Intelligence Artificielle

basé sur "Artificial Intelligence : A modern approach" de Russel et Nowig

ENSISA 2A

Jonathan Weber

Hiver 2021

Jeux

Plan du chapitre

Jeux

Définition

Jeux

- Environnement multi-agents

Jeux

- Environnement multi-agents concurrents

Jeux

- Environnement multi-agents concurrents
- Il y a un ou plusieurs adversaires jouant contre nous

Jeux

- Environnement multi-agents concurrents
- Il y a un ou plusieurs adversaires jouant contre nous
- Jeu vs. recherche :

Jeux

- Environnement multi-agents concurrents
- Il y a un ou plusieurs adversaires jouant contre nous
- Jeu vs. recherche :
  - Solution optimale n'est pas une suite d'actions mais une stratégie

Jeux

- Environnement multi-agents concurrents
- Il y a un ou plusieurs adversaires jouant contre nous
- Jeu vs. recherche :
  - Solution optimale n'est pas une suite d'actions mais une stratégie
  - Si opposant fait a alors faire b sinon si opposant fait c alors faire d, ...

Jeux

- Environnement multi-agents concurrents
- Il y a un ou plusieurs adversaires jouant contre nous
- Jeu vs. recherche :
  - Solution optimale n'est pas une suite d'actions mais une stratégie
  - Si opposant fait a alors faire b sinon si opposant fait c alors faire d, ...
- Fastidieux et fragile si codé en dur

Jeux

- Environnement multi-agents concurrents
- Il y a un ou plusieurs adversaires jouant contre nous
- Jeu vs. recherche :
  - Solution optimale n'est pas une suite d'actions mais une stratégie
  - Si opposant fait a alors faire b sinon si opposant fait c alors faire d, ...
- Fastidieux et fragile si codé en dur
  - par exemple en utilisant des règles

Jeux

- Environnement multi-agents concurrents
- Il y a un ou plusieurs adversaires jouant contre nous
- Jeu vs. recherche :
  - Solution optimale n'est pas une suite d'actions mais une stratégie
  - Si opposant fait a alors faire b sinon si opposant fait c alors faire d, ...
- Fastidieux et fragile si codé en dur
  - par exemple en utilisant des règles
- Bonne nouvelle : jeux sont modélisés comme des problèmes de recherche et utilisent des heuristiques

Intérêt des jeux pour l'IA

- Les jeux sont très important en IA

Intérêt des jeux pour l'IA

- Les jeux sont très important en IA
- Ils sont difficiles à résoudre

Intérêt des jeux pour l'IA

- Les jeux sont très important en IA
- Ils sont difficiles à résoudre
- Ils impliquent de prendre des décisions même quand la décision optimale n'est pas atteignable

Intérêt des jeux pour l'IA

- Les jeux sont très important en IA
- Ils sont difficiles à résoudre
- Ils impliquent de prendre des décisions même quand la décision optimale n'est pas atteignable
- Ils impliquent des problèmes combinatoires :

Intérêt des jeux pour l'IA

- Les jeux sont très important en IA
- Ils sont difficiles à résoudre
- Ils impliquent de prendre des décisions même quand la décision optimale n'est pas atteignable
- Ils impliquent des problèmes combinatoires :
  - Échecs = facteur de branchement moyen de 35

Intérêt des jeux pour l'IA

- Les jeux sont très important en IA
- Ils sont difficiles à résoudre
- Ils impliquent de prendre des décisions même quand la décision optimale n'est pas atteignable
- Ils impliquent des problèmes combinatoires :
  - Échecs = facteur de branchement moyen de 35
  - Une partie = 50 mouvements par joueur en moyenne

Intérêt des jeux pour l'IA

- Les jeux sont très important en IA
- Ils sont difficiles à résoudre
- Ils impliquent de prendre des décisions même quand la décision optimale n'est pas atteignable
- Ils impliquent des problèmes combinatoires :
  - Échecs = facteur de branchement moyen de 35
  - Une partie = 50 mouvements par joueur en moyenne
  - Soit 35^{100} ($= 10^{154}$) nœuds possibles (rappel : univers $\simeq 10^{80}$ atomes)

Intérêt des jeux pour l'IA

- Les jeux sont très important en IA
- Ils sont difficiles à résoudre
- Ils impliquent de prendre des décisions même quand la décision optimale n'est pas atteignable
- Ils impliquent des problèmes combinatoires :
  - Échecs = facteur de branchement moyen de 35
  - Une partie = 50 mouvements par joueur en moyenne
  - Soit 35^{100} ($= 10^{154}$) nœuds possibles (rappel : univers $\simeq 10^{80}$ atomes)
  - Mais, il n'y a "que" 10^{40} nœuds possibles environ

Intérêt des jeux pour l'IA

- Les jeux sont très important en IA
- Ils sont difficiles à résoudre
- Ils impliquent de prendre des décisions même quand la décision optimale n'est pas atteignable
- Ils impliquent des problèmes combinatoires :
  - Échecs = facteur de branchement moyen de 35
  - Une partie = 50 mouvements par joueur en moyenne
  - Soit 35^{100} ($= 10^{154}$) nœuds possibles (rappel : univers $\simeq 10^{80}$ atomes)
  - Mais, il n'y a "que" 10^{40} nœuds possibles environ
- Ils doivent prendre en compte l'imprévisibilité de l'adversaire

Intérêt des jeux pour l'IA

- Les jeux sont très important en IA
- Ils sont difficiles à résoudre
- Ils impliquent de prendre des décisions même quand la décision optimale n'est pas atteignable
- Ils impliquent des problèmes combinatoires :
  - Échecs = facteur de branchement moyen de 35
  - Une partie = 50 mouvements par joueur en moyenne
  - Soit 35^{100} ($= 10^{154}$) nœuds possibles (rappel : univers $\simeq 10^{80}$ atomes)
  - Mais, il n'y a "que" 10^{40} nœuds possibles environ
- Ils doivent prendre en compte l'imprévisibilité de l'adversaire
- Si limite de temps : peut-être impossible d'atteindre solution optimale donc nécessité de l'approximer

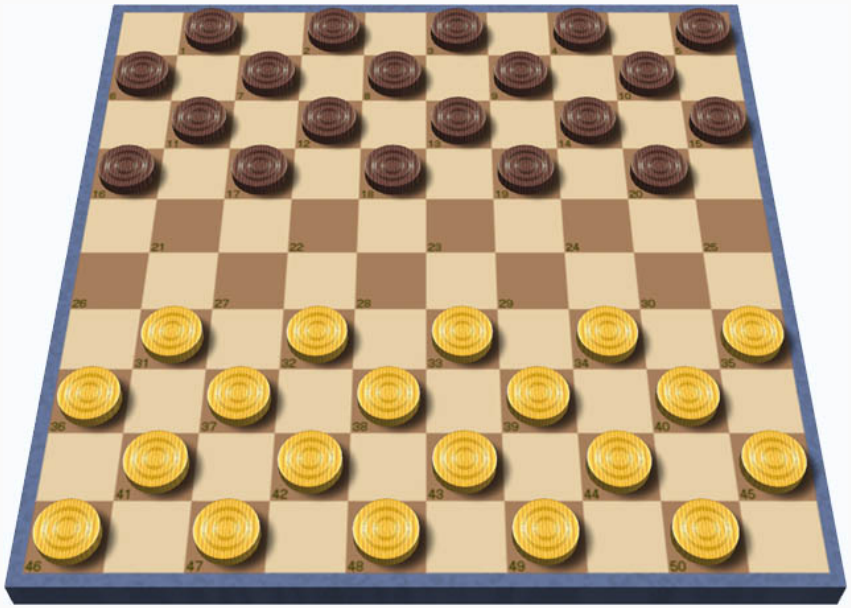
Intérêt des jeux pour l'IA

- Les jeux sont très important en IA
- Ils sont difficiles à résoudre
- Ils impliquent de prendre des décisions même quand la décision optimale n'est pas atteignable
- Ils impliquent des problèmes combinatoires :
  - Échecs = facteur de branchement moyen de 35
  - Une partie = 50 mouvements par joueur en moyenne
  - Soit 35^{100} ($= 10^{154}$) nœuds possibles (rappel : univers $\simeq 10^{80}$ atomes)
  - Mais, il n'y a "que" 10^{40} nœuds possibles environ
- Ils doivent prendre en compte l'imprévisibilité de l'adversaire
- Si limite de temps : peut-être impossible d'atteindre solution optimale donc nécessité de l'approximer

$\Rightarrow$ Les jeux sont à l'IA ce que les grands prix sont à la conception des voitures

Exemple : Dames

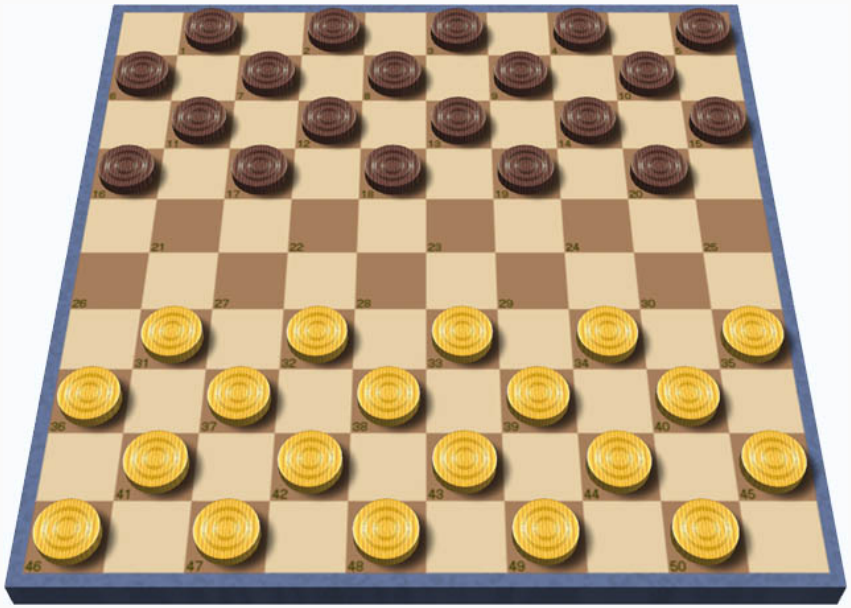
- Chinook a mis fin à 40 ans de règne du champion mondial Marion Tinsley en 1994



©Michel32NI, Wikipedia

Exemple : Dames

- Chinook a mis fin à 40 ans de règne du champion mondial Marion Tinsley en 1994
- Utilisait une base de données définissant le coup parfait pour toutes les configurations impliquant 8 pièces ou moins sur le plateau

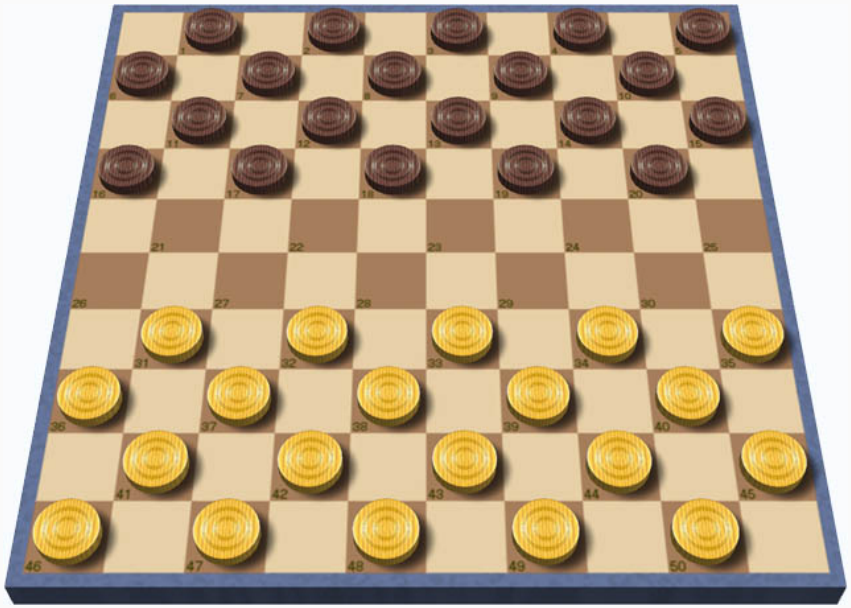


©Michel32NI, Wikipedia

Exemple : Dames

- ▷ Chinook a mis fin à 40 ans de règne du champion mondial Marion Tinsley en 1994
- ▷ Utilisait une base de données définissant le coup parfait pour toutes les configurations impliquant 8 pièces ou moins sur le plateau

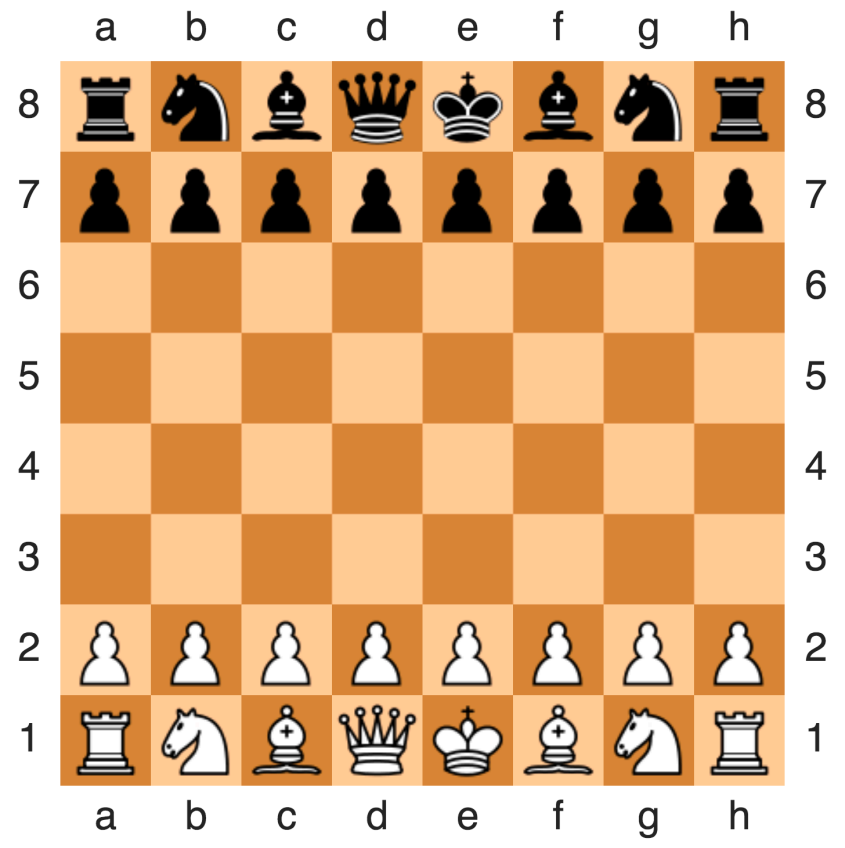
⇒ 443 748 401 247 configurations



©Michel32NI, Wikipedia

Exemple : Échecs

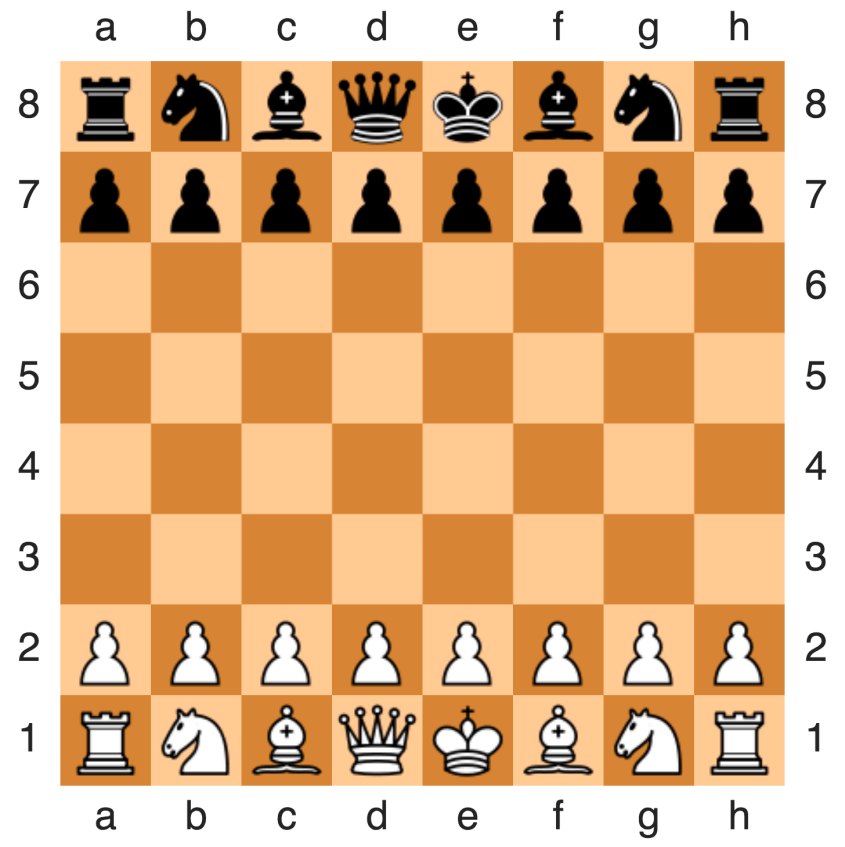
- Deep Blue bat Gary Kasparov, le champion du monde, en 6 parties en 1997



