie complet à 8 pieces. 2007: Résolu!
- **Echecs:** 1997: Deep Blue bat le champion du monde Gary Kasparov dans un match a six partie. Deep Blue examinait 200M positions par secondes, utilisait une évalution sophistiquée pour étendre une ligne en 40 coups prochains. Les programmes recent servent comme reference au Grand masters.
- **Go:** le programme AlphaGo a battu à deux reprise le champion du monde Ke Jie lors d'un tournoi organisé en chine. AlphaGo est conçu par DeepMind.
- **Pacman**



Stratégie extradite du calcul

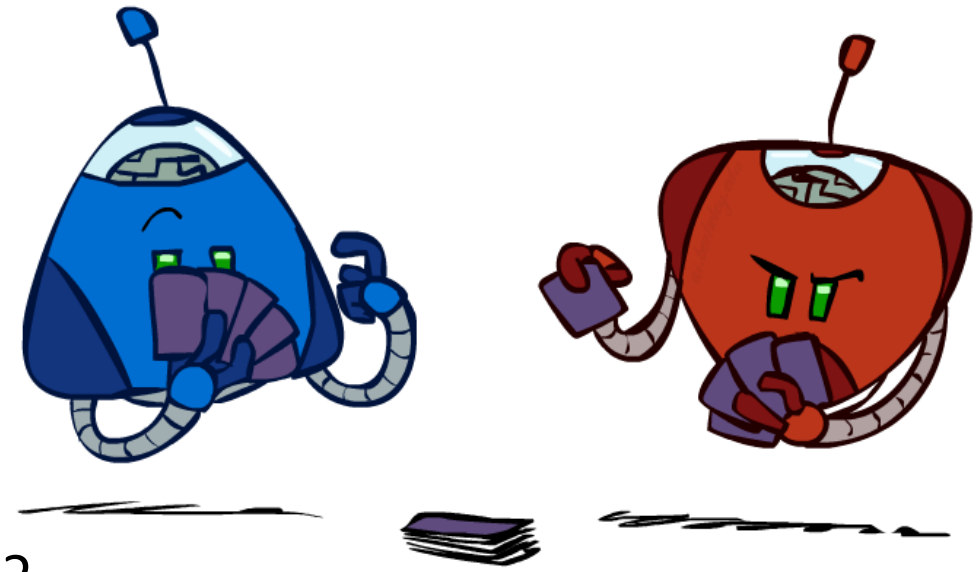


Jeux en adversité



Types de jeux

- Different type de jeux!

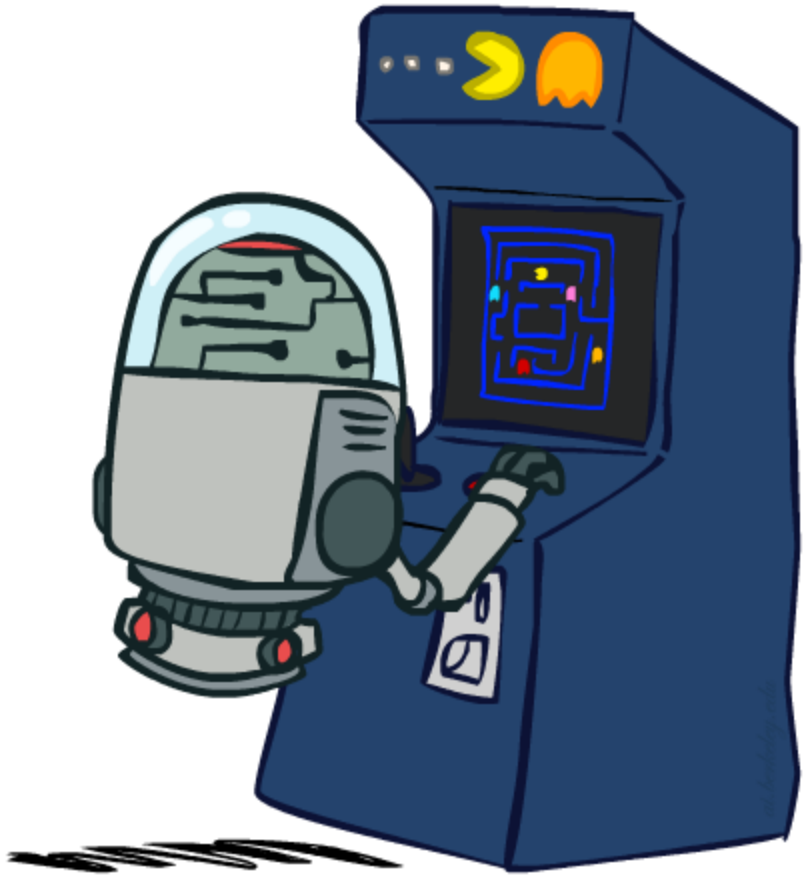


- Axes:
  - Déterministique ou stochastique?
  - Un, deux, ou plusieur joueurs?
  - Somme zero?
  - Information complète (On peut voir les états)?

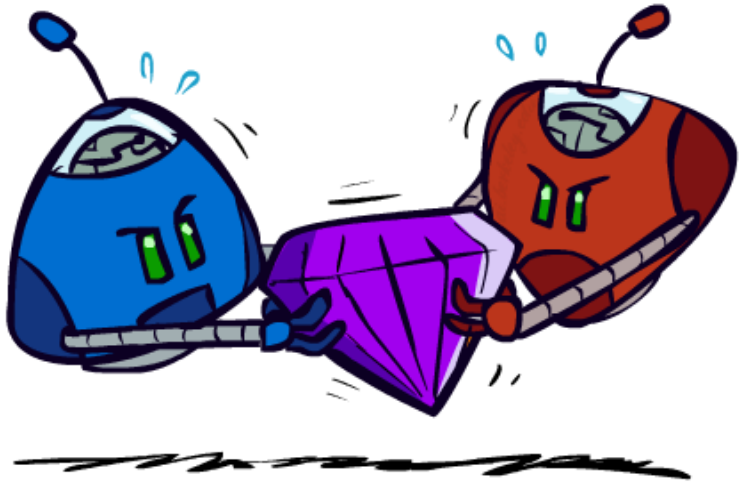
- Dévelloper des algorithms pour calculer une strategie (policy) qui recommande un coup pour chaque état.

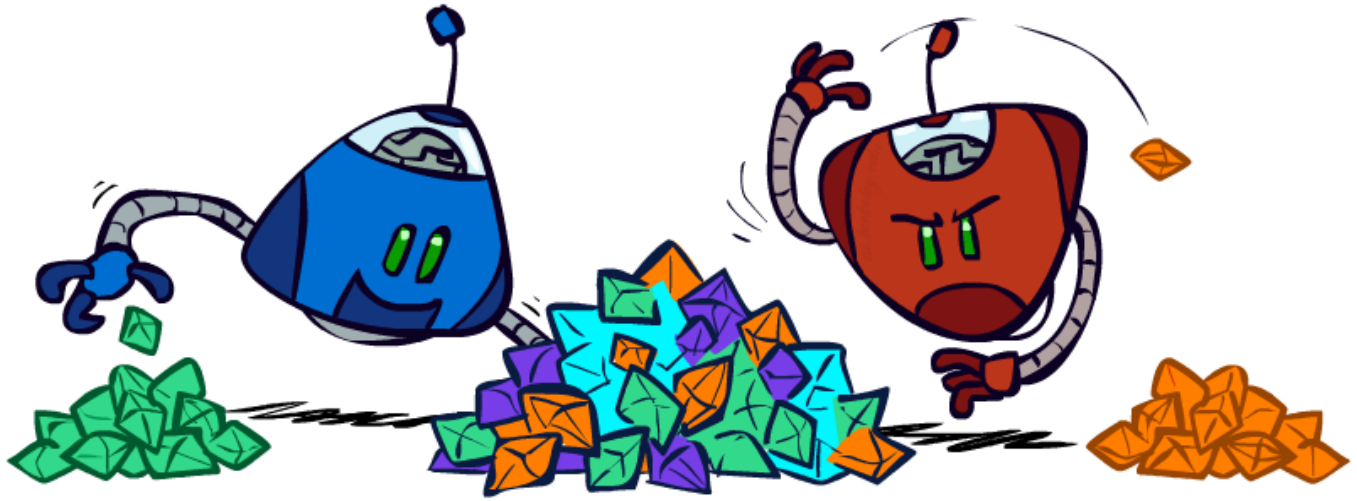
Jeux déterministiques

- Plusieurs formulation possible, on choisit:
  - Etats: S (commence a s_0)
  - Joueurs: P={1...N} (chacun attend son tour)
  - Actions: A (Dependent du joueur et de l'état)
  - Fonction de transition: SxA $\rightarrow$ S
  - Test d'arrêt: S $\rightarrow$ {t,f}
  - Utilités : SxP $\rightarrow$ R
- La solution d'un joueur est stratégie: S $\rightarrow$ A