©Wikipedia

Exemple : Échecs

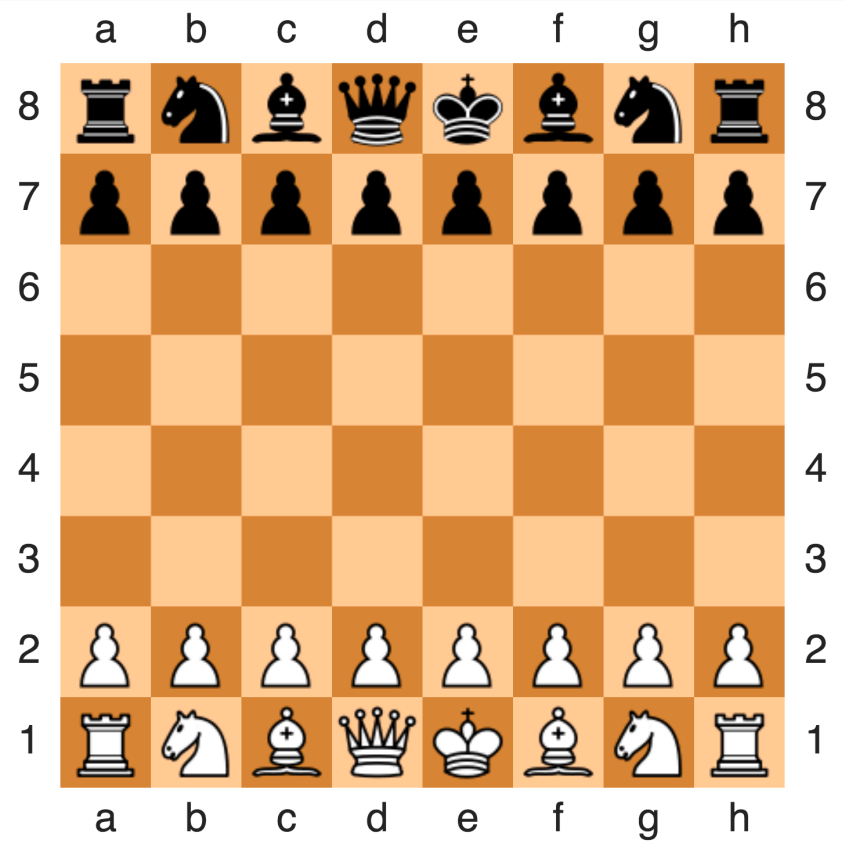
- Deep Blue bat Gary Kasparov, le champion du monde, en 6 parties en 1997
- Deep Fritz bat 4-2 Vladmir Kramnik, le champion du monde, en 2006



©Wikipedia

Exemple : Échecs

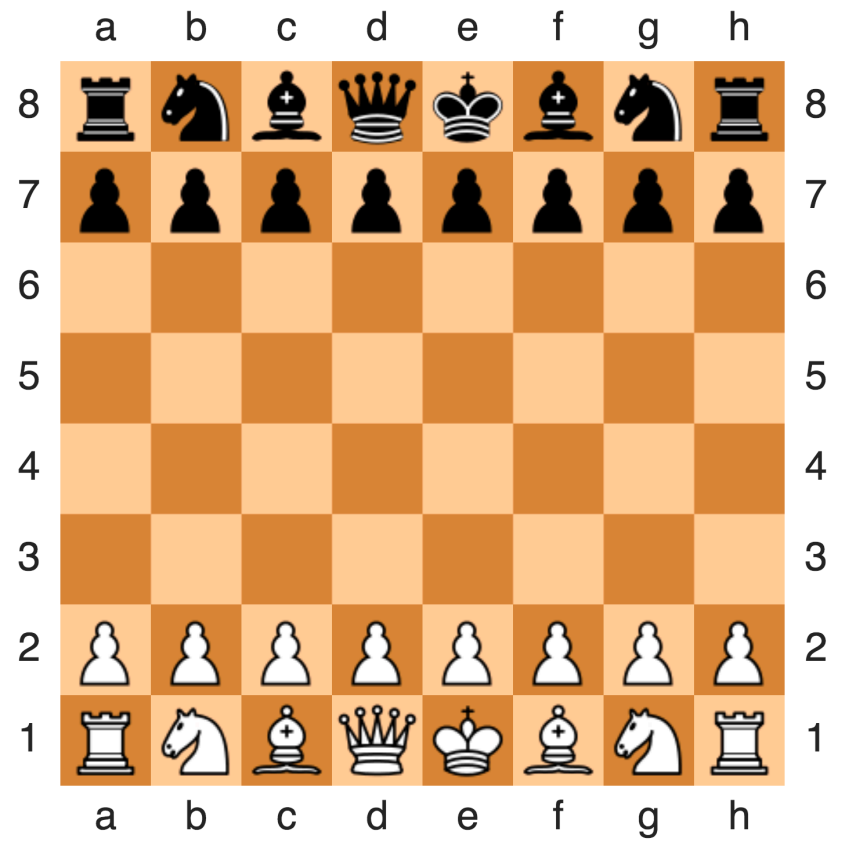
- Deep Blue bat Gary Kasparov, le champion du monde, en 6 parties en 1997
- Deep Fritz bat 4-2 Vladmir Kramnik, le champion du monde, en 2006
  - Deep Fritz tournait sur un PC avec 2 Xeon bi-cœur 5160 à 3,0 GHz



©Wikipedia

Exemple : Échecs

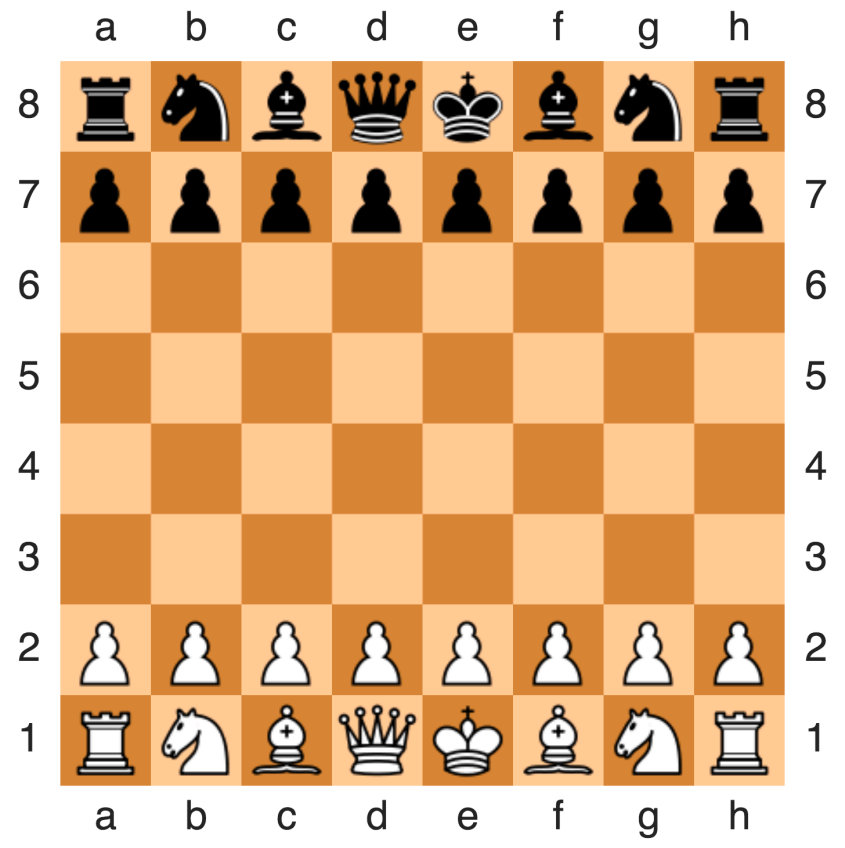
- Deep Blue bat Gary Kasparov, le champion du monde, en 6 parties en 1997
- Deep Fritz bat 4-2 Vladmir Kramnik, le champion du monde, en 2006
  - Deep Fritz tournait sur un PC avec 2 Xeon bi-cœur 5160 à 3,0 GHz
  - évaluation de 8 millions de coups par seconde



©Wikipedia

Exemple : Échecs

- Deep Blue bat Gary Kasparov, le champion du monde, en 6 parties en 1997
- Deep Fritz bat 4-2 Vladmir Kramnik, le champion du monde, en 2006
  - Deep Fritz tournait sur un PC avec 2 Xeon bi-cœur 5160 à 3,0 GHz
  - évaluation de 8 millions de coups par seconde
  - grâce aux heuristiques, profondeur évaluée de 17 ou 18 coups



©Wikipedia

Exemple : Go

- $b > 300$!

Exemple : Go

- $b > 300$!
- En 2016, AlphaGo bat Fan Hui, champion européen et Lee Sedol, meilleur joueur mondial (de 2001 à 2010)

Exemple : Go

- $b > 300$!
- En 2016, AlphaGo bat Fan Hui, champion européen et Lee Sedol, meilleur joueur mondial (de 2001 à 2010)
- Pour la recherche en IA, c'est la fin de l'intérêt des jeux de plateau

Exemple : Go

- $b > 300$!
- En 2016, AlphaGo bat Fan Hui, champion européen et Lee Sedol, meilleur joueur mondial (de 2001 à 2010)
- Pour la recherche en IA, c'est la fin de l'intérêt des jeux de plateau
- L'intérêt se porte maintenant sur les jeux vidéo comme Starcraft 2

Les types de jeux

information	déterministe	hasard
totale	échecs, dames, go, othello, ...	backgammon, monopoly, ...
partielle	bataille navale, ...	bridge, poker, scrabble, ...

Les types de jeux

information	déterministe	hasard
totale	échecs, dames, go, othello, ...	backgammon, monopoly, ...
partielle	bataille navale, ...	bridge, poker, scrabble, ...

- Historiquement en IA, on s'intéresse surtout aux jeux déterministes, totalement observable, à somme nulle où deux agents jouent à tour de rôle.

Les types de jeux

information	déterministe	hasard
totale	échecs, dames, go, othello, ...	backgammon, monopoly, ...
partielle	bataille navale, ...	bridge, poker, scrabble, ...

- ▷ Historiquement en IA, on s'intéresse surtout aux jeux déterministes, totalement observable, à somme nulle où deux agents jouent à tour de rôle.
- ▷ C'est moins vrai depuis la victoire d'AlphaGo

- s_0 : l'état initial

- s_0 : l'état initial
- `Player(s)` : définit quel joueur doit jouer dans l'état s

Formalisation

- s_0 : l'état initial
- `Player(s)` : définit quel joueur doit jouer dans l'état s
- `Action(s)` : retourne l'ensemble d'actions possibles dans l'état s

- s_0 : l'état initial
- `Player(s)` : définit quel joueur doit jouer dans l'état s
- `Action(s)` : retourne l'ensemble d'actions possibles dans l'état s
- `Result(s; a)` : fonction de transition, qui définit quel est le résultat de l'action a dans un état s

- s_0 : l'état initial
- `Player(s)` : définit quel joueur doit jouer dans l'état s
- `Action(s)` : retourne l'ensemble d'actions possibles dans l'état s
- `Result(s; a)` : fonction de transition, qui définit quel est le résultat de l'action a dans un état s
- `Terminal-Test(s)` : test de terminaison. Vrai si le jeu est fini dans l'état s, faux sinon. Les états dans lesquels le jeu est terminé sont appelés états terminaux.

Formalisation

- s_0 : l'état initial
- `Player(s)` : définit quel joueur doit jouer dans l'état s
- `Action(s)` : retourne l'ensemble d'actions possibles dans l'état s
- `Result(s; a)` : fonction de transition, qui définit quel est le résultat de l'action a dans un état s
- `Terminal-Test(s)` : test de terminaison. Vrai si le jeu est fini dans l'état s, faux sinon. Les états dans lesquels le jeu est terminé sont appelés états terminaux.
- `Utility(s; p)` : une fonction d'utilité associe une valeur numérique à chaque état terminal s pour un joueur p

Jeu à somme nulle

- Un jeu à somme nulle (*zero-sum game*) est un jeu pour lequel la somme des utilités de tous les joueurs est la même pour toutes les issues possible du jeu

Jeu à somme nulle

- Un jeu à somme nulle (*zero-sum game*) est un jeu pour lequel la somme des utilités de tous les joueurs est la même pour toutes les issues possible du jeu
- Par exemple, le jeu d'échecs est un jeu à somme nulle :

Jeu à somme nulle

- Un jeu à somme nulle (*zero-sum game*) est un jeu pour lequel la somme des utilités de tous les joueurs est la même pour toutes les issues possible du jeu
- Par exemple, le jeu d'échecs est un jeu à somme nulle :
  - Victoire `p2` : `Utility(s; p1)`=0 ; `Utility(s; p2)`=1 ⇒ somme = 1

Jeu à somme nulle

- Un jeu à somme nulle (*zero-sum game*) est un jeu pour lequel la somme des utilités de tous les joueurs est la même pour toutes les issues possible du jeu
- Par exemple, le jeu d'échecs est un jeu à somme nulle :
  - Victoire `p2` : `Utility(s; p1)`=0 ; `Utility(s; p2)`=1 ⇒ somme = 1
  - Victoire `p1` : `Utility(s; p1)`=1 ; `Utility(s; p2)`=0 ⇒ somme = 1

Jeu à somme nulle

- Un jeu à somme nulle (*zero-sum game*) est un jeu pour lequel la somme des utilités de tous les joueurs est la même pour toutes les issues possible du jeu
- Par exemple, le jeu d'échecs est un jeu à somme nulle :
  - Victoire `p2` : `Utility(s; p1)`=0 ; `Utility(s; p2)`=1 ⇒ somme = 1
  - Victoire `p1` : `Utility(s; p1)`=1 ; `Utility(s; p2)`=0 ⇒ somme = 1
  - Nul : `Utility(s; p1)`=$\frac{1}{2}$; `Utility(s; p2)`=$\frac{1}{2}$ ⇒ somme = 1

Jeu à somme nulle

- Un jeu à somme nulle (*zero-sum game*) est un jeu pour lequel la somme des utilités de tous les joueurs est la même pour toutes les issues possible du jeu
- Par exemple, le jeu d'échecs est un jeu à somme nulle :
  - Victoire `p2` : `Utility(s; p1)`=0 ; `Utility(s; p2)`=1 $\Rightarrow$ somme = 1
  - Victoire `p1` : `Utility(s; p1)`=1 ; `Utility(s; p2)`=0 $\Rightarrow$ somme = 1
  - Nul : `Utility(s; p1)`=$\frac{1}{2}$; `Utility(s; p2)`=$\frac{1}{2}$ $\Rightarrow$ somme = 1
- Le nom n'est pas le plus adapté, jeu à somme constante serait plus approprié

Jeu à somme nulle

- Un jeu à somme nulle (*zero-sum game*) est un jeu pour lequel la somme des utilités de tous les joueurs est la même pour toutes les issues possible du jeu
- Par exemple, le jeu d'échecs est un jeu à somme nulle :
  - Victoire `p2` : `Utility(s; p1)`=0 ; `Utility(s; p2)`=1 $\Rightarrow$ somme = 1
  - Victoire `p1` : `Utility(s; p1)`=1 ; `Utility(s; p2)`=0 $\Rightarrow$ somme = 1
  - Nul : `Utility(s; p1)`=$\frac{1}{2}$; `Utility(s; p2)`=$\frac{1}{2}$ $\Rightarrow$ somme = 1
- Le nom n'est pas le plus adapté, jeu à somme constante serait plus approprié
  - Historiquement, on considère un droit d'entrée pour chaque joueur de la moitié de cette "somme"

Jeu à somme nulle

- Un jeu à somme nulle (*zero-sum game*) est un jeu pour lequel la somme des utilités de tous les joueurs est la même pour toutes les issues possible du jeu
- Par exemple, le jeu d'échecs est un jeu à somme nulle :
  - Victoire `p2` : `Utility(s; p1)`=0 ; `Utility(s; p2)`=1 $\Rightarrow$ somme = 1
  - Victoire `p1` :`Utility(s; p1)`=1 ; `Utility(s; p2)`=0 $\Rightarrow$ somme = 1
  - Nul : `Utility(s; p1)`=$\frac{1}{2}$; `Utility(s; p2)`=$\frac{1}{2}$ $\Rightarrow$ somme = 1
- Le nom n'est pas le plus adapté, jeu à somme constante serait plus approprié
  - Historiquement, on considère un droit d'entrée pour chaque joueur de la moitié de cette "somme"
- Dans un jeu à somme nulle, on peut formaliser en disant que :