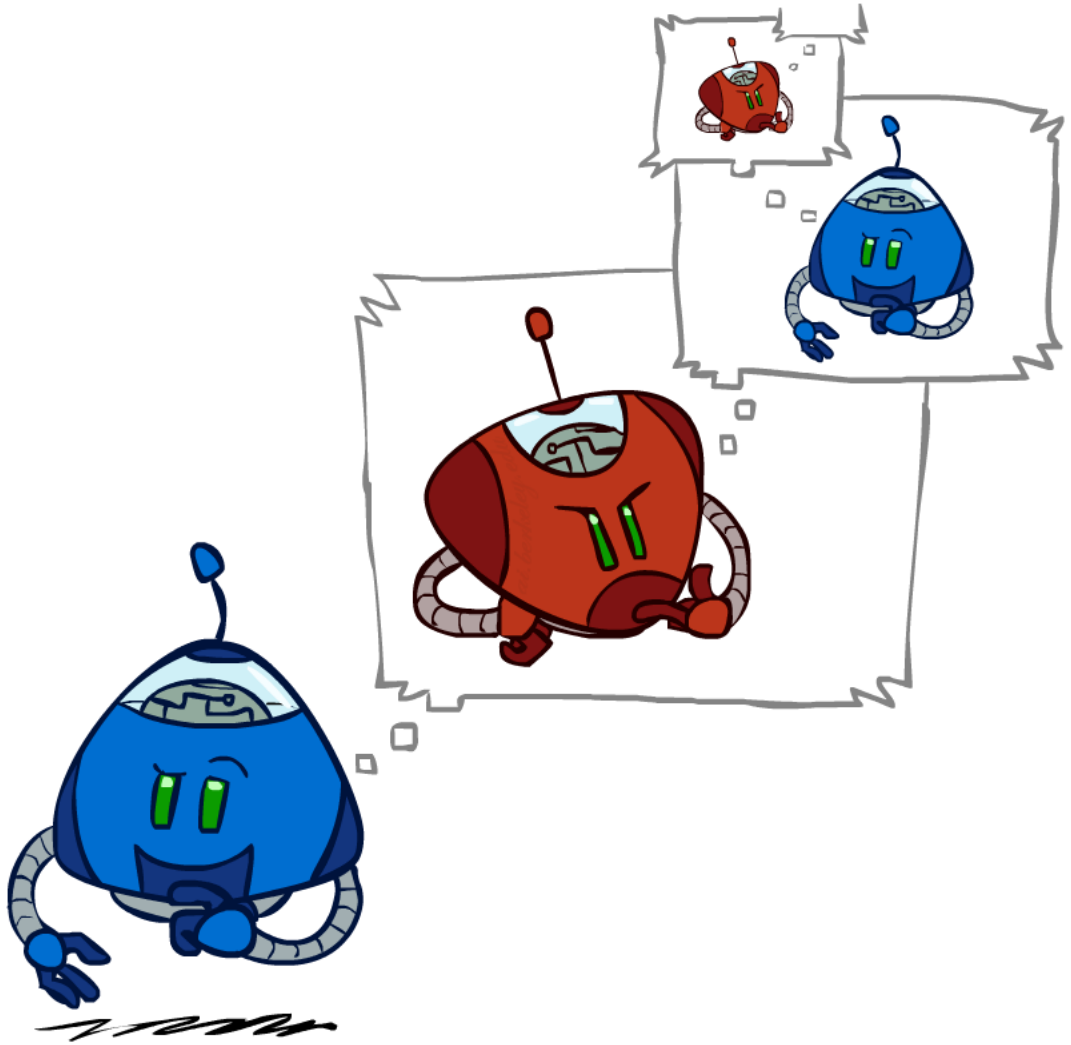
Jeux Zero-Somme



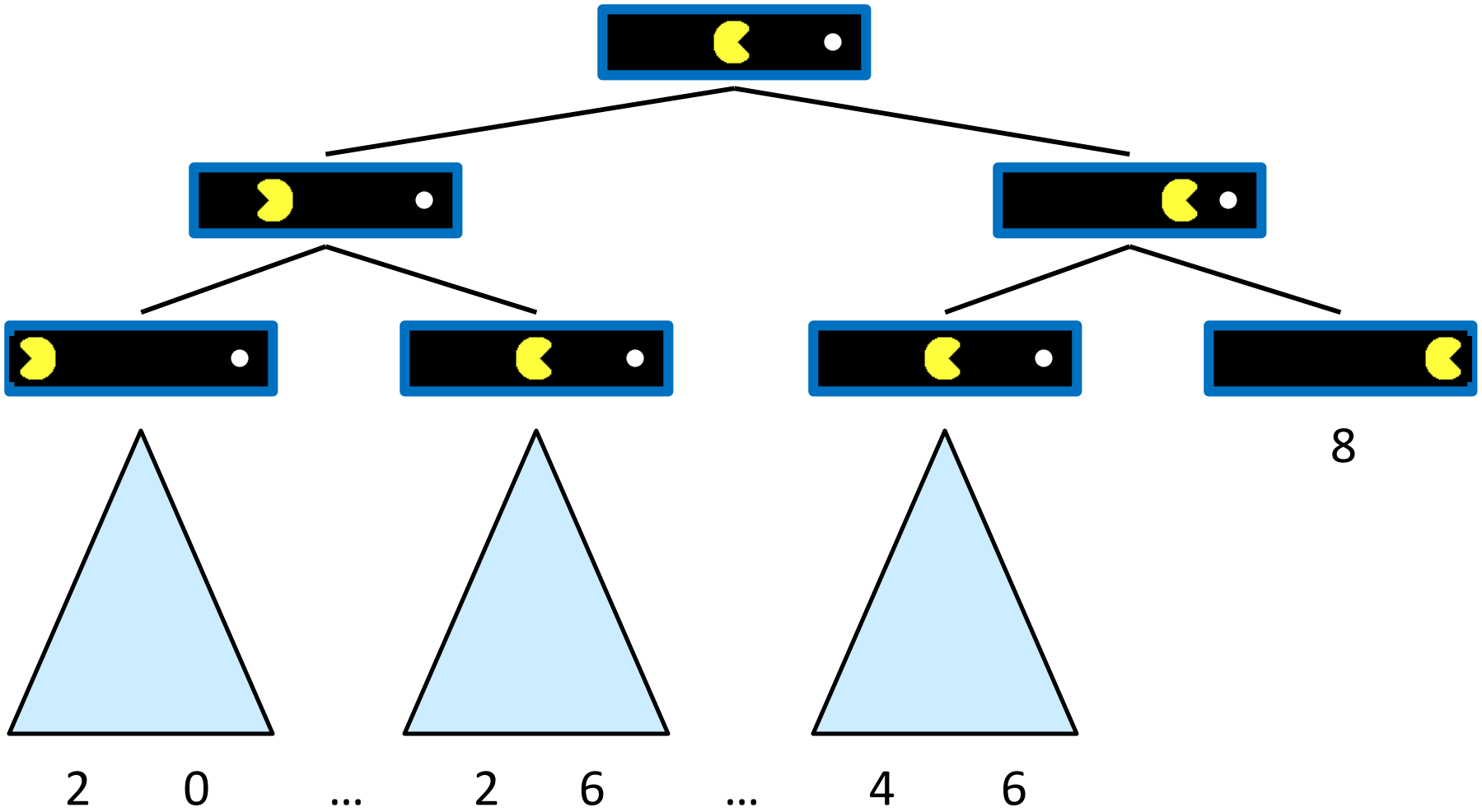


- Jeux Zero-Somme
  - Agents possèdent des utilitées **opposées**
  - Un agent essaie de maximizer et l'autre minimizer la function d'utilité
  - Compétition pure..
- Jeux généraux
  - Chaque agent possède sa **propre** function d'utilité
  - Cooperation, indifférence, competition. Tous possible.

Recherche en Adversité



Arbre d’un seul agent

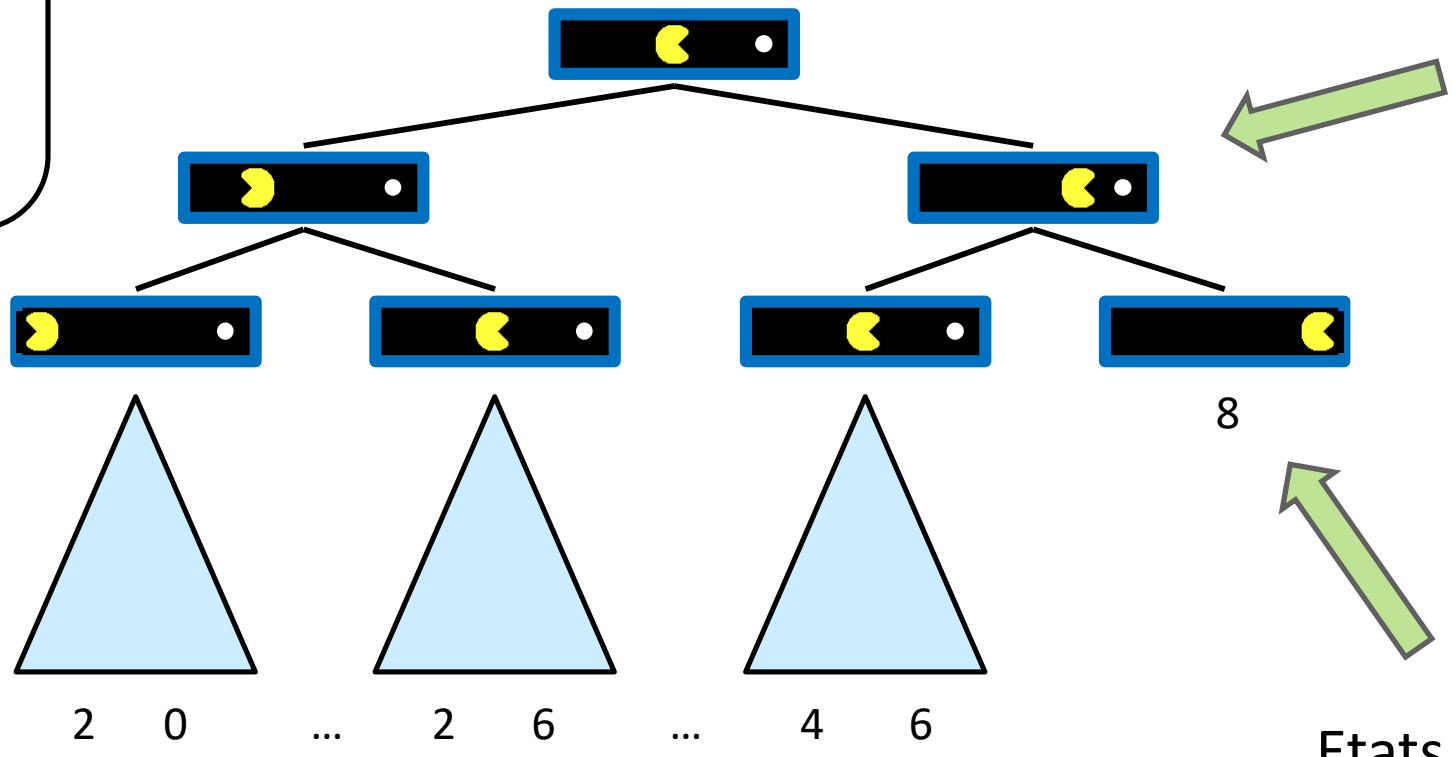


Valeur d'un état

Valeur d'un état:
La meilleure valeur possible qu'on peut achever de cet état

Etat non terminaux:

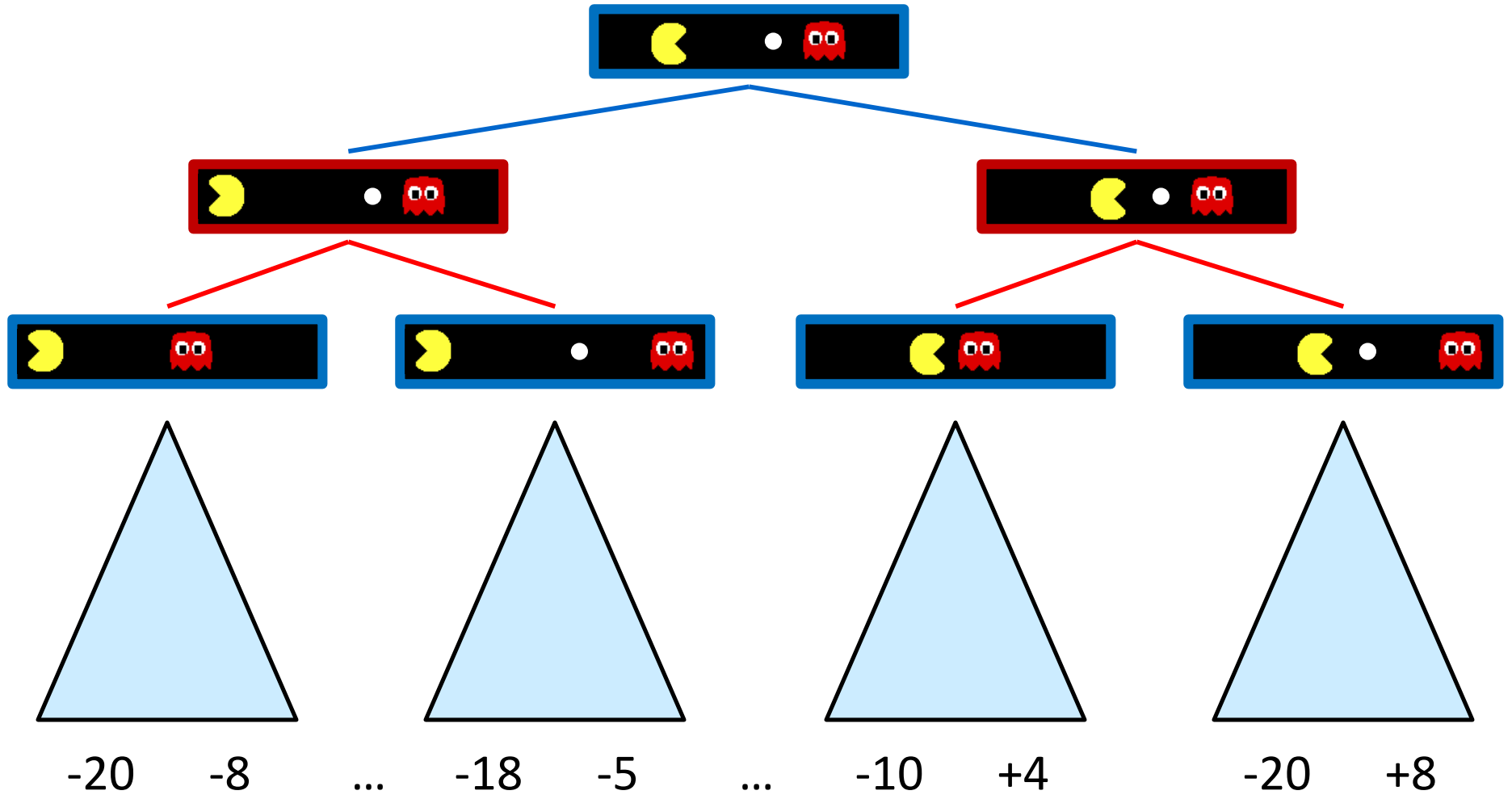
$$V(s) = \max_{s' \in desc(s)} V(s')$$



Etats terminaux:

V(s) = connu

Arbre jeux en adversité



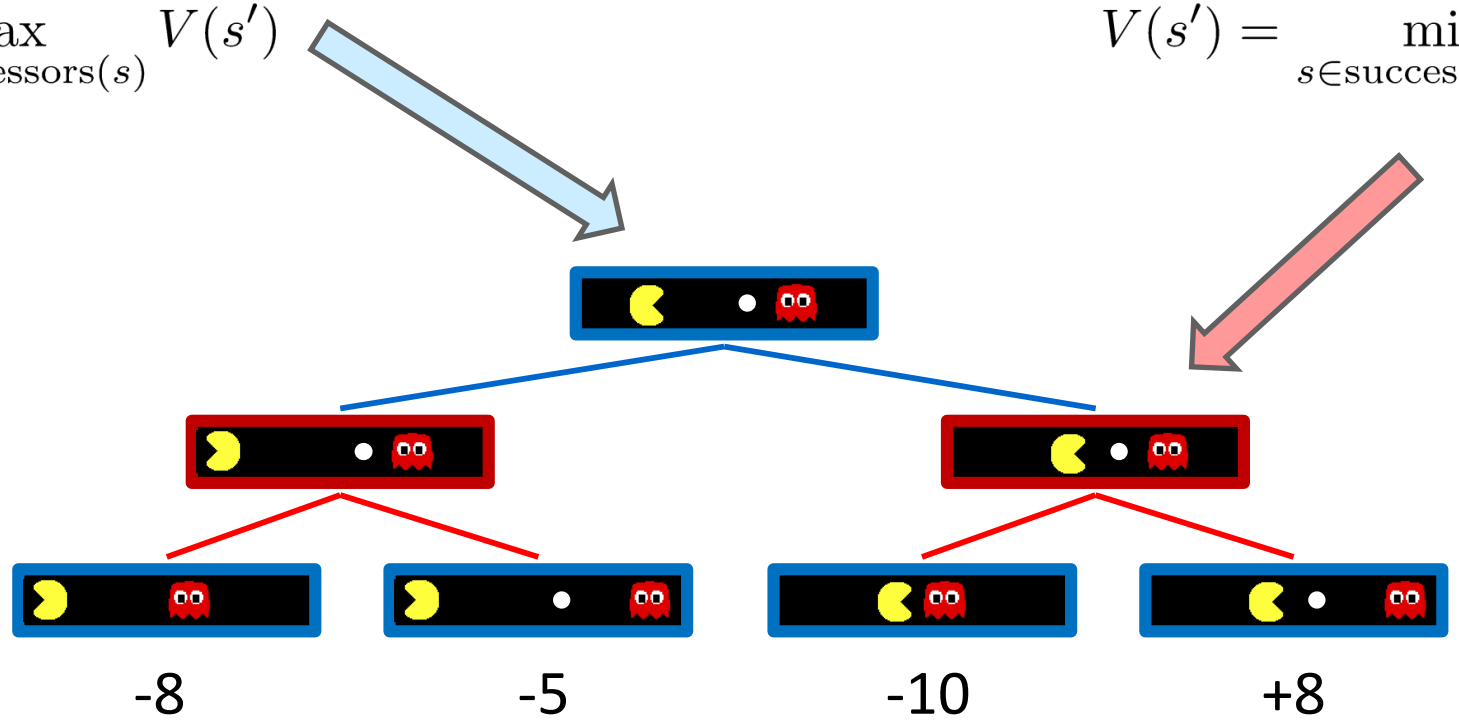
Veleurs MiniMax

Etats de l'agent controllé:

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

Etats de l'agent adverse:

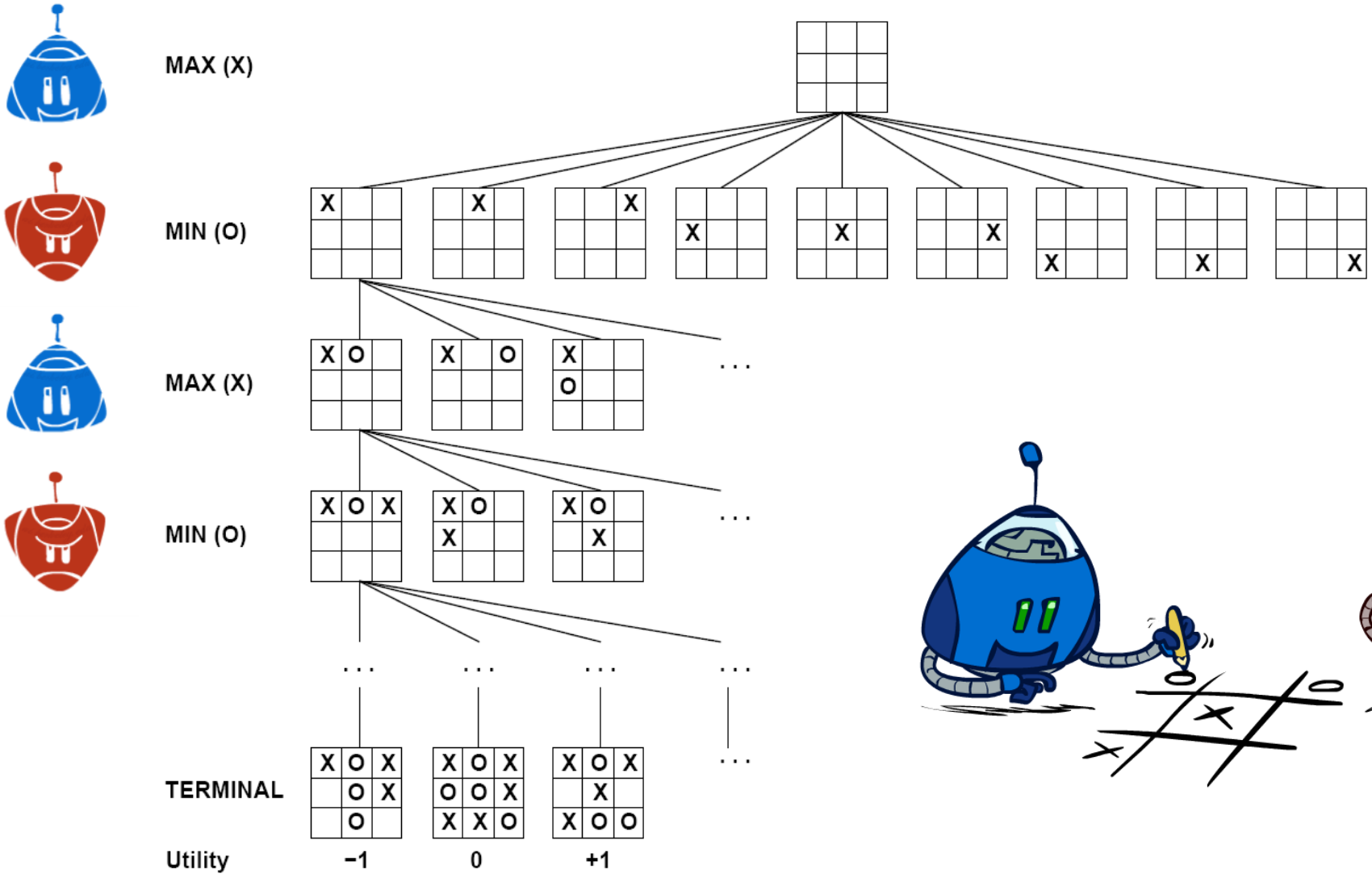
$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$