Jeu à somme nulle

- Un jeu à somme nulle (*zero-sum game*) est un jeu pour lequel la somme des utilités de tous les joueurs est la même pour toutes les issues possible du jeu
- Par exemple, le jeu d'échecs est un jeu à somme nulle :
  - Victoire `p2` : `Utility(s; p1)`=0 ; `Utility(s; p2)`=1 $\Rightarrow$ somme = 1
  - Victoire `p1` : `Utility(s; p1)`=1 ; `Utility(s; p2)`=0 $\Rightarrow$ somme = 1
  - Nul : `Utility(s; p1)`=$\frac{1}{2}$; `Utility(s; p2)`=$\frac{1}{2}$ $\Rightarrow$ somme = 1
- Le nom n'est pas le plus adapté, jeu à somme constante serait plus approprié
  - Historiquement, on considère un droit d'entrée pour chaque joueur de la moitié de cette "somme"
- Dans un jeu à somme nulle, on peut formaliser en disant que :
  - Un joueur cherche à maximiser son utilité

Jeu à somme nulle

- Un jeu à somme nulle (*zero-sum game*) est un jeu pour lequel la somme des utilités de tous les joueurs est la même pour toutes les issues possible du jeu
- Par exemple, le jeu d'échecs est un jeu à somme nulle :
  - Victoire `p2` : `Utility(s; p1)`=0 ; `Utility(s; p2)`=1 $\Rightarrow$ somme = 1
  - Victoire `p1` : `Utility(s; p1)`=1 ; `Utility(s; p2)`=0 $\Rightarrow$ somme = 1
  - Nul : `Utility(s; p1)`=$\frac{1}{2}$; `Utility(s; p2)`=$\frac{1}{2}$ $\Rightarrow$ somme = 1
- Le nom n'est pas le plus adapté, jeu à somme constante serait plus approprié
  - Historiquement, on considère un droit d'entrée pour chaque joueur de la moitié de cette "somme"
- Dans un jeu à somme nulle, on peut formaliser en disant que :
  - Un joueur cherche à maximiser son utilité
  - L'autre joueur cherche à minimiser l'utilité de son adversaire

Jeu à somme nulle

- Un jeu à somme nulle (*zero-sum game*) est un jeu pour lequel la somme des utilités de tous les joueurs est la même pour toutes les issues possible du jeu
- Par exemple, le jeu d'échecs est un jeu à somme nulle :
  - Victoire `p2` : `Utility(s; p1)`=0 ; `Utility(s; p2)`=1 $\Rightarrow$ somme = 1
  - Victoire `p1` : `Utility(s; p1)`=1 ; `Utility(s; p2)`=0 $\Rightarrow$ somme = 1
  - Nul : `Utility(s; p1)`=$\frac{1}{2}$; `Utility(s; p2)`=$\frac{1}{2}$ $\Rightarrow$ somme = 1
- Le nom n'est pas le plus adapté, jeu à somme constante serait plus approprié
  - Historiquement, on considère un droit d'entrée pour chaque joueur de la moitié de cette "somme"
- Dans un jeu à somme nulle, on peut formaliser en disant que :
  - Un joueur cherche à maximiser son utilité
  - L'autre joueur cherche à minimiser l'utilité de son adversaire
    - $\Rightarrow$ Ce qui revient à maximiser la sienne !

Un seul joueur

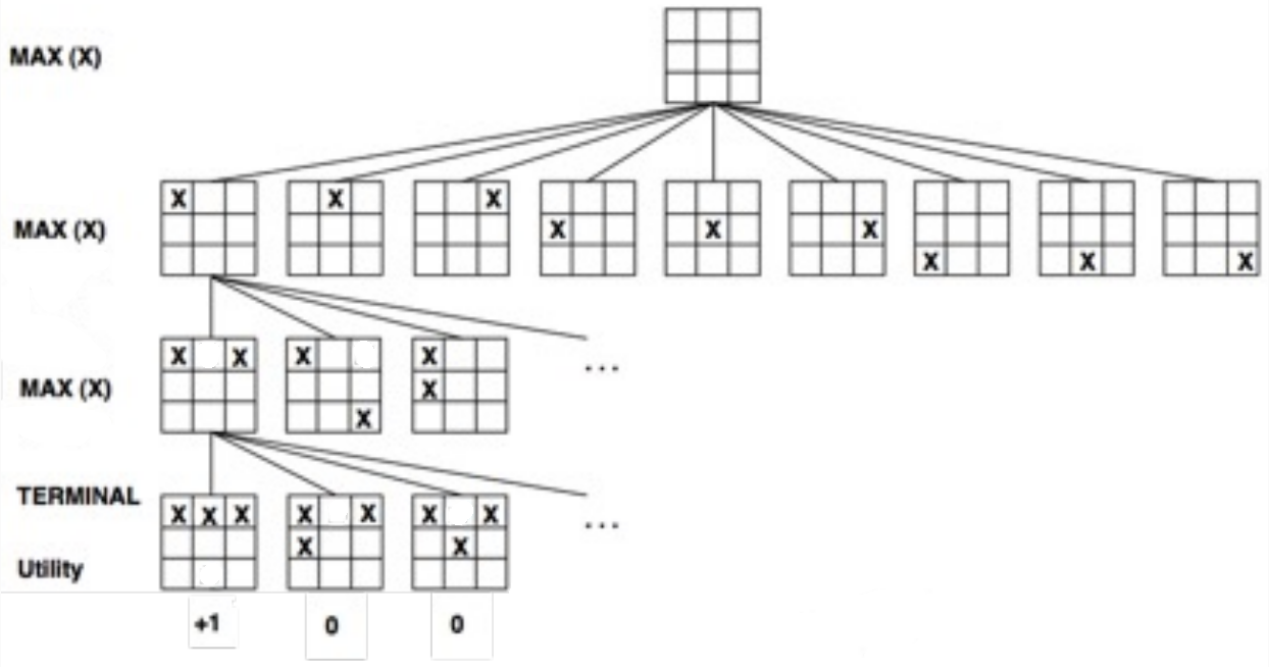
- Imaginons une partie de Morpion à un seul joueur

Un seul joueur

- Imaginons une partie de Morpion à un seul joueur
- Appelons-le Max et faisons le jouer 3 coups

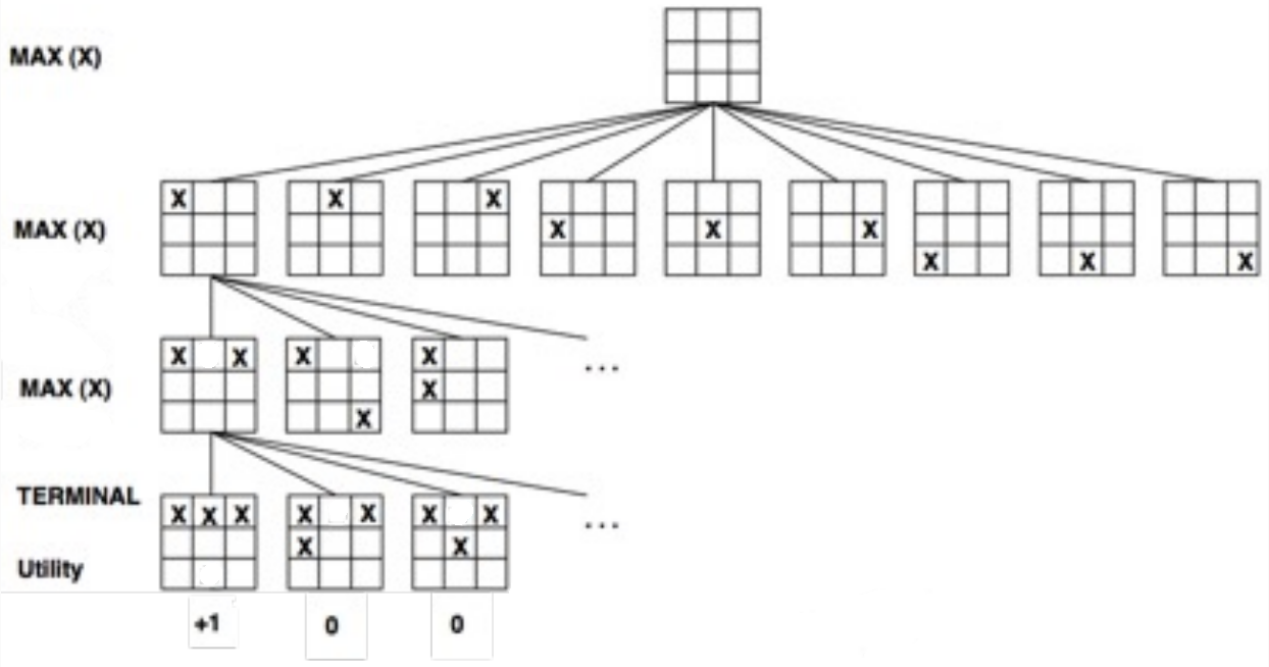
Un seul joueur

- Imaginons une partie de Morpion à un seul joueur
- Appelons-le Max et faisons le jouer 3 coups



Un seul joueur

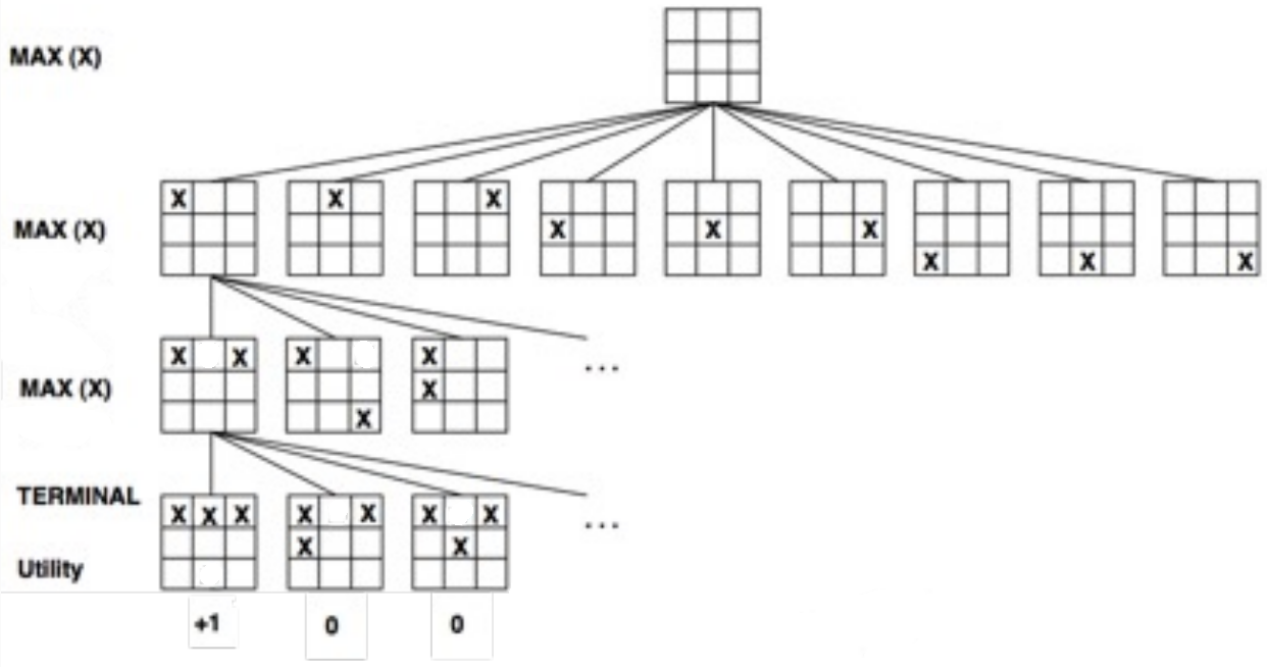
- Imaginons une partie de Morpion à un seul joueur
- Appelons-le Max et faisons le jouer 3 coups



- À un joueur, rien ne peut empêcher Max de gagner (à part lui-même) en choisissant le chemin menant vers une utilité finale de 1

Un seul joueur

- Imaginons une partie de Morpion à un seul joueur
- Appelons-le Max et faisons le jouer 3 coups



- À un joueur, rien ne peut empêcher Max de gagner (à part lui-même) en choisissant le chemin menant vers une utilité finale de 1
- Mais un autre joueur fera tout pour minimiser l'utilité des chemins de Max, nous l'appellerons astucieusement Min

Jeux

Algorithme Min-Max

Principes de Minimax

- Jeu à somme nulle

Principes de Minimax

- Jeu à somme nulle
- Deux joueurs : Min et Max

Principes de Minimax

- Jeu à somme nulle
- Deux joueurs : Min et Max
- Joueurs jouent alternativement

Principes de Minimax

- Jeu à somme nulle
- Deux joueurs : Min et Max
- Joueurs jouent alternativement
- Max maximise son utilité

Principes de Minimax

- Jeu à somme nulle
- Deux joueurs : Min et Max
- Joueurs jouent alternativement
- Max maximise son utilité
- Min minimise l'utilité de Max

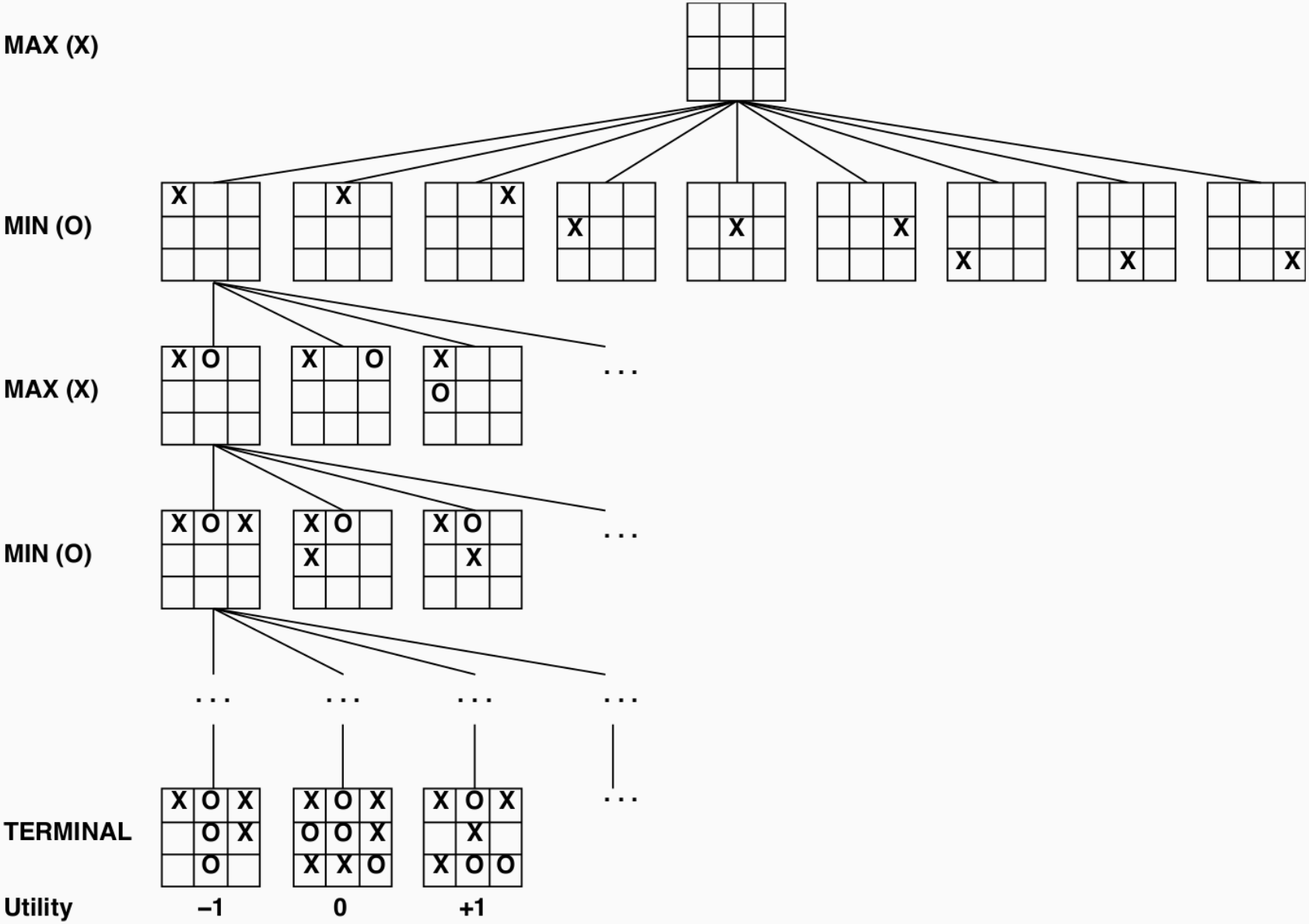
Principes de Minimax

- Jeu à somme nulle
- Deux joueurs : Min et Max
- Joueurs jouent alternativement
- Max maximise son utilité
- Min minimise l'utilité de Max
- Calcule pour chaque nœud minimax la plus haute utilité atteignable contre un adversaire optimal