Etats terminaux:

$V(s) = connu$

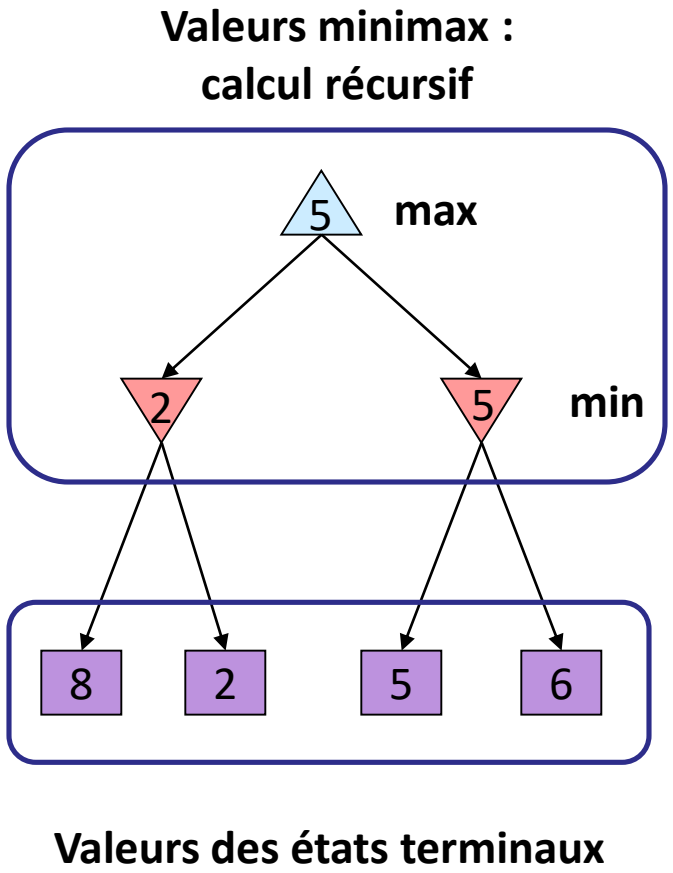
Arbre Tic-Tac-Toe



Recherche en adversité (Minimax)

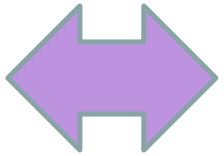
- Jeux deterministique, zero-somme:
  - Tic-tac-toe, echecs, dammes
  - Un joueur **maximise** le résultat
  - L'autre le **minimise**

- Recherche Minimax :
  - Arbre de recherche
  - Les joueurs alternent
  - Chacun calcule la valeur minimax : Meilleur comportement face à agent **rationnel.**



Implémentation Minimax

```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, min-value(successor))
    return v
```



```
def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, max-value(successor))
    return v
```

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$

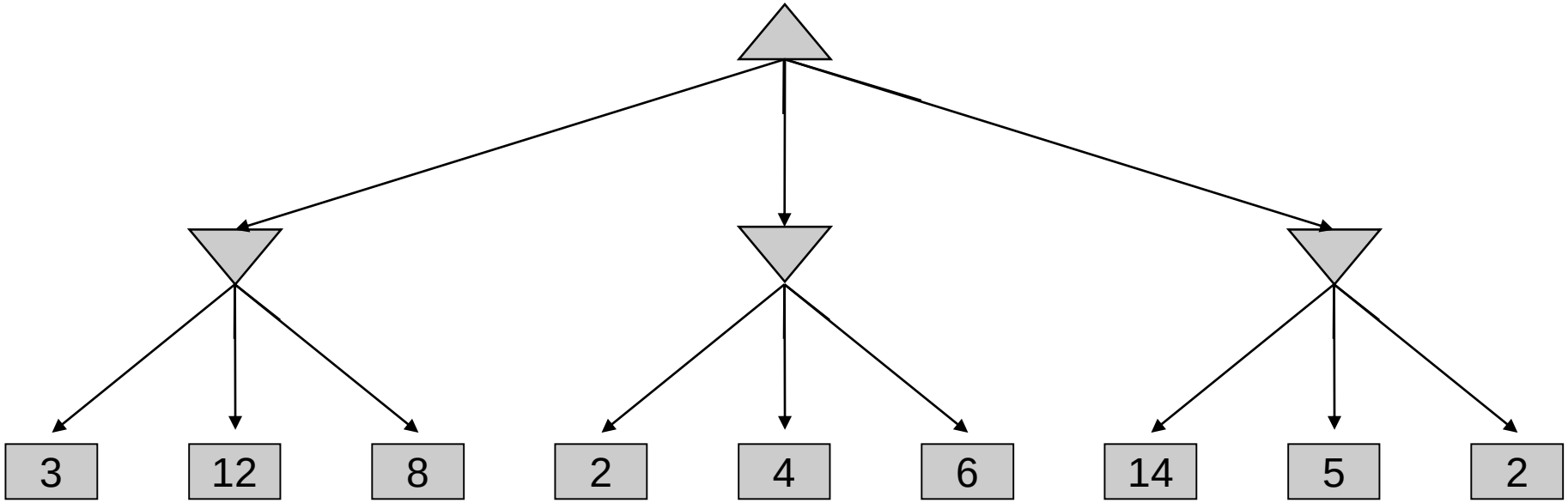
Implémentation MiniMax

```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is MIN: return min-value(state)
```

```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```

```
def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor))
    return v
```

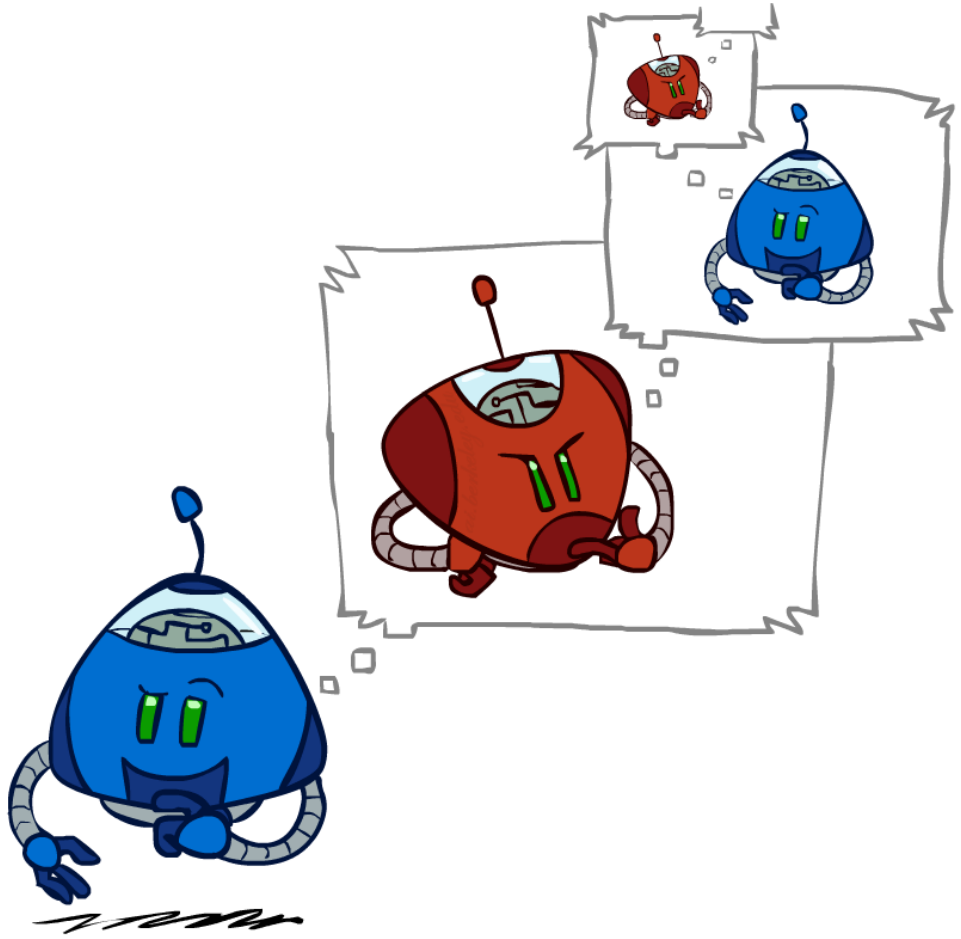
Exemple Minimax



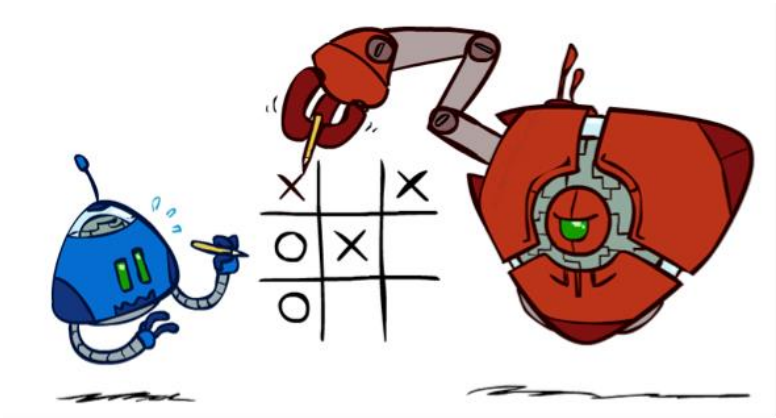
Efficacité Minimax

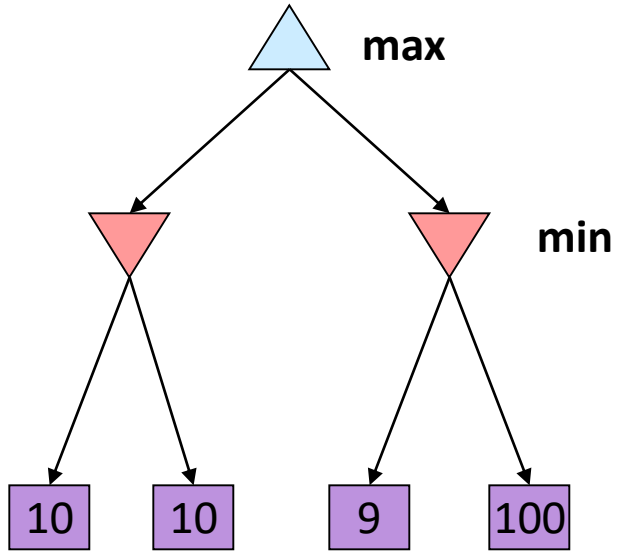
- Efficacité de minimax?
  - Comme DFS exhaustive
  - Temps: $O(b^m)$
  - Espace: O(bm)

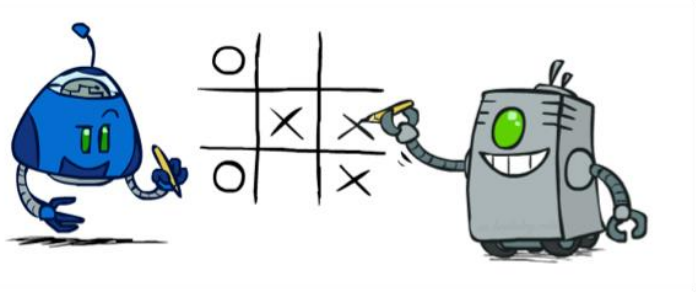
- Exemple: echecs, $b \approx 35$, $m \approx 100$
  - Exacte solution est **infaisable**
  - Mais, dois t on explorer toute l'arbre?



Propriété de Minimax

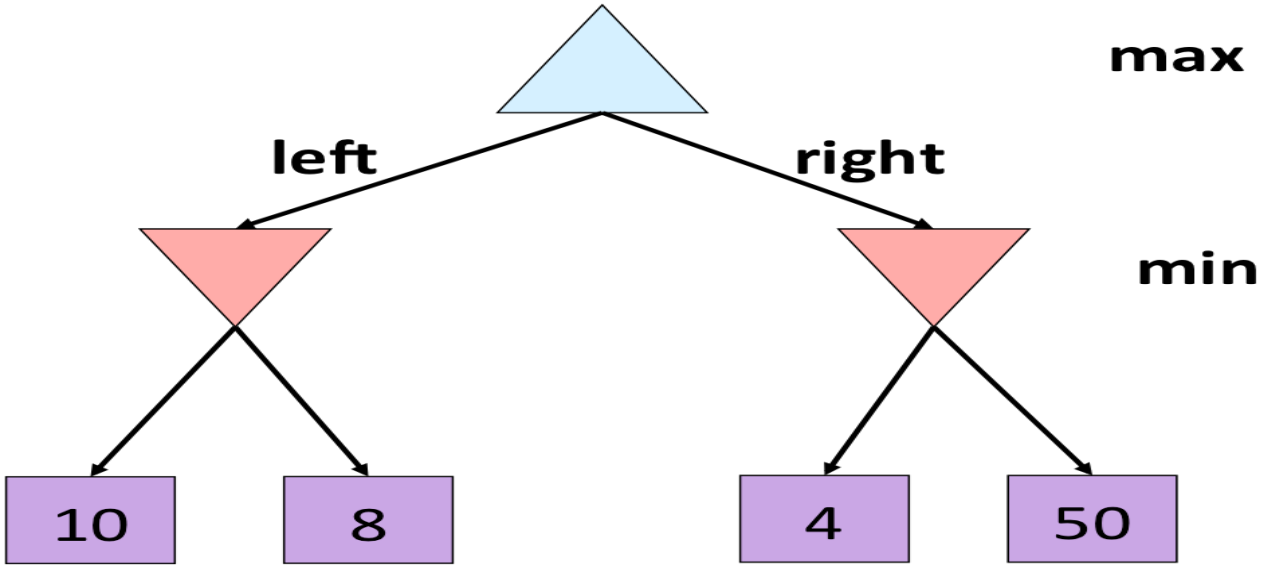






Optimal contre un joueur parfait. Mais?

Quiz: MiniMax valeur



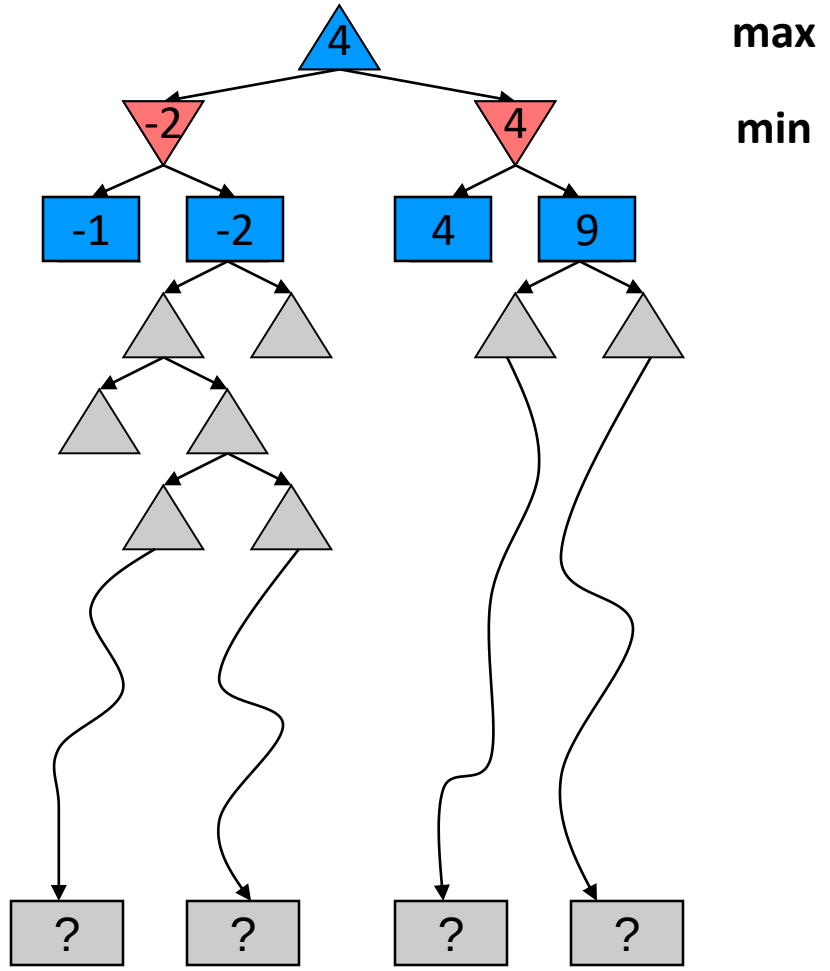
1. Quelle est la **valeur** de l'arbre du jeu ?

2. Quelle est l'**action** choisie par l'agent maximiseur selon *minimax*?

Limites des Resources

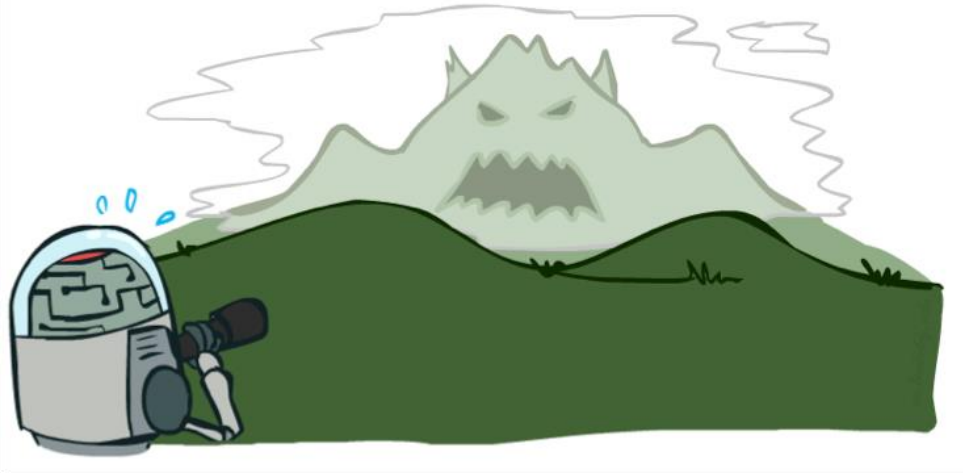
Limites des Resources

- Problème : En jeu réalistique, On ne peut pas atteindre les **feuilles** (*noeuds terminaux*)!
- Solution: Recherche avec profendeur limitée
  - Ainsi, Explorer l'arbre jusqu'à un profondeur determine.
  - Utiliser une function **d'évaluation** pour affecter des valeurs aux noeuds de l'arbre.
- Exemple:
  - Supposons qu'on possède 100 sec, et qu'on peut explorer 10K noeuds / sec
  - On peut seulement verifier 1M noeuds par coup.