Principes de Minimax

- Jeu à somme nulle
- Deux joueurs : Min et Max
- Joueurs jouent alternativement
- Max maximise son utilité
- Min minimise l'utilité de Max
- Calcule pour chaque nœud minimax la plus haute utilité atteignable contre un adversaire optimal
- Valeur minimax = meilleur score atteignable contre le meilleur jeu de l'adversaire

Minimax - morpion



Minimax

- Donne le coup parfait pour un jeu déterministe à information parfaite

Minimax

- Donne le coup parfait pour un jeu déterministe à information parfaite
- **Idée** : choisir le coup qui mène vers l'état qui a la meilleure valeur minimax = meilleure valeur possible contre le meilleur jeu de l'adversaire

Minimax

- Donne le coup parfait pour un jeu déterministe à information parfaite
- **Idée** : choisir le coup qui mène vers l'état qui a la meilleure valeur minimax = meilleure valeur possible contre le meilleur jeu de l'adversaire
- Trouver la meilleure stratégie pour Max :

Minimax

- Donne le coup parfait pour un jeu déterministe à information parfaite
- **Idée** : choisir le coup qui mène vers l'état qui a la meilleure valeur minimax = meilleure valeur possible contre le meilleur jeu de l'adversaire
- Trouver la meilleure stratégie pour Max :
  - Parcours en profondeur de l'arbre de jeu

Minimax

- Donne le coup parfait pour un jeu déterministe à information parfaite
- **Idée** : choisir le coup qui mène vers l'état qui a la meilleure valeur minimax = meilleure valeur possible contre le meilleur jeu de l'adversaire
- Trouver la meilleure stratégie pour Max :
  - Parcours en profondeur de l'arbre de jeu
  - Un nœud peut apparaitre à n'importe quelle profondeur

Minimax

- Donne le coup parfait pour un jeu déterministe à information parfaite
- **Idée** : choisir le coup qui mène vers l'état qui a la meilleure valeur minimax = meilleure valeur possible contre le meilleur jeu de l'adversaire
- Trouver la meilleure stratégie pour Max :
  - Parcours en profondeur de l'arbre de jeu
  - Un nœud peut apparaitre à n'importe quelle profondeur
  - Calculer l'utilité d'un état part du principe que les deux joueurs jouent de façon optimale depuis cet état jusqu'à la fin du jeu

Minimax

- Donne le coup parfait pour un jeu déterministe à information parfaite
- **Idée** : choisir le coup qui mène vers l'état qui a la meilleure valeur minimax = meilleure valeur possible contre le meilleur jeu de l'adversaire
- Trouver la meilleure stratégie pour Max :
  - Parcours en profondeur de l'arbre de jeu
  - Un nœud peut apparaitre à n'importe quelle profondeur
  - Calculer l'utilité d'un état part du principe que les deux joueurs jouent de façon optimale depuis cet état jusqu'à la fin du jeu
  - Propagation de la valeur minimax vers le haut de l'arbre dès que les nœuds terminaux sont découverts

Minimax

- Donne le coup parfait pour un jeu déterministe à information parfaite
- **Idée** : choisir le coup qui mène vers l'état qui a la meilleure valeur minimax = meilleure valeur possible contre le meilleur jeu de l'adversaire
- Trouver la meilleure stratégie pour Max :
  - Parcours en profondeur de l'arbre de jeu
  - Un nœud peut apparaitre à n'importe quelle profondeur
  - Calculer l'utilité d'un état part du principe que les deux joueurs jouent de façon optimale depuis cet état jusqu'à la fin du jeu
  - Propagation de la valeur minimax vers le haut de l'arbre dès que les nœuds terminaux sont découverts
- Valeur des nœuds :

Minimax

- Donne le coup parfait pour un jeu déterministe à information parfaite
- **Idée** : choisir le coup qui mène vers l'état qui a la meilleure valeur minimax = meilleure valeur possible contre le meilleur jeu de l'adversaire
- Trouver la meilleure stratégie pour Max :
  - Parcours en profondeur de l'arbre de jeu
  - Un nœud peut apparaitre à n'importe quelle profondeur
  - Calculer l'utilité d'un état part du principe que les deux joueurs jouent de façon optimale depuis cet état jusqu'à la fin du jeu
  - Propagation de la valeur minimax vers le haut de l'arbre dès que les nœuds terminaux sont découverts
- Valeur des nœuds :
  - Nœud terminal : utilité de l'état terminal

Minimax

- Donne le coup parfait pour un jeu déterministe à information parfaite
- **Idée** : choisir le coup qui mène vers l'état qui a la meilleure valeur minimax = meilleure valeur possible contre le meilleur jeu de l'adversaire
- Trouver la meilleure stratégie pour Max :
  - Parcours en profondeur de l'arbre de jeu
  - Un nœud peut apparaitre à n'importe quelle profondeur
  - Calculer l'utilité d'un état part du principe que les deux joueurs jouent de façon optimale depuis cet état jusqu'à la fin du jeu
  - Propagation de la valeur minimax vers le haut de l'arbre dès que les nœuds terminaux sont découverts
- Valeur des nœuds :
  - Nœud terminal : utilité de l'état terminal
  - Nœud Max : valeur maximum des nœuds précédents

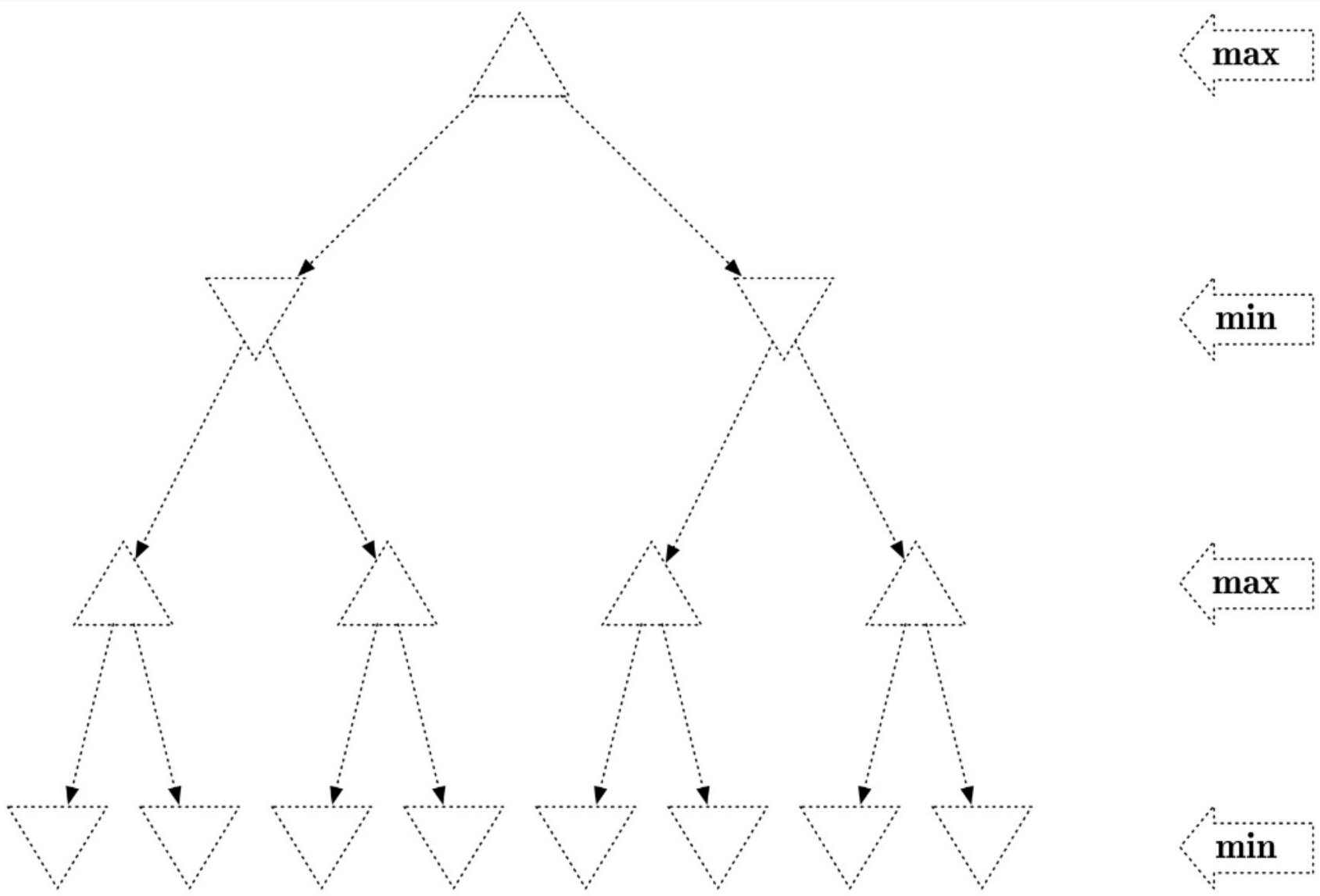
Minimax

- Donne le coup parfait pour un jeu déterministe à information parfaite
- **Idée** : choisir le coup qui mène vers l'état qui a la meilleure valeur minimax = meilleure valeur possible contre le meilleur jeu de l'adversaire
- Trouver la meilleure stratégie pour Max :
  - Parcours en profondeur de l'arbre de jeu
  - Un nœud peut apparaitre à n'importe quelle profondeur
  - Calculer l'utilité d'un état part du principe que les deux joueurs jouent de façon optimale depuis cet état jusqu'à la fin du jeu
  - Propagation de la valeur minimax vers le haut de l'arbre dès que les nœuds terminaux sont découverts
- Valeur des nœuds :
  - Nœud terminal : utilité de l'état terminal
  - Nœud Max : valeur maximum des nœuds précédents
  - Nœud Min : valeur minimum des nœuds précédents

Minimax - algorithme

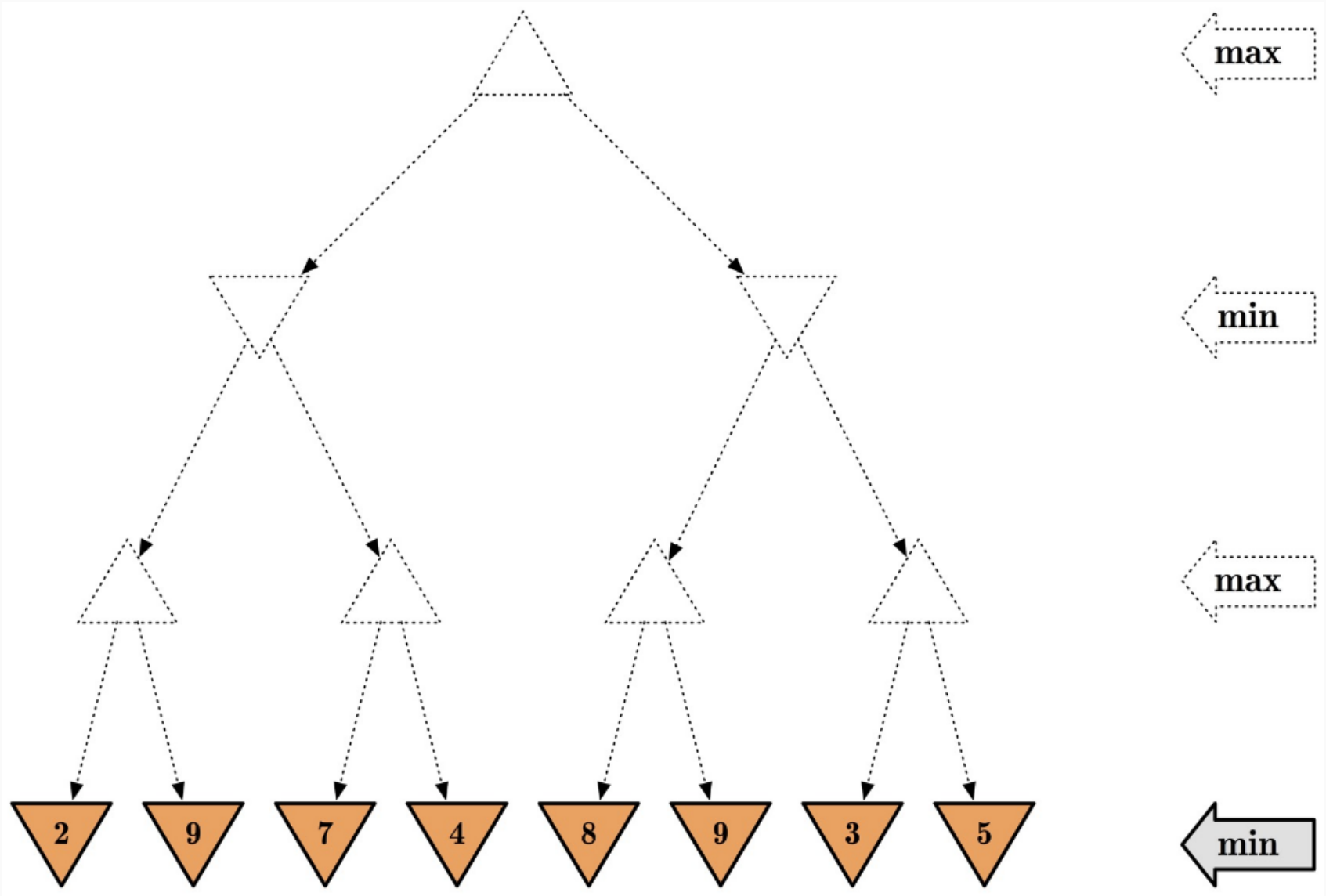
```
function MINIMAX-DECISION(state) returns an action
    inputs: state, current state in game

    return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))

function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do v ← MAX(v, MIN-VALUE(s))
    return v

function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for a, s in SUCCESSORS(state) do v ← MIN(v, MAX-VALUE(s))
    return v
```