  - α-β atteint une profondeur de 8. profonde pour le jeu d'échecs.
- On perd l'optimalité.

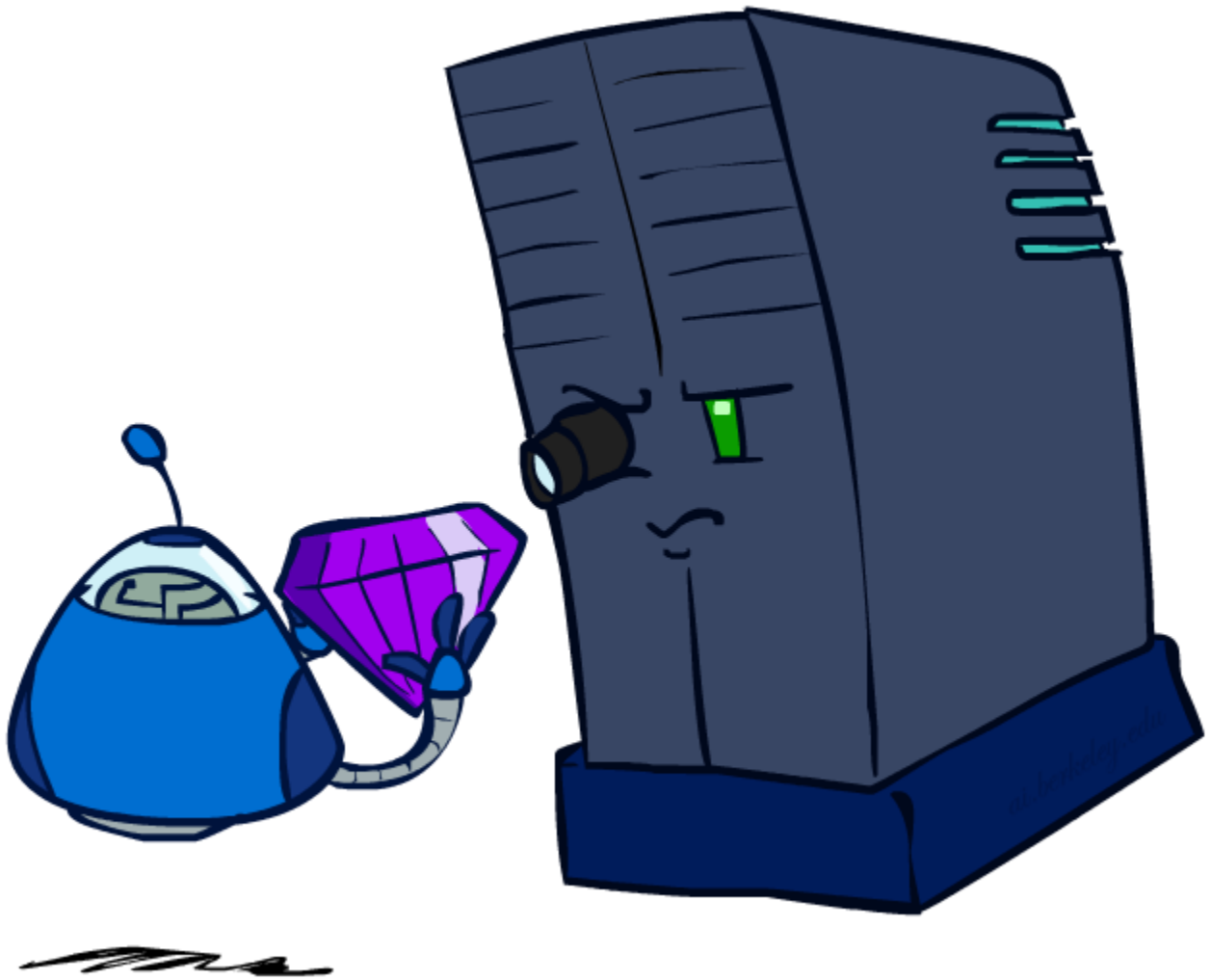


Importance de la profondeur

- Les fonctions d'évaluations sont toujours des approximations.
- Plus on approfondit les positions d'évaluation, plus la precision de la function d'évaluation n'est plus importante.
- Paradigme classique du compromise entre la precision et la profondeur.

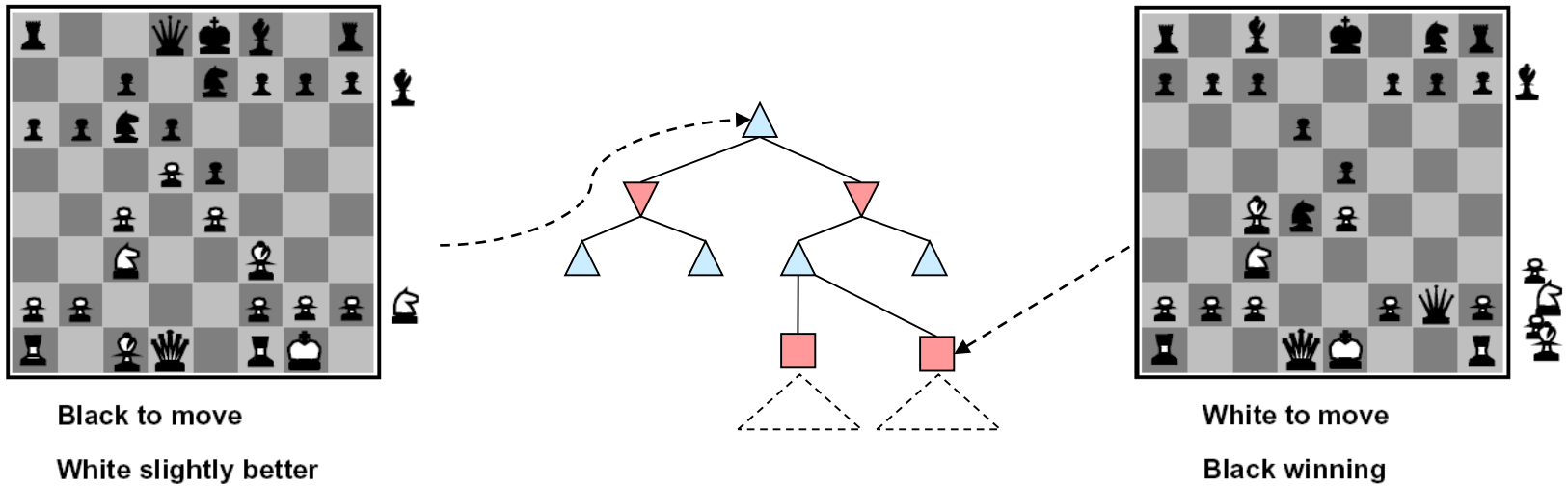


Fonctions d’evaluation



Functions d'évaluation

- Fonction utilisés pour donner un score aux noeuds non terminaux.

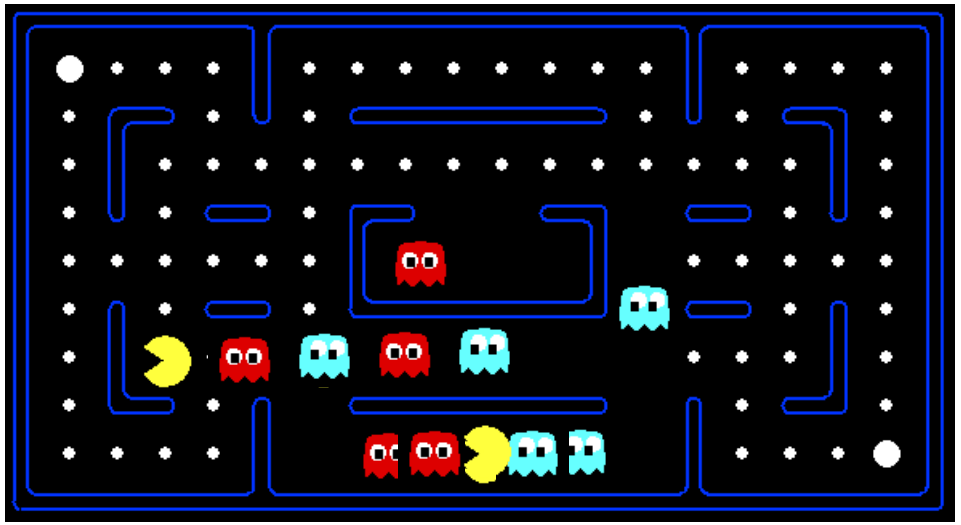


- Fonction idéale : Renvoie le score exact du noeud.
- En practique: Un ponderation des caractéristique d'un état:

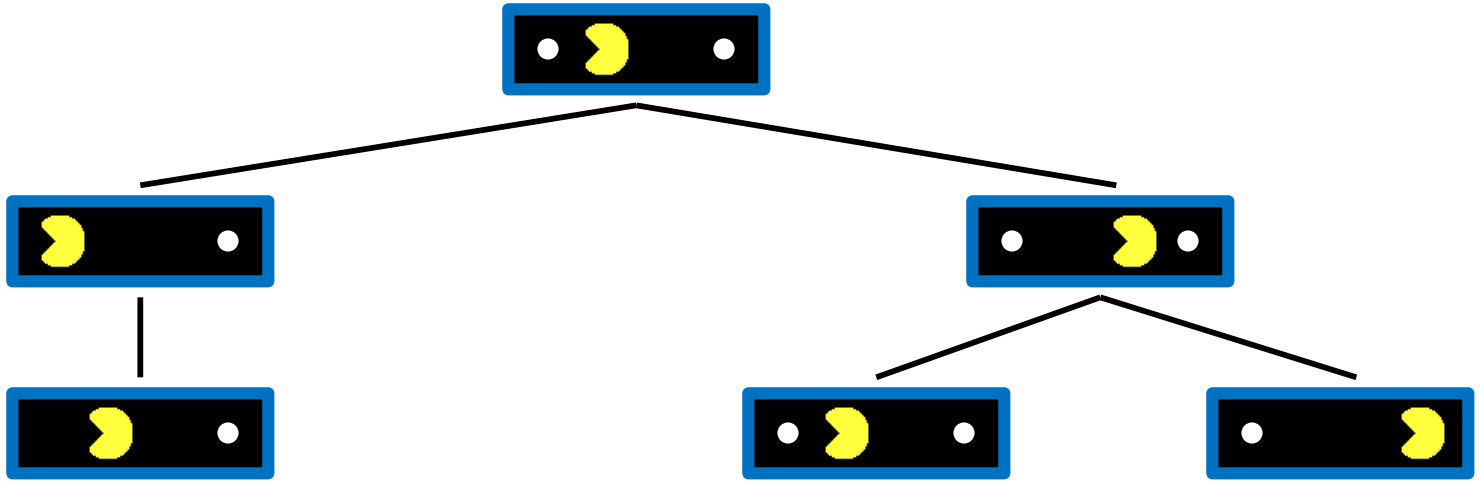
$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

- Exemple: $f_1(s)$ = (num white queens – num black queens), etc.

Evaluation pour Pacman

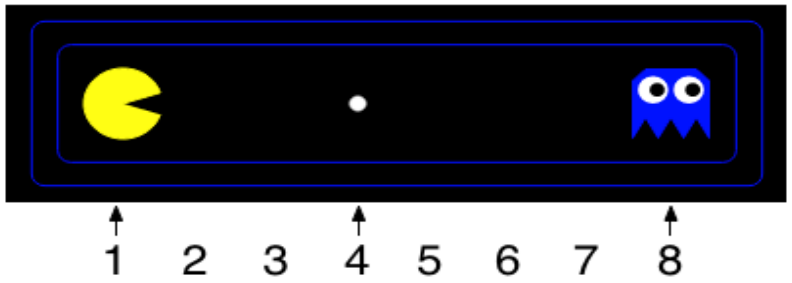


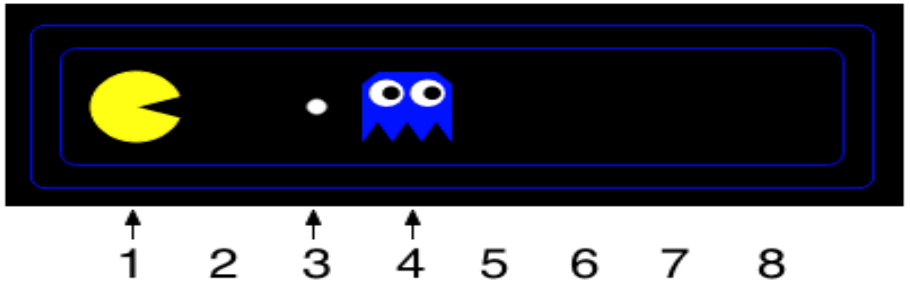
Source du problème



- Le danger d'un agent qui recalcule son plan!
  - Il sait qu'une nouriture peut être prise pour les deux directions (west, east)
  - Il sait que même en attendant il pourra prendre la nouriture (east, west)
  - Avec une profondeur 2, il ne voit pas les consequences après.
  - Ainsi, Attendre à la même valeur que se déplacer.

Quiz: Fonction d'évaluation

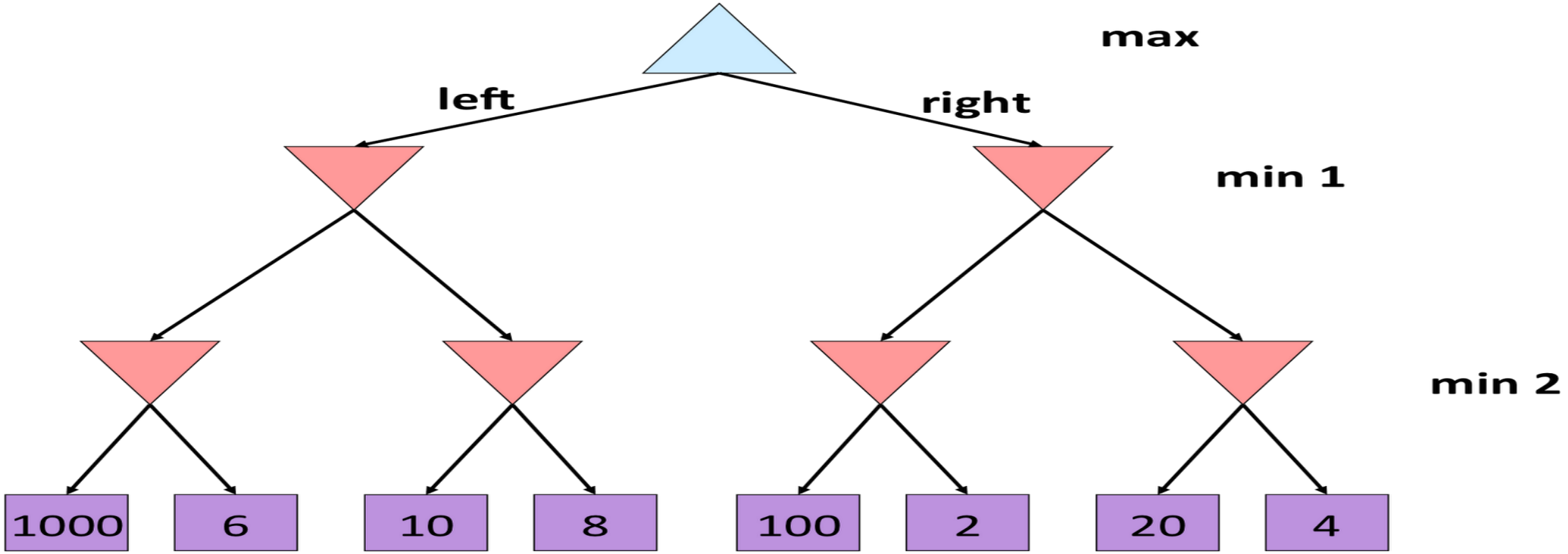




Choisir la fonction d'évaluation qui favorise la situation à **Gauche.**

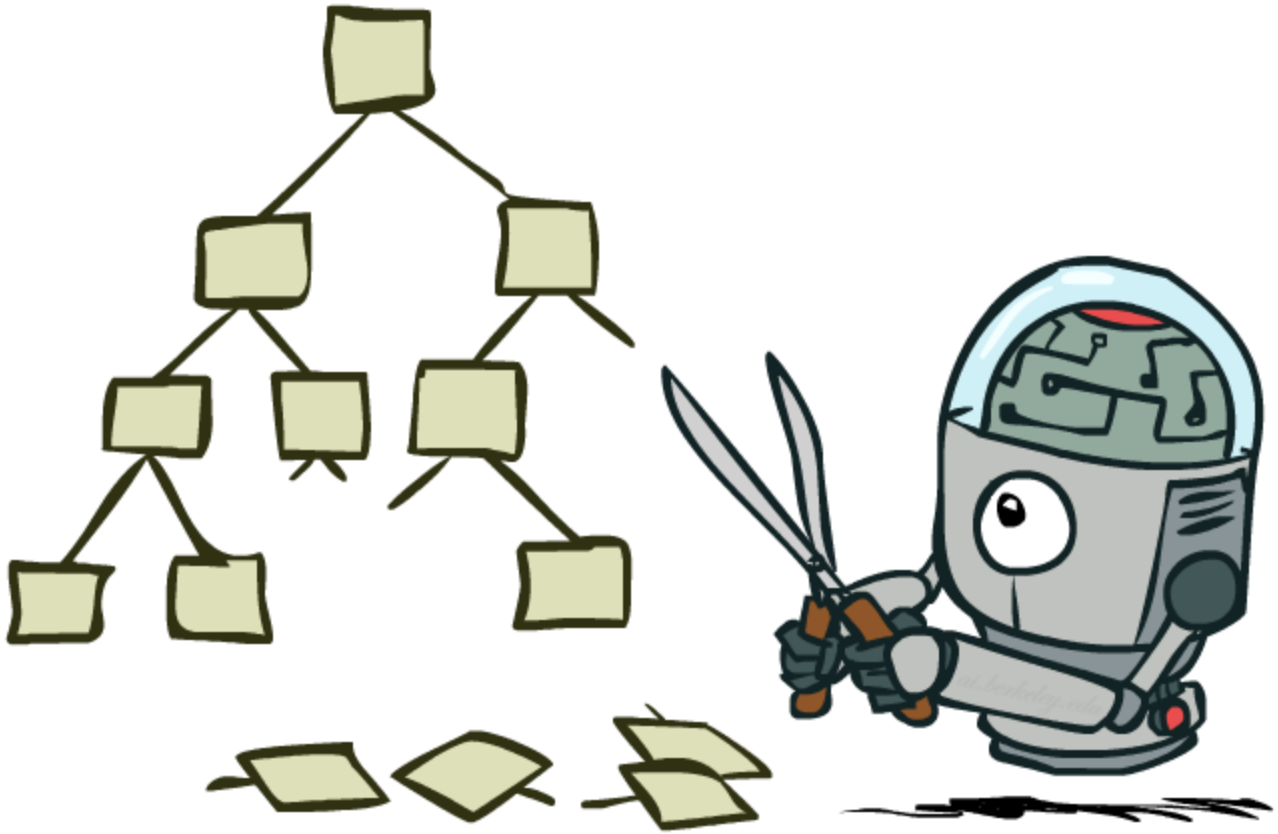
- ❑ 1/(distance de pacman à la nourriture la plus proche)
- ❑ Distance de pacman du plus proche fontôme.
- ❑ Distance de pcman du plus proche fontôme + 1/(distance pacman de la nourriture la plus proche)
- ❑ Distance de pcman du plus proche fontôme + 1000/(distance pacman de la nourriture la plus proche)

Quiz: Collaboration

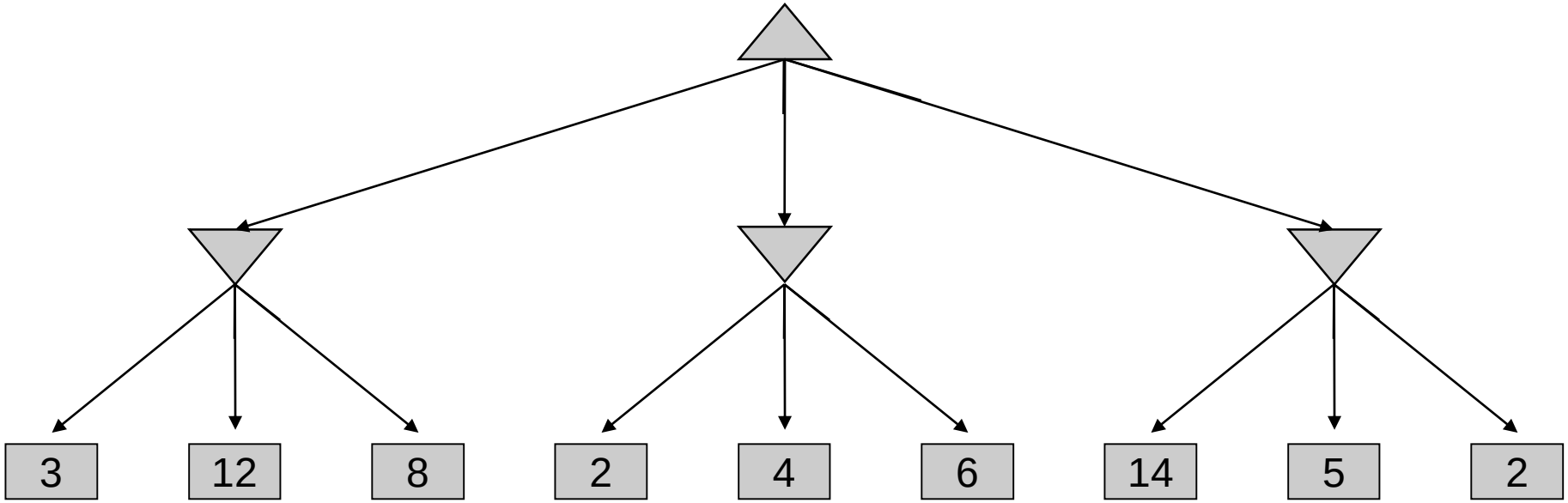


1. Donner la valeur minimax de cet arbre
2. Quelle est l'action que le maximiseur doit prendre?

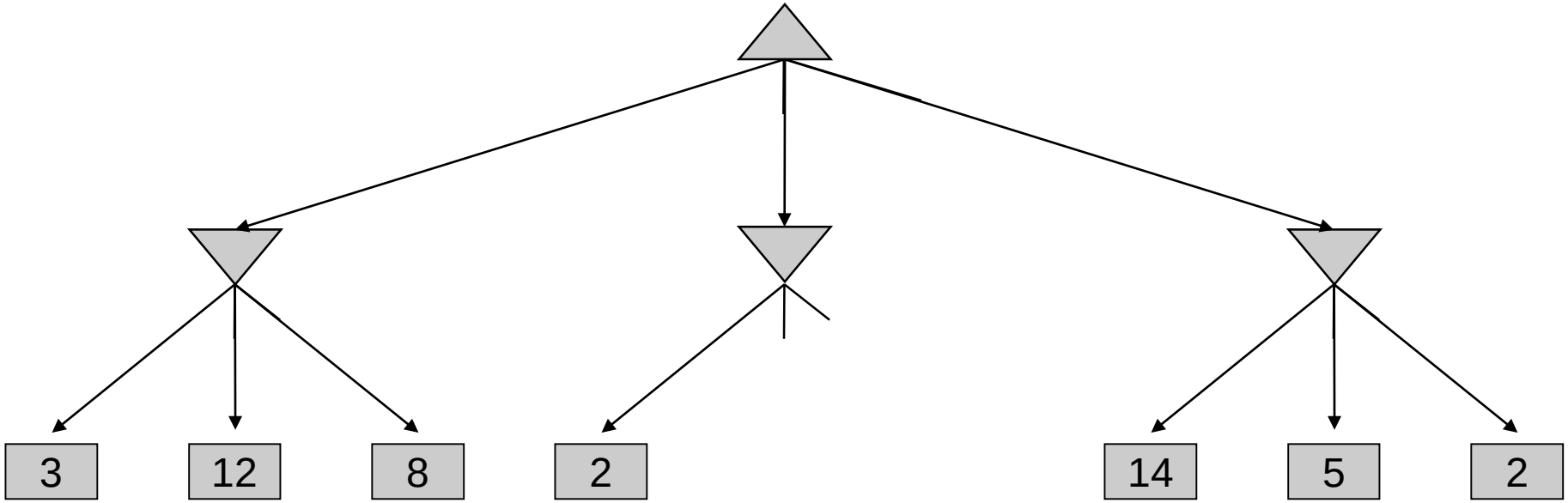
Elagage de l’arbre de recherche



Minimax Example

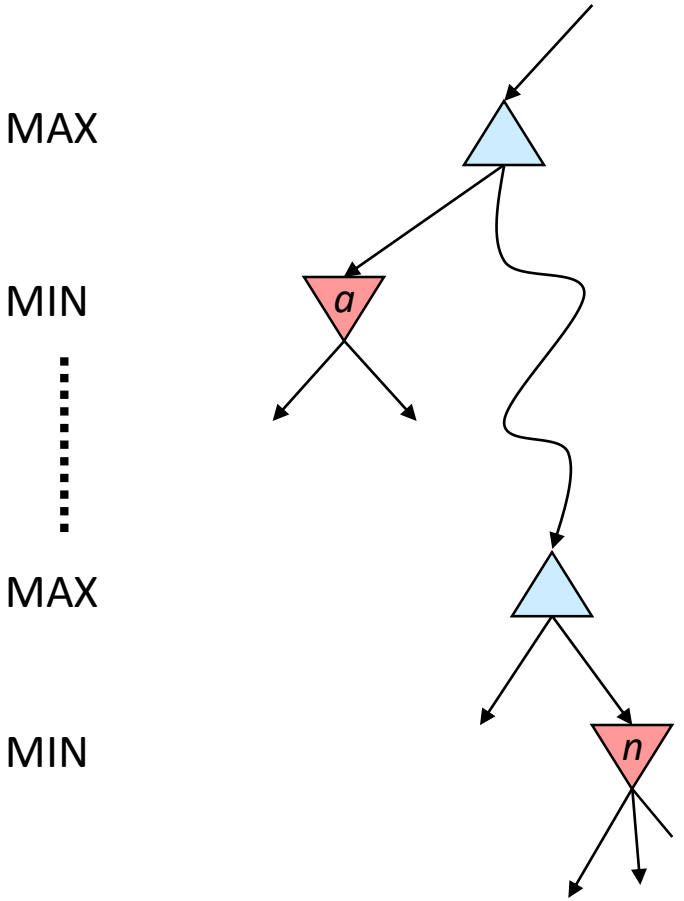


Elagage Minimax



Elagage Alpha-Beta

- Configuration générale (MIN version)
  - On calcule la valeur minimal dans un neoud **n**
  - Nous itérons dans les fils de **n**.
  - Qui va utilizer la valeur de ***n***? MAX
  - Soit ***a*** la meilleur valeur de MAX qu'il peut otenir dans le chemin de l'arbre.
  - Ainsi dès que ***n deviens inférieur à a***, MAX va l'**éviter,** alors pas la peinde de continuer le calcul dans **n.**



- La version de MAX est symmétrique.

Implémentation Alpha-Beta

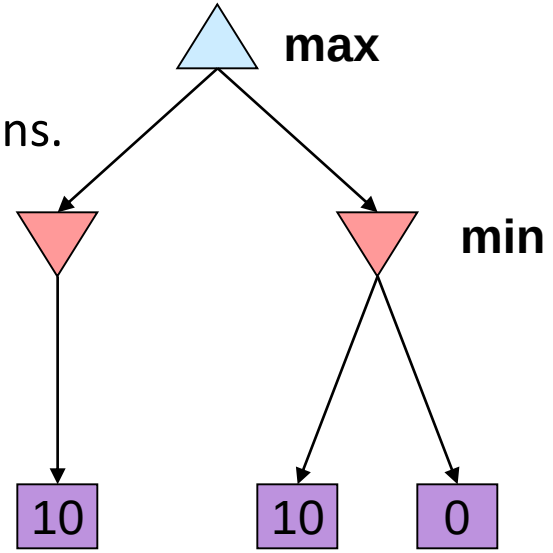
α: MAX'meilleure option
β: MIN meilleure option