Minimax - exemple



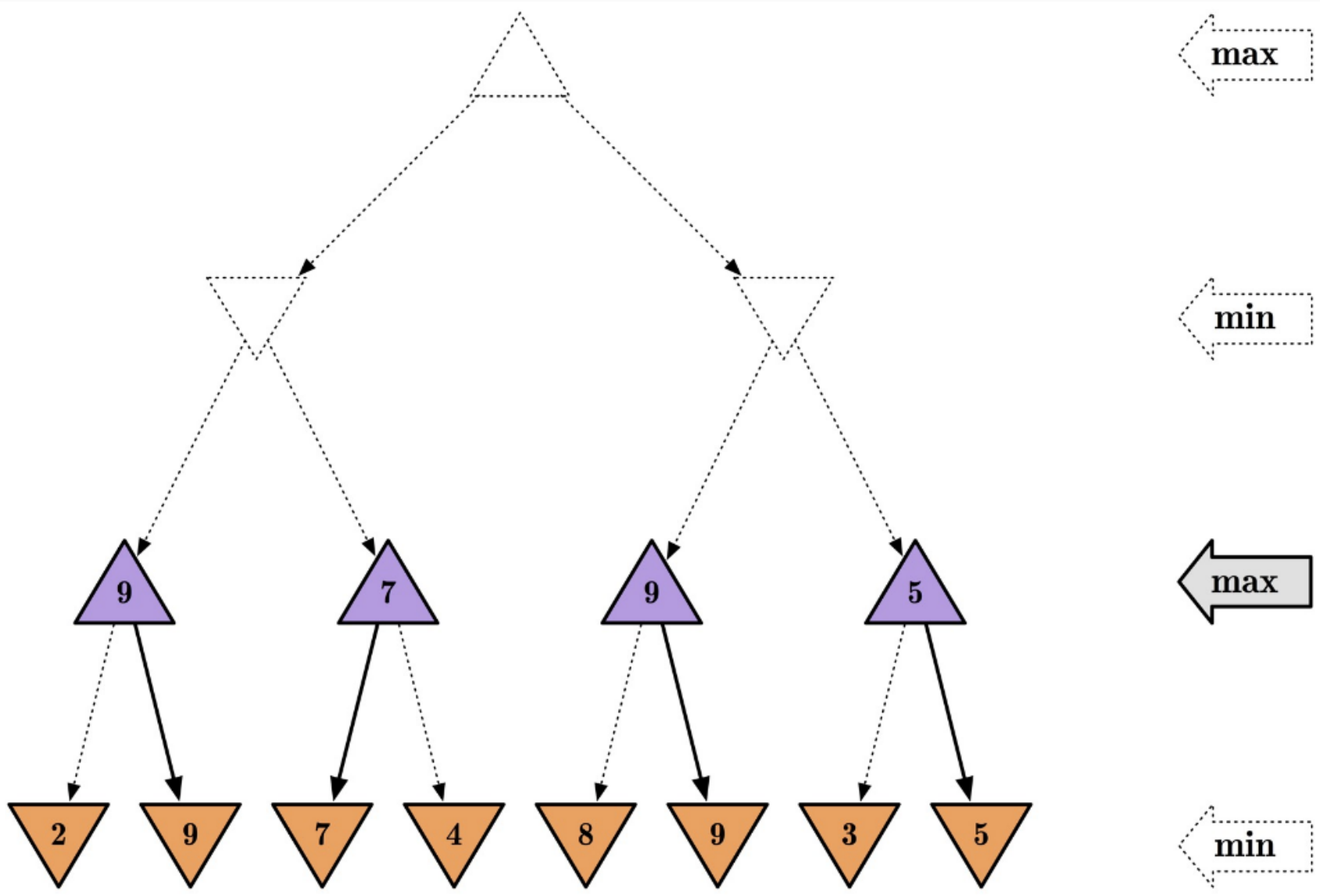
©Ansaf Salleb-Aouissi, Columbia

Minimax - exemple



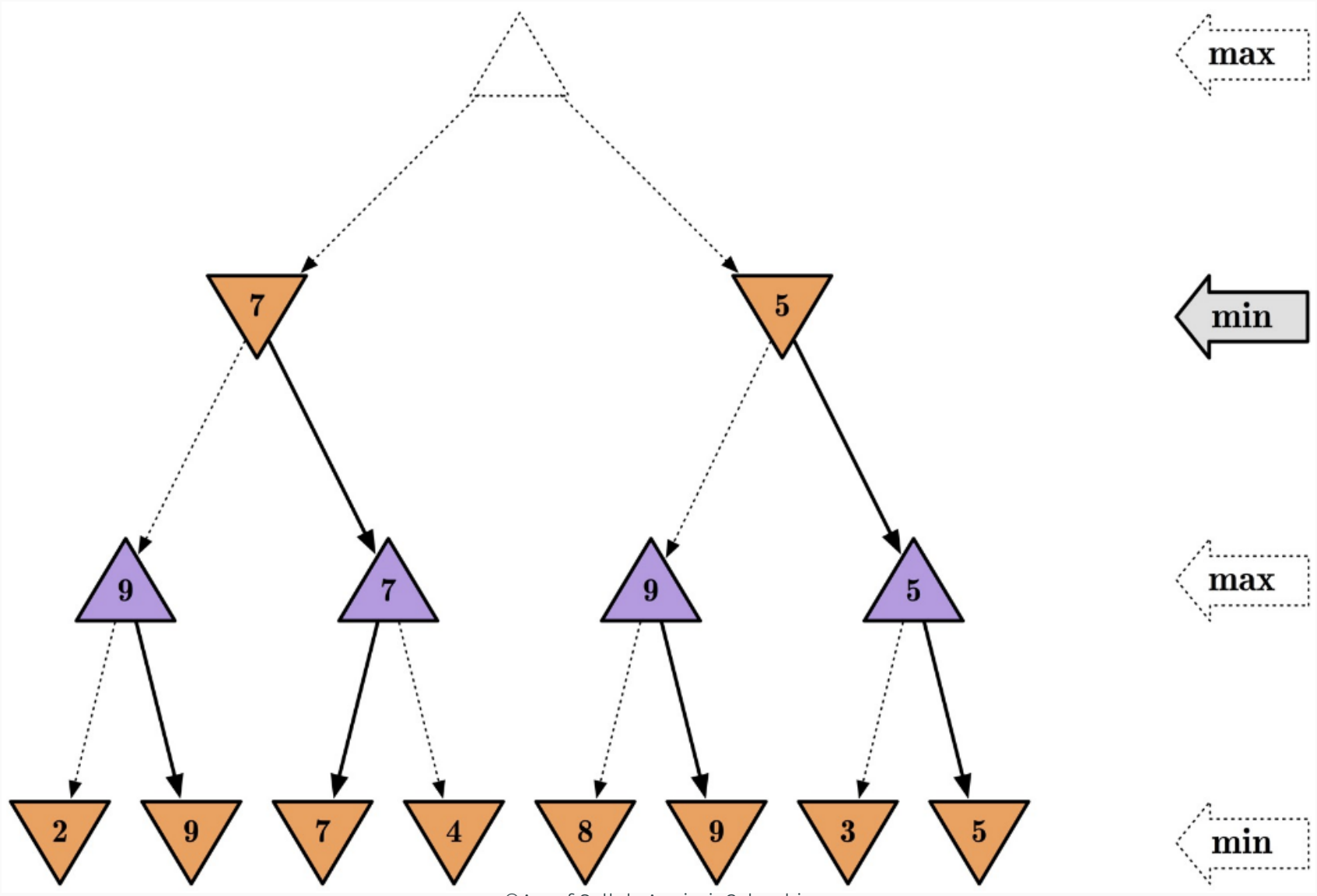
©Ansaf Salleb-Aouissi, Columbia

Minimax - exemple



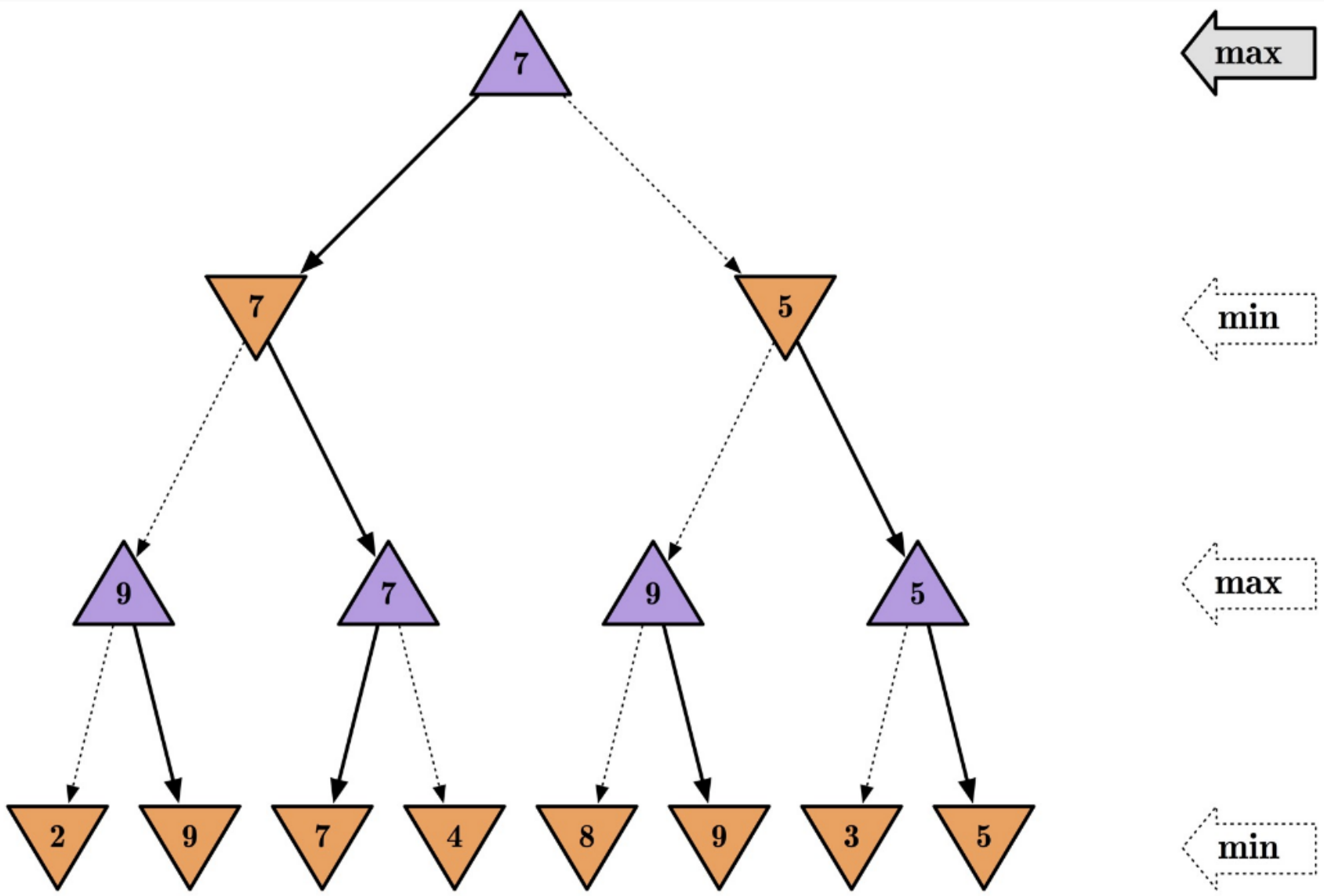
©Ansaf Salleb-Aouissi, Columbia

Minimax - exemple



©Ansaf Salleb-Aouissi, Columbia

Minimax - exemple



©Ansaf Salleb-Aouissi, Columbia

- complétude : Oui (sauf si l'arbre de jeu est infini)

- complétude : Oui (sauf si l'arbre de jeu est infini)
- complexité en temps : $O(b^m)$

- complétude : Oui (sauf si l'arbre de jeu est infini)
- complexité en temps : $O(b^m)$
- complexité en mémoire : $O(bm)$

Minimax

- complétude : Oui (sauf si l'arbre de jeu est infini)
- complexité en temps : $O(b^m)$
- complexité en mémoire : $O(bm)$
- optimalité : Oui (si l'adversaire joue de façon optimale)

- complétude : Oui (sauf si l'arbre de jeu est infini)
- complexité en temps : $O(b^m)$
- complexité en mémoire : $O(bm)$
- optimalité : Oui (si l'adversaire joue de façon optimale)
- Morpion :

Minimax

- complétude : Oui (sauf si l'arbre de jeu est infini)
- complexité en temps : $O(b^m)$
- complexité en mémoire : $O(bm)$
- optimalité : Oui (si l'adversaire joue de façon optimale)
- Morpion :
  - $b \simeq 5$ en moyenne

- complétude : Oui (sauf si l'arbre de jeu est infini)
- complexité en temps : $O(b^m)$
- complexité en mémoire : $O(bm)$
- optimalité : Oui (si l'adversaire joue de façon optimale)
- Morpion :
  - $b \simeq 5$ en moyenne
  - $d = 9$

- complétude : Oui (sauf si l'arbre de jeu est infini)
- complexité en temps : $O(b^m)$
- complexité en mémoire : $O(bm)$
- optimalité : Oui (si l'adversaire joue de façon optimale)
- Morpion :
  - $b \simeq 5$ en moyenne
  - $d = 9$
  - $5^9 = 1953125$

- complétude : Oui (sauf si l'arbre de jeu est infini)
- complexité en temps : $O(b^m)$
- complexité en mémoire : $O(bm)$
- optimalité : Oui (si l'adversaire joue de façon optimale)
- Morpion :
  - $b \simeq 5$ en moyenne
  - $d = 9$
  - $5^9 = 1953125$
- Échecs :

Minimax

- complétude : Oui (sauf si l'arbre de jeu est infini)
- complexité en temps : $O(b^m)$
- complexité en mémoire : $O(bm)$
- optimalité : Oui (si l'adversaire joue de façon optimale)
- Morpion :
  - $b \simeq 5$ en moyenne
  - $d = 9$
  - $5^9 = 1953125$
- Échecs :
  - $b \simeq 35$ en moyenne

Minimax

- complétude : Oui (sauf si l'arbre de jeu est infini)
- complexité en temps : $O(b^m)$
- complexité en mémoire : $O(bm)$
- optimalité : Oui (si l'adversaire joue de façon optimale)
- Morpion :
  - $b \simeq 5$ en moyenne
  - $d = 9$
  - $5^9 = 1953125$
- Échecs :
  - $b \simeq 35$ en moyenne
  - $d \simeq 100$ en moyenne

Minimax

- complétude : Oui (sauf si l'arbre de jeu est infini)
- complexité en temps : $O(b^m)$
- complexité en mémoire : $O(bm)$
- optimalité : Oui (si l'adversaire joue de façon optimale)
- Morpion :
  - $b \simeq 5$ en moyenne
  - $d = 9$
  - $5^9 = 1953125$
- Échecs :
  - $b \simeq 35$ en moyenne
  - $d \simeq 100$ en moyenne
  - $35^{100} = 10^{154}$

Minimax

- complétude : Oui (sauf si l'arbre de jeu est infini)
- complexité en temps : $O(b^m)$
- complexité en mémoire : $O(bm)$
- optimalité : Oui (si l'adversaire joue de façon optimale)
- Morpion :
  - $b \simeq 5$ en moyenne
  - $d = 9$
  - $5^9 = 1953125$
- Échecs :
  - $b \simeq 35$ en moyenne
  - $d \simeq 100$ en moyenne
  - $35^{100} = 10^{154}$
  - minimax pas adapté aux échecs ...

Minimax

- complétude : Oui (sauf si l'arbre de jeu est infini)
- complexité en temps : $O(b^m)$
- complexité en mémoire : $O(bm)$
- optimalité : Oui (si l'adversaire joue de façon optimale)
- Morpion :
  - $b \simeq 5$ en moyenne
  - $d = 9$
  - $5^9 = 1953125$
- Échecs :
  - $b \simeq 35$ en moyenne
  - $d \simeq 100$ en moyenne
  - $35^{100} = 10^{154}$
  - minimax pas adapté aux échecs ...
- Go :

Minimax

- complétude : Oui (sauf si l'arbre de jeu est infini)
- complexité en temps : $O(b^m)$
- complexité en mémoire : $O(bm)$
- optimalité : Oui (si l'adversaire joue de façon optimale)
- Morpion :
  - $b \simeq 5$ en moyenne
  - $d = 9$
  - $5^9 = 1953125$
- Échecs :
  - $b \simeq 35$ en moyenne
  - $d \simeq 100$ en moyenne
  - $35^{100} = 10^{154}$
  - minimax pas adapté aux échecs ...
- Go :
  - $b \simeq 300$ en moyenne ....

Minimax

- complétude : Oui (sauf si l'arbre de jeu est infini)
- complexité en temps : $O(b^m)$
- complexité en mémoire : $O(bm)$
- optimalité : Oui (si l'adversaire joue de façon optimale)
- Morpion :
  - $b \simeq 5$ en moyenne
  - $d = 9$
  - $5^9 = 1953125$
- Échecs :
  - $b \simeq 35$ en moyenne
  - $d \simeq 100$ en moyenne
  - $35^{100} = 10^{154}$
  - minimax pas adapté aux échecs ...
- Go :
  - $b \simeq 300$ en moyenne ....

$\Rightarrow$ Est-il nécessaire d'explorer toutes les possibilités ?

Ressources limitées

- Dans les jeux réels, il y a une limite de temps, on ne peut donc pas explorer tout l'arbre de jeu

Ressources limitées

- Dans les jeux réels, il y a une limite de temps, on ne peut donc pas explorer tout l'arbre de jeu
- Pour s'exécuter dans un temps raisonnable, minimax devrait limiter la profondeur de recherche

Ressources limitées

- Dans les jeux réels, il y a une limite de temps, on ne peut donc pas explorer tout l'arbre de jeu
- Pour s'exécuter dans un temps raisonnable, minimax devrait limiter la profondeur de recherche
- Mais chaque "coup" exploré améliore/renseigne la stratégie, les choix.

Ressources limitées

- Dans les jeux réels, il y a une limite de temps, on ne peut donc pas explorer tout l'arbre de jeu
- Pour s'exécuter dans un temps raisonnable, minimax devrait limiter la profondeur de recherche
- Mais chaque "coup" exploré améliore/renseigne la stratégie, les choix.
- Possibilités :

Ressources limitées

- Dans les jeux réels, il y a une limite de temps, on ne peut donc pas explorer tout l'arbre de jeu
- Pour s'exécuter dans un temps raisonnable, minimax devrait limiter la profondeur de recherche
- Mais chaque "coup" exploré améliore/renseigne la stratégie, les choix.
- Possibilités :
  1. Remplacer l'utilité terminal par une fonction d'évaluation des états non-terminaux

Ressources limitées

- Dans les jeux réels, il y a une limite de temps, on ne peut donc pas explorer tout l'arbre de jeu
- Pour s'exécuter dans un temps raisonnable, minimax devrait limiter la profondeur de recherche
- Mais chaque "coup" exploré améliore/renseigne la stratégie, les choix.
- Possibilités :
  1. Remplacer l'utilité terminal par une fonction d'évaluation des états non-terminaux
  2. Utiliser le parcours itératif en profondeur

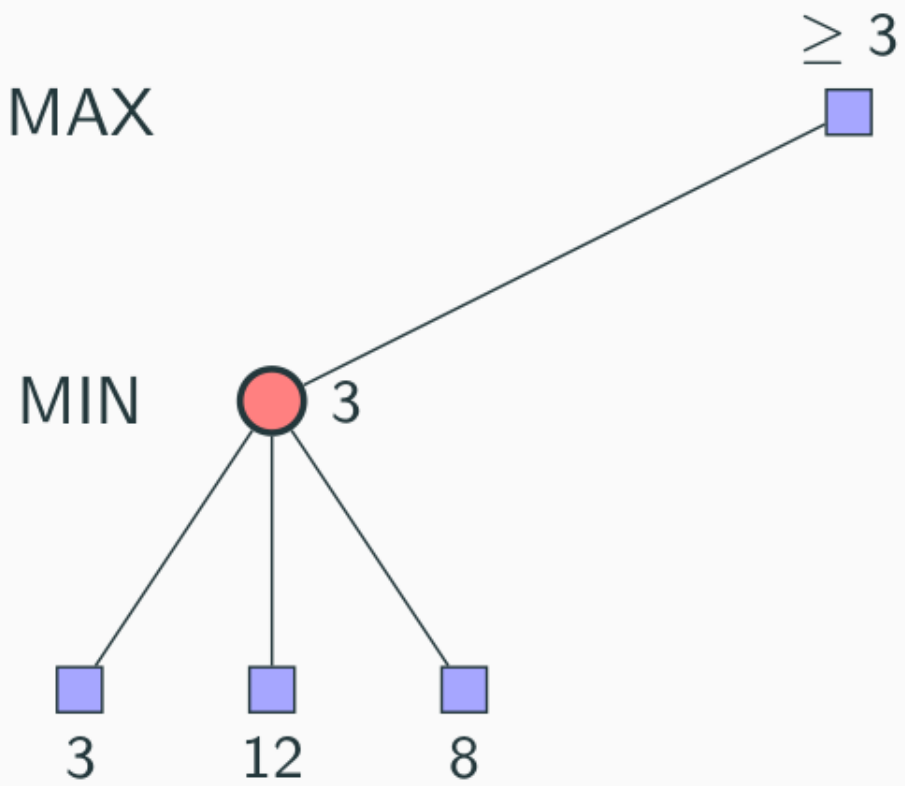
Ressources limitées

- Dans les jeux réels, il y a une limite de temps, on ne peut donc pas explorer tout l'arbre de jeu
- Pour s'exécuter dans un temps raisonnable, minimax devrait limiter la profondeur de recherche
- Mais chaque "coup" exploré améliore/renseigne la stratégie, les choix.
- Possibilités :
  1. Remplacer l'utilité terminal par une fonction d'évaluation des états non-terminaux
  2. Utiliser le parcours itératif en profondeur
  3. Élaguer l'arbre de recherche

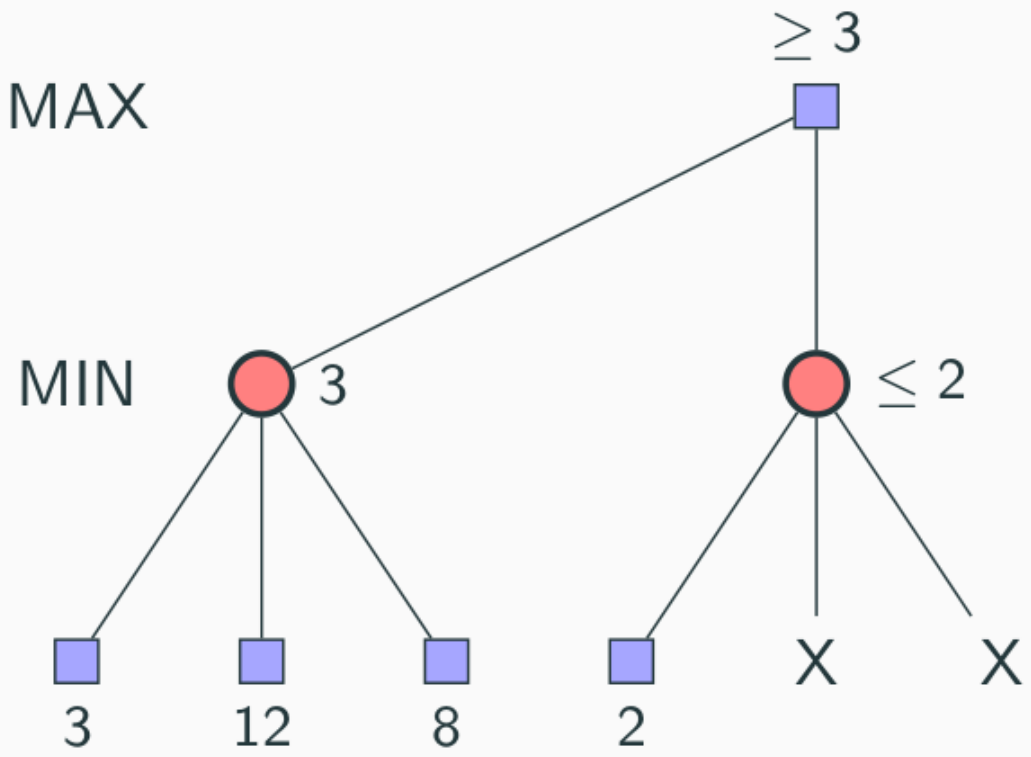
Jeux

Élagage $\alpha - \beta$

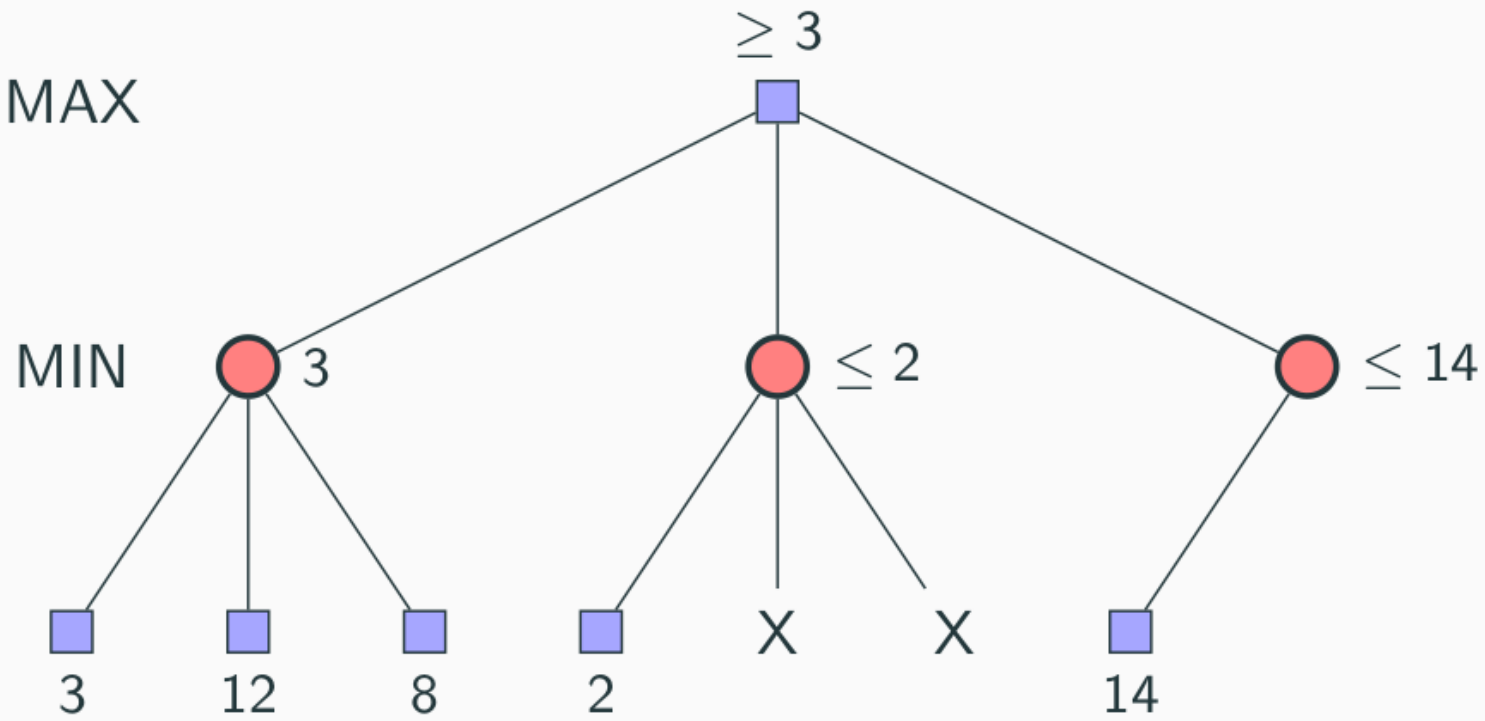
$\alpha - \beta$ PAR L'EXEMPLE



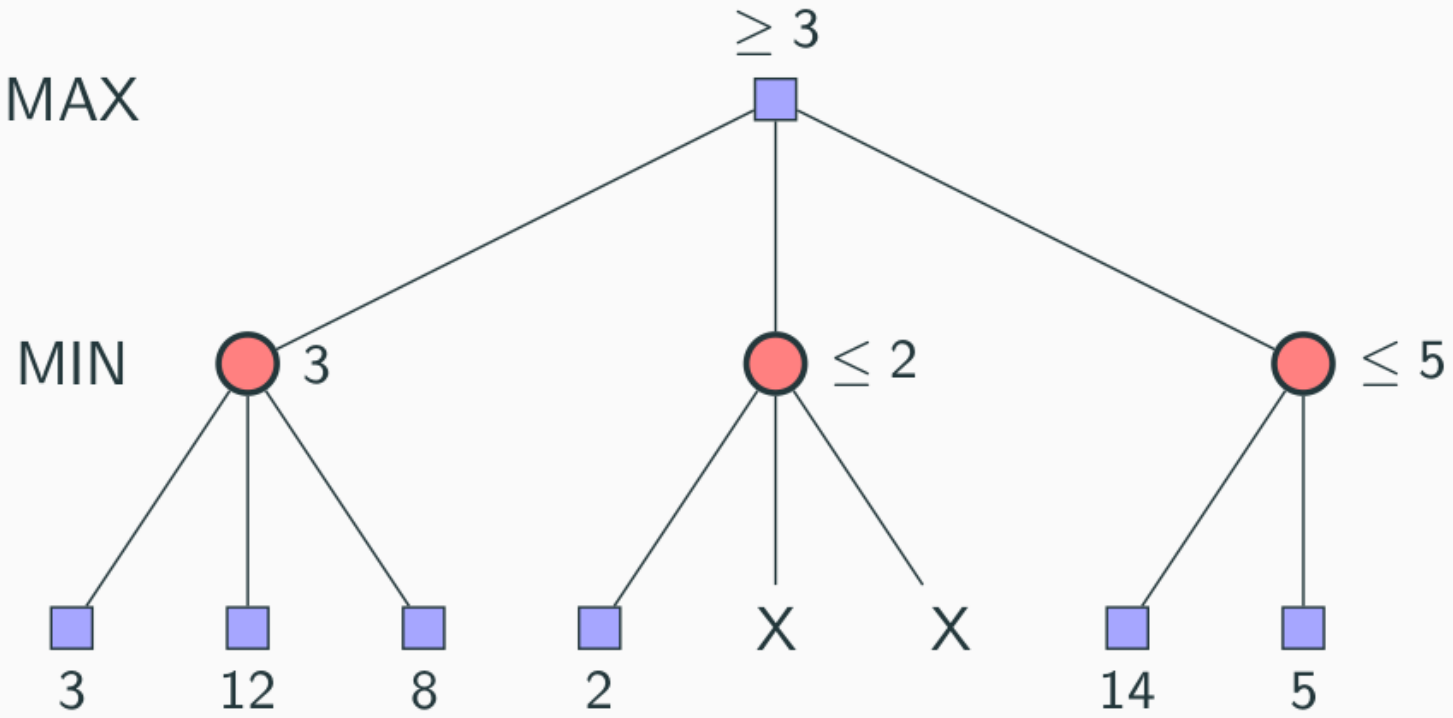
$\alpha - \beta$ PAR L'EXEMPLE



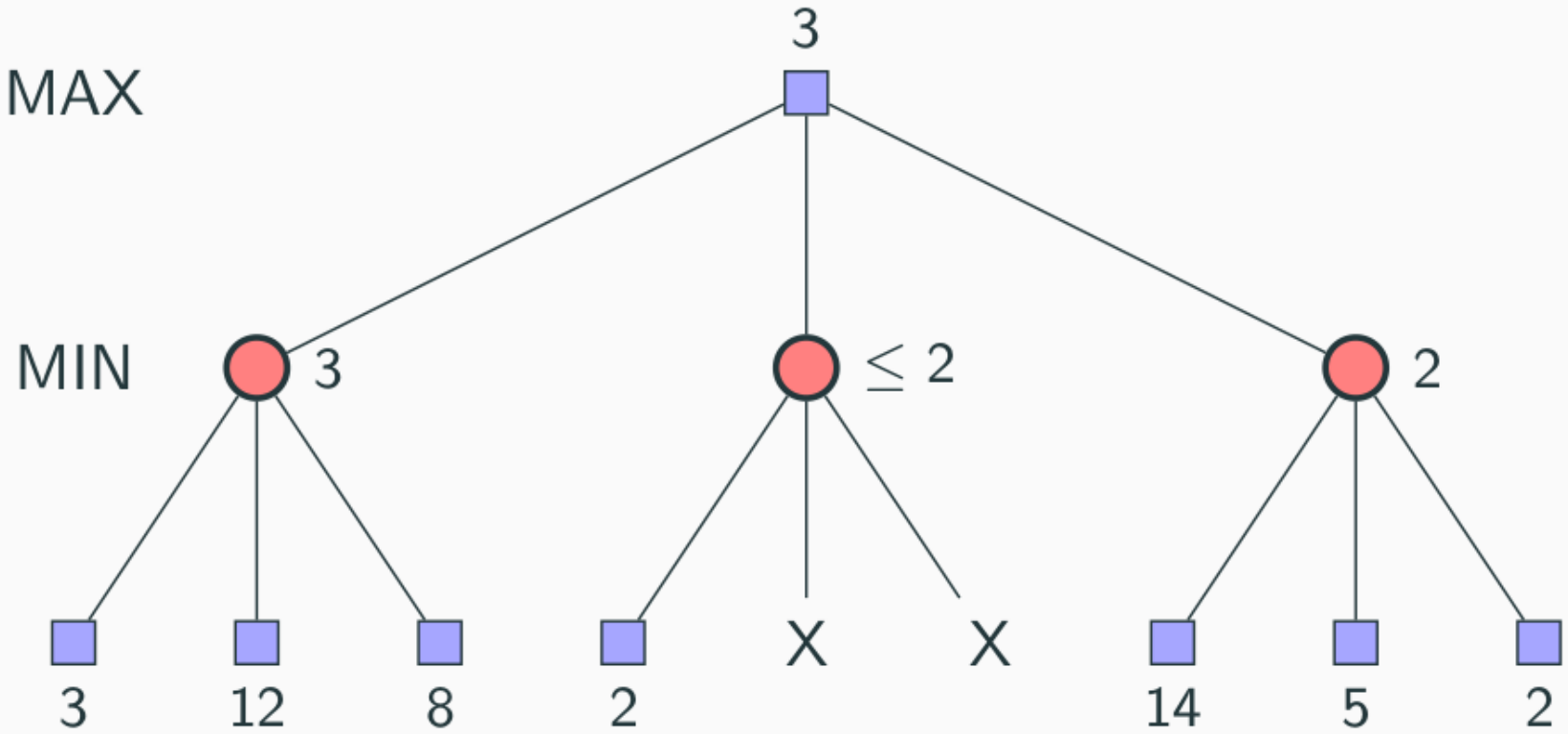
$\alpha - \beta$ PAR L'EXEMPLE



$\alpha - \beta$ PAR L'EXEMPLE



$\alpha - \beta$ PAR L'EXEMPLE



Élagage $\alpha - \beta$

- Comme minimax, parcours en profondeur

Élagage $\alpha - \beta$

- Comme minimax, parcours en profondeur
- Paramètres :

Élagage $\alpha - \beta$

- Comme minimax, parcours en profondeur
- Paramètres :
  - α : plus grande valeur pour MAX parmi les successeurs développés (limite inférieure des valeurs de MAX)

Élagage $\alpha - \beta$

- Comme minimax, parcours en profondeur
- Paramètres :
  - α : plus grande valeur pour MAX parmi les successeurs développés (limite inférieure des valeurs de MAX)
  - β : plus petite valeur pour MIN parmi les successeurs développés (limite supérieure des valeurs de MIN)

Élagage $\alpha - \beta$

- Comme minimax, parcours en profondeur
- Paramètres :
  - α : plus grande valeur pour MAX parmi les successeurs développés (limite inférieure des valeurs de MAX)
  - β : plus petite valeur pour MIN parmi les successeurs développés (limite supérieure des valeurs de MIN)
- Initialisation : $\alpha = -\infty, \beta = +\infty$

Élagage $\alpha - \beta$

- Comme minimax, parcours en profondeur
- Paramètres :
  - α : plus grande valeur pour MAX parmi les successeurs développés (limite inférieure des valeurs de MAX)
  - β : plus petite valeur pour MIN parmi les successeurs développés (limite supérieure des valeurs de MIN)
- Initialisation : $\alpha = -\infty, \beta = +\infty$
- Propagation :

Élagage $\alpha - \beta$

- Comme minimax, parcours en profondeur
- Paramètres :
  - α : plus grande valeur pour MAX parmi les successeurs développés (limite inférieure des valeurs de MAX)
  - β : plus petite valeur pour MIN parmi les successeurs développés (limite supérieure des valeurs de MIN)
- Initialisation : $\alpha = -\infty, \beta = +\infty$
- Propagation :
  - Envoyer α, β vers les feuilles de l'arbre pour l'élagage