```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v ≥ β return v
        α = max(α, v)
    return v
```

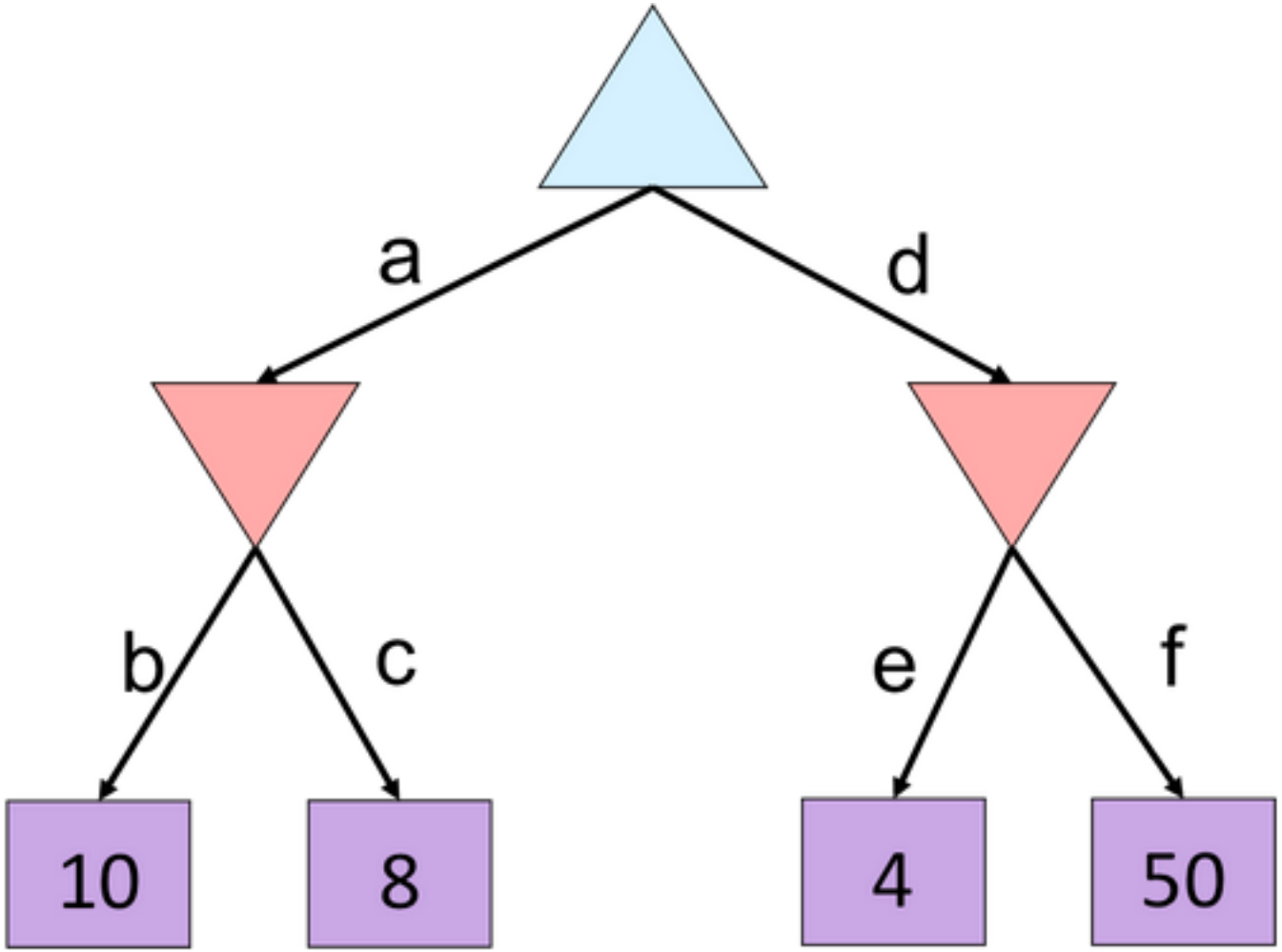
```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```

Propriétés Elagage Alpha-Beta

- L'élagage n'as pas d'effet sur les valeurs calculées dans la **racine**!
- Valeur dans les noeuds indermédiare ne le sont pas forcément.
  - Important: les fis de la racine peuvent avoir une valeur erronée.
  - Ainsi, cette version naïve de alpha-beta ne permet pas le choix des actions.
- L'ordre des fils affects l'éfficacité de l'élagage.
- Avec "un ordre parfait":
  - Complexité devient $O(b^{m/2})$
  - Doubles alors la profondeur!



Quiz: Alpha-Beta



Alpha-Beta Quiz 2