Élagage $\alpha - \beta$

- Comme minimax, parcours en profondeur
- Paramètres :
  - α : plus grande valeur pour MAX parmi les successeurs développés (limite inférieure des valeurs de MAX)
  - β : plus petite valeur pour MIN parmi les successeurs développés (limite supérieure des valeurs de MIN)
- Initialisation : $\alpha = -\infty, \beta = +\infty$
- Propagation :
  - Envoyer α, β vers les feuilles de l'arbre pour l'élagage
  - Mise à jour des α, β en remontant les valeurs des nœuds terminaux

Élagage $\alpha - \beta$

- Comme minimax, parcours en profondeur
- Paramètres :
  - α : plus grande valeur pour MAX parmi les successeurs développés (limite inférieure des valeurs de MAX)
  - β : plus petite valeur pour MIN parmi les successeurs développés (limite supérieure des valeurs de MIN)
- Initialisation : $\alpha = -\infty, \beta = +\infty$
- Propagation :
  - Envoyer α, β vers les feuilles de l'arbre pour l'élagage
  - Mise à jour des α, β en remontant les valeurs des nœuds terminaux
  - Mise à jour d'α dans les nœuds MAX et de β dans les nœuds MIN

Élagage $\alpha - \beta$

- Comme minimax, parcours en profondeur
- Paramètres :
  - α : plus grande valeur pour MAX parmi les successeurs développés (limite inférieure des valeurs de MAX)
  - β : plus petite valeur pour MIN parmi les successeurs développés (limite supérieure des valeurs de MIN)
- Initialisation : $\alpha = -\infty, \beta = +\infty$
- Propagation :
  - Envoyer α, β vers les feuilles de l'arbre pour l'élagage
  - Mise à jour des α, β en remontant les valeurs des nœuds terminaux
  - Mise à jour d'α dans les nœuds MAX et de β dans les nœuds MIN
- Élagage : Élaguer toute branche pour laquelle $\alpha \geq \beta$

Algorithme $\alpha - \beta$

function ALPHA-BETA-DECISION($state$) **returns** an action
  return the a in ACTIONS($state$) maximizing MIN-VALUE(RESULT($a, state$))

---

function MAX-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  inputs: $state$, current state in game
    α, the value of the best alternative for MAX along the path to $state$
    β, the value of the best alternative for MIN along the path to $state$
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow -\infty$
  for a, s in SUCCESSORS($state$) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(s, α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

---

function MIN-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  same as MAX-VALUE but with roles of α, β reversed

Propriétés de $\alpha - \beta$

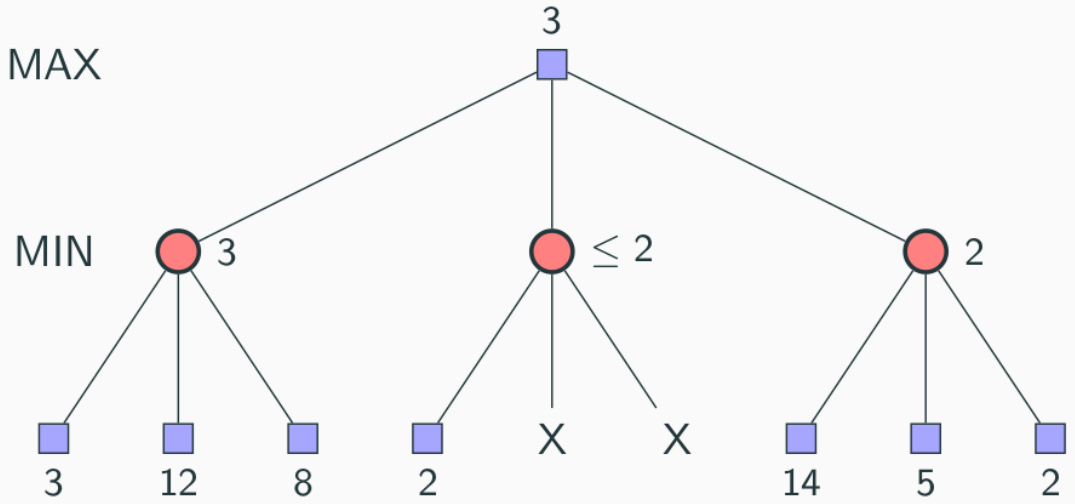
- L'élagage n'affecte pas le résultat final

Propriétés de $\alpha - \beta$

- ▷ L'élagage n'affecte pas le résultat final
- ⚠ Ordre de parcours affecte l'élagage !

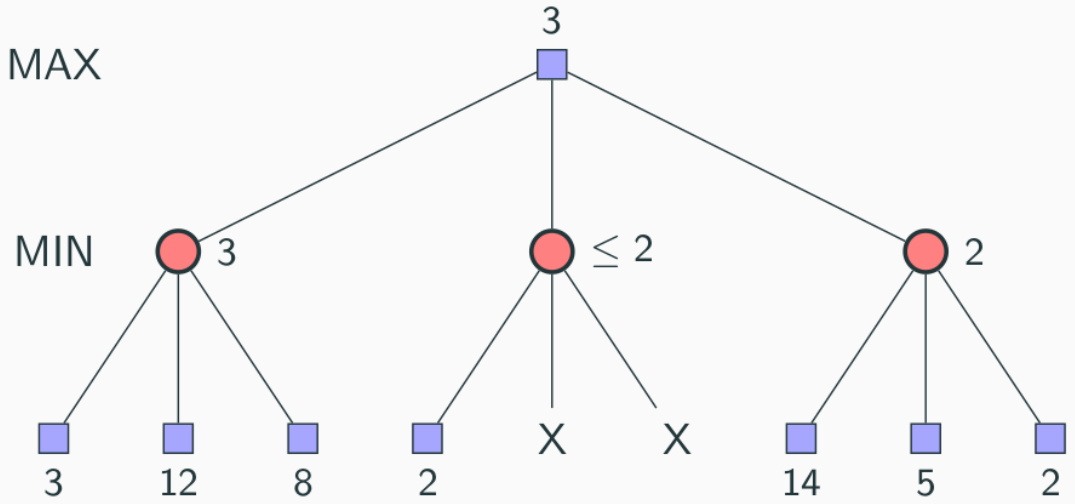
Propriétés de $\alpha - \beta$

- ▷ L'élagage n'affecte pas le résultat final
- ⚠ Ordre de parcours affecte l'élagage !



Propriétés de $\alpha - \beta$

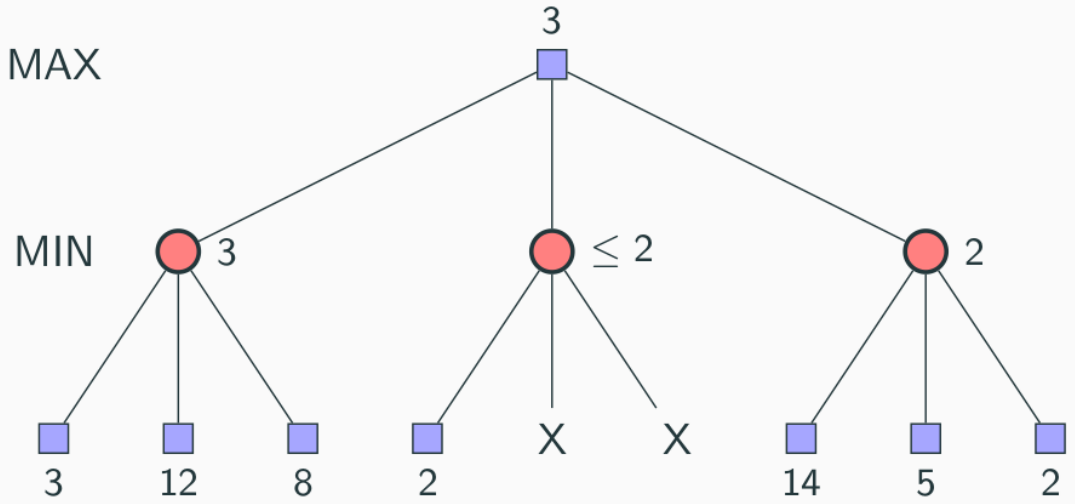
- ▷ L'élagage n'affecte pas le résultat final
- ⚠ Ordre de parcours affecte l'élagage !



- ▷ Pire ordre :

Propriétés de $\alpha - \beta$

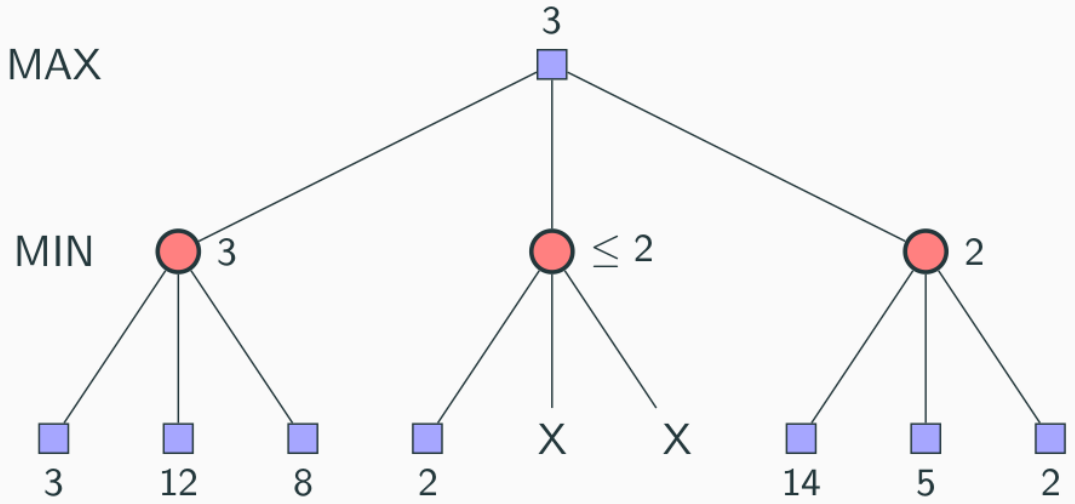
- ▷ L'élagage n'affecte pas le résultat final
- ⚠ Ordre de parcours affecte l'élagage !



- ▷ Pire ordre :
  - ▷ aucun élagage (meilleurs coups toujours à droite dans l'arbre de jeu)

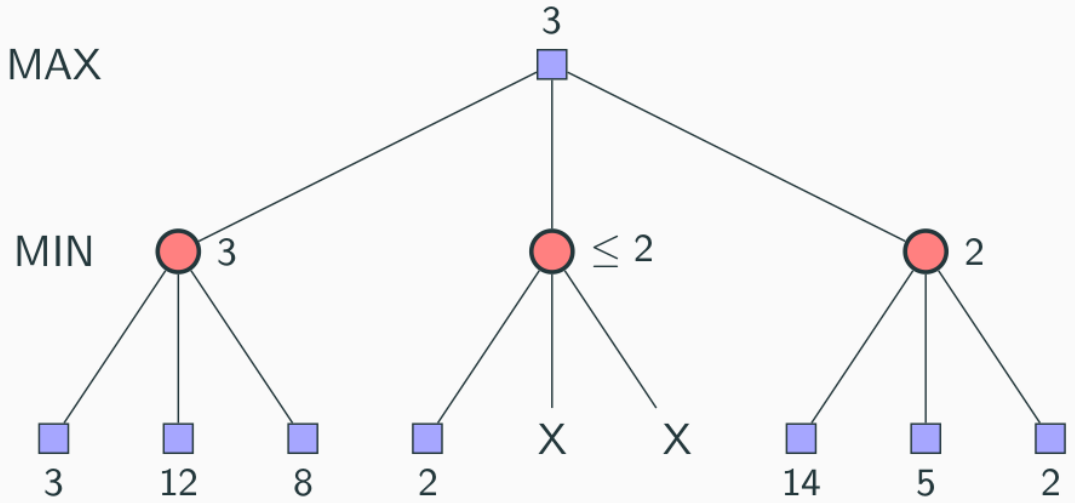
Propriétés de $\alpha - \beta$

- ▷ L'élagage n'affecte pas le résultat final
- ⚠ Ordre de parcours affecte l'élagage !

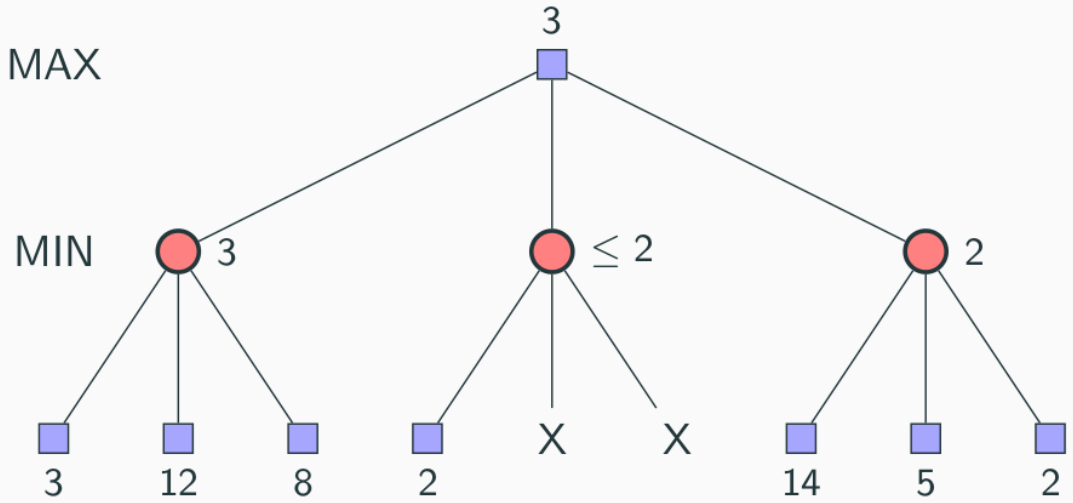


- ▷ Pire ordre :
  - ▷ aucun élagage (meilleurs coups toujours à droite dans l'arbre de jeu)
  - ▷ Complexité toujours en $O(b^m)$

Propriétés de $\alpha - \beta$

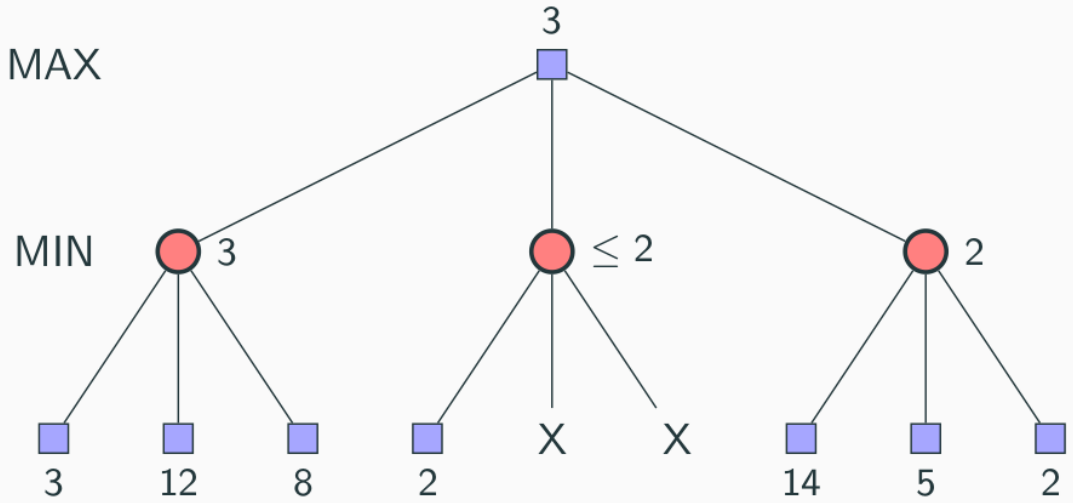


Propriétés de $\alpha - \beta$



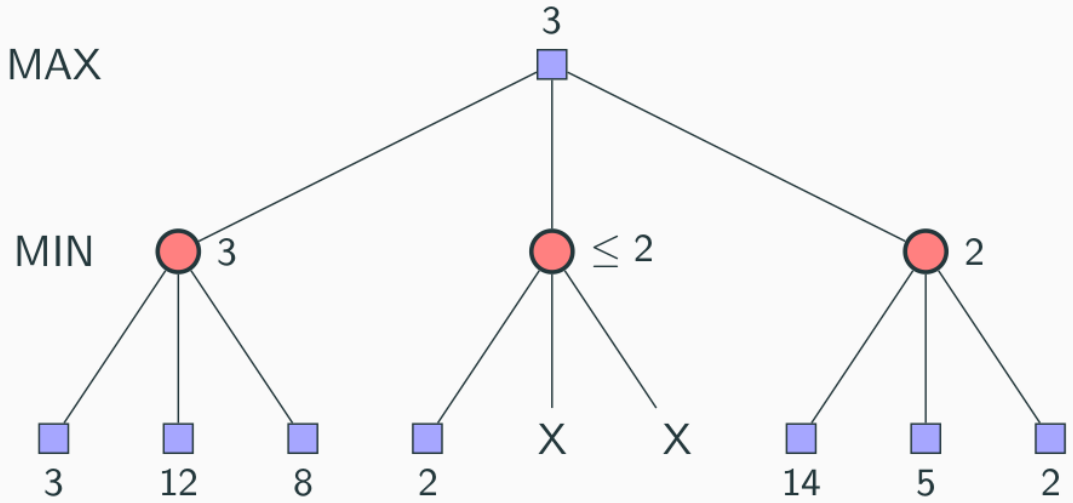
- Ordre parfait :

Propriétés de $\alpha - \beta$



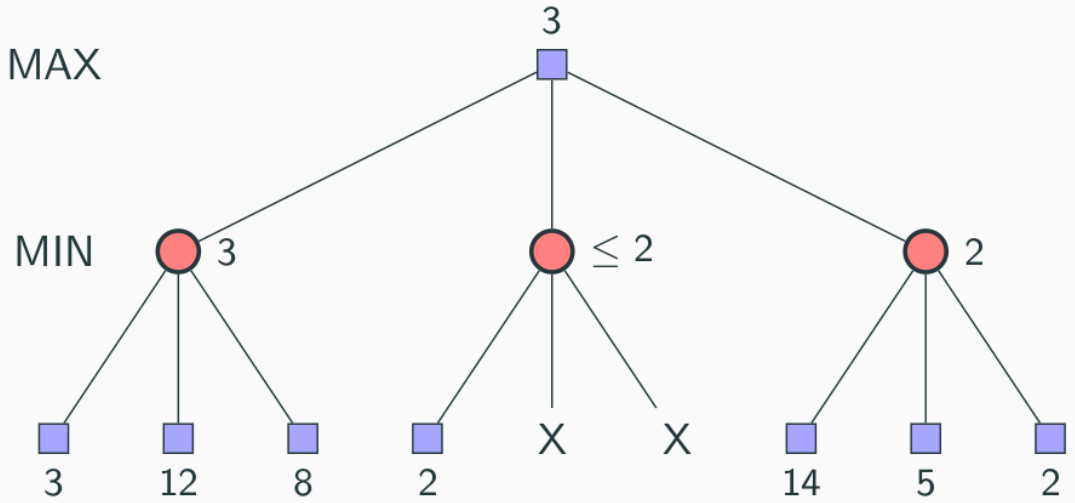
- Ordre parfait :
  - Complexité en $O(b^{\frac{m}{2}})$

Propriétés de $\alpha - \beta$



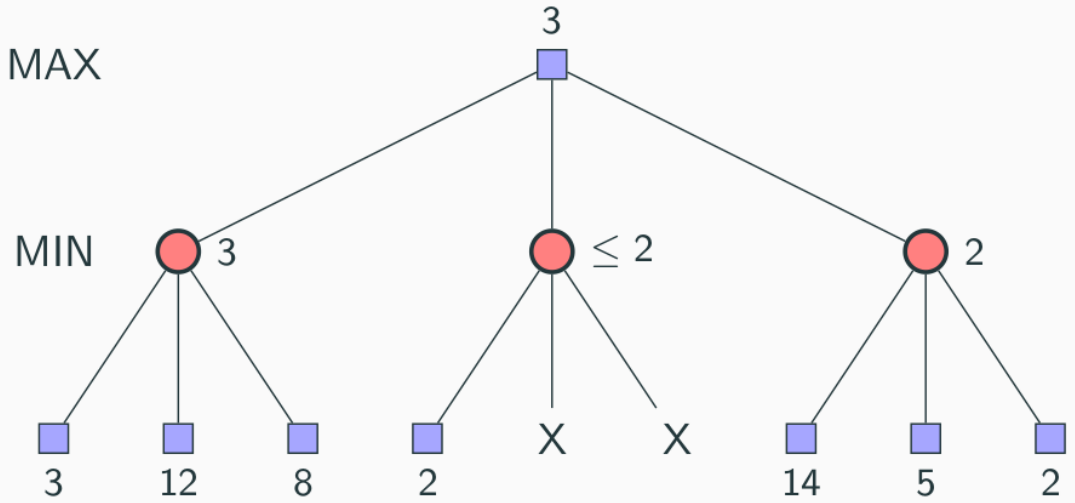
- Ordre parfait :
  - Complexité en $O(b^{\frac{m}{2}})$
  - double la profondeur de recherche par rapport à minimax

Propriétés de $\alpha - \beta$



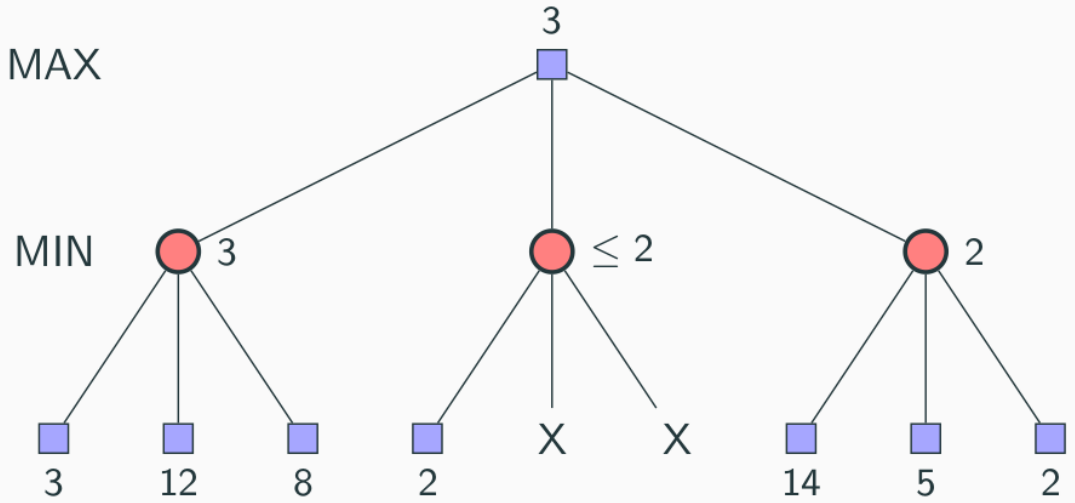
- Ordre parfait :
  - Complexité en $O(b^{\frac{m}{2}})$
  - double la profondeur de recherche par rapport à minimax
  - mais toujours impossible de trouver la solution exacte pour les échecs avec une complexité de 35^{50}

Propriétés de $\alpha - \beta$



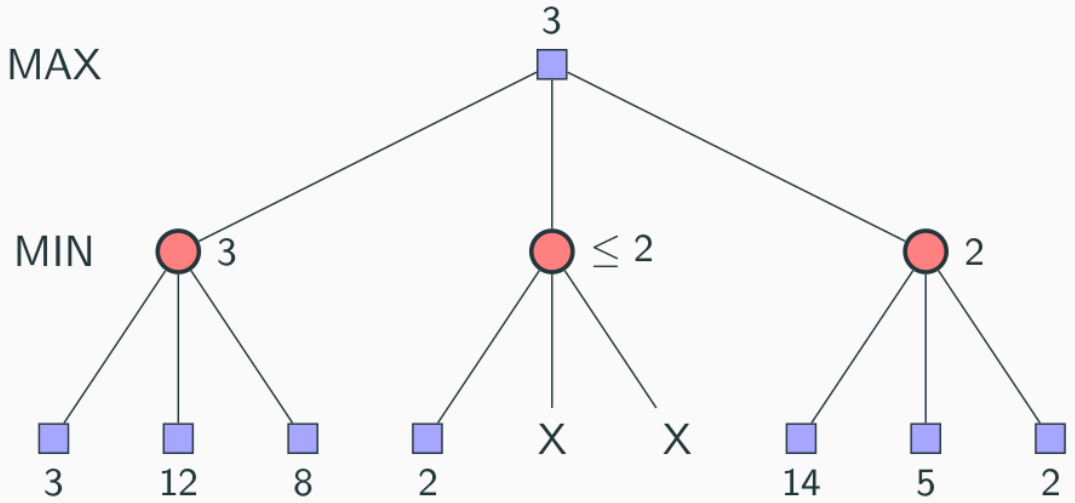
- Ordre parfait :
  - Complexité en $O(b^{\frac{m}{2}})$
  - double la profondeur de recherche par rapport à minimax
  - mais toujours impossible de trouver la solution exacte pour les échecs avec une complexité de 35^{50}
- Cependant :

Propriétés de $\alpha - \beta$



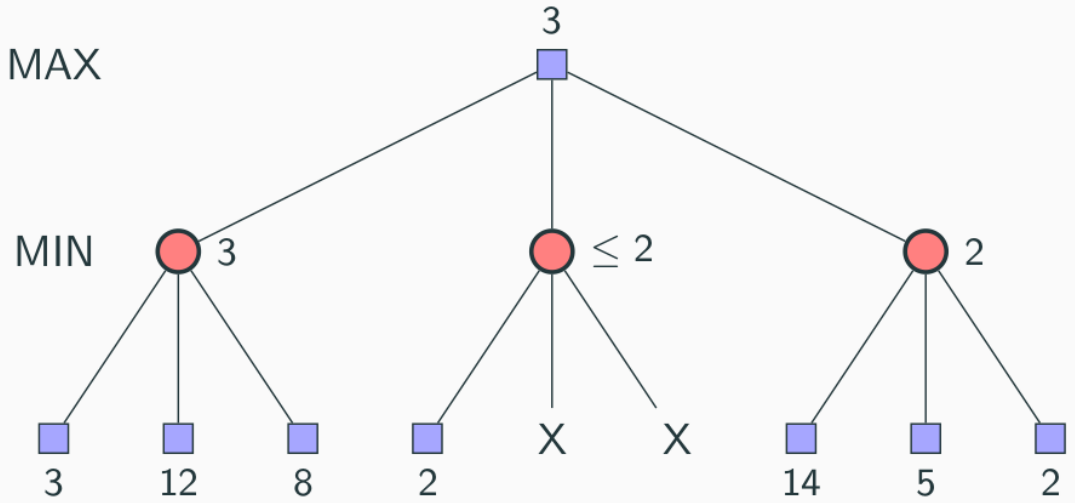
- Ordre parfait :
  - Complexité en $O(b^{\frac{m}{2}})$
  - double la profondeur de recherche par rapport à minimax
  - mais toujours impossible de trouver la solution exacte pour les échecs avec une complexité de 35^{50}
- Cependant :
  - pour un ordinateur capable d'examiner 10^6 coups par seconde et qui disposerait d'une seconde de calcul

Propriétés de $\alpha - \beta$



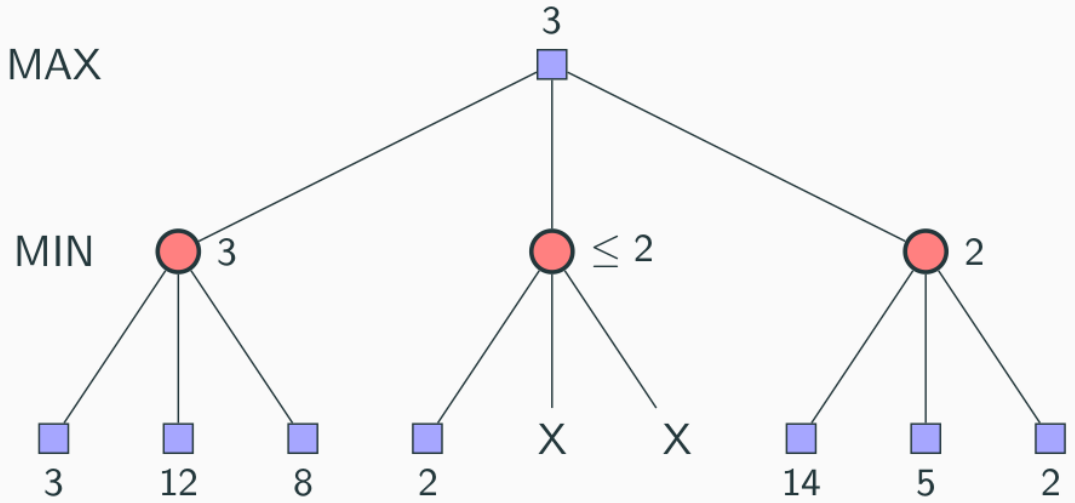
- Ordre parfait :
  - Complexité en $O(b^{\frac{m}{2}})$
  - double la profondeur de recherche par rapport à minimax
  - mais toujours impossible de trouver la solution exacte pour les échecs avec une complexité de 35^{50}
- Cependant :
  - pour un ordinateur capable d'examiner 10^6 coups par seconde et qui disposerait d'une seconde de calcul
  - nous pouvons évaluer 10^6 coups soit environ $35^{\frac{8}{2}}$

Propriétés de $\alpha - \beta$



- Ordre parfait :
  - Complexité en $O(b^{\frac{m}{2}})$
  - double la profondeur de recherche par rapport à minimax
  - mais toujours impossible de trouver la solution exacte pour les échecs avec une complexité de 35^{50}
- Cependant :
  - pour un ordinateur capable d'examiner 10^6 coups par seconde et qui disposerait d'une seconde de calcul
  - nous pouvons évaluer 10^6 coups soit environ $35^{\frac{8}{2}}$
  - $\alpha - \beta$ peut donc "élaborer" des stratégies à 8 coups

Propriétés de $\alpha - \beta$



- Ordre parfait :
  - Complexité en $O(b^{\frac{m}{2}})$
  - double la profondeur de recherche par rapport à minimax
  - mais toujours impossible de trouver la solution exacte pour les échecs avec une complexité de 35^{50}
- Cependant :
  - pour un ordinateur capable d'examiner 10^6 coups par seconde et qui disposerait d'une seconde de calcul
  - nous pouvons évaluer 10^6 coups soit environ $35^{\frac{8}{2}}$
  - $\alpha - \beta$ peut donc "élaborer" des stratégies à 8 coups
  - déjà un bon programme d'échec

Ressources limitées v.2

- Minimax : explore tout l'arbre de jeu

Ressources limitées v.2

- Minimax : explore tout l'arbre de jeu
- $\alpha - \beta$: élague une grande partie de l'arbre de jeu

Ressources limitées v.2

- Minimax : explore tout l'arbre de jeu
- $\alpha - \beta$: élague une grande partie de l'arbre de jeu
- MAIS $\alpha - \beta$ doit toujours aller jusqu'aux nœuds terminaux

Ressources limitées v.2

- Minimax : explore tout l'arbre de jeu
- $\alpha - \beta$: élague une grande partie de l'arbre de jeu
- MAIS $\alpha - \beta$ doit toujours aller jusqu'aux nœuds terminaux
- Impossible en temps réel ou limité

Ressources limitées v.2

- ▷ Minimax : explore tout l'arbre de jeu
- ▷ $\alpha - \beta$: élague une grande partie de l'arbre de jeu
- ▷ MAIS $\alpha - \beta$ doit toujours aller jusqu'aux nœuds terminaux
- ▷ Impossible en temps réel ou limité

⇒ Solution :

Ressources limitées v.2

- ▷ Minimax : explore tout l'arbre de jeu
- ▷ $\alpha - \beta$: élague une grande partie de l'arbre de jeu
- ▷ MAIS $\alpha - \beta$ doit toujours aller jusqu'aux nœuds terminaux
- ▷ Impossible en temps réel ou limité

⇒ Solution :

- ▷ Limiter profondeur de la recherche

Ressources limitées v.2

- ▷ Minimax : explore tout l'arbre de jeu
- ▷ $\alpha - \beta$: élague une grande partie de l'arbre de jeu
- ▷ MAIS $\alpha - \beta$ doit toujours aller jusqu'aux nœuds terminaux
- ▷ Impossible en temps réel ou limité

⇒ Solution :

- ▷ Limiter profondeur de la recherche
- ▷ Remplacer l'utilité par une fonction d'évaluation pour estimer la valeur de l'état courant

Ressources limitées v.2

- `eval(s)` une heuristique pour l'état s :

Ressources limitées v.2

- `eval(s)` une heuristique pour l'état s :
  - Ex. Dames : pièces blanches - pièces noires

Ressources limitées v.2

- `eval(s)` une heuristique pour l'état s :
  - Ex. Dames : pièces blanches - pièces noires
  - Ex. Échecs : valeur des pièces blanches - valeur des pièces noires

Ressources limitées v.2

- `eval(s)` une heuristique pour l'état s :
  - Ex. Dames : pièces blanches - pièces noires
  - Ex. Échecs : valeur des pièces blanches - valeur des pièces noires
  - Transforme les nœuds non-terminaux en feuilles

Ressources limitées v.2

- `eval(s)` une heuristique pour l'état s :
  - Ex. Dames : pièces blanches - pièces noires
  - Ex. Échecs : valeur des pièces blanches - valeur des pièces noires
  - Transforme les nœuds non-terminaux en feuilles
- Une évaluation idéale classerait les états terminaux de la même façon que l'utilité mais doit être rapide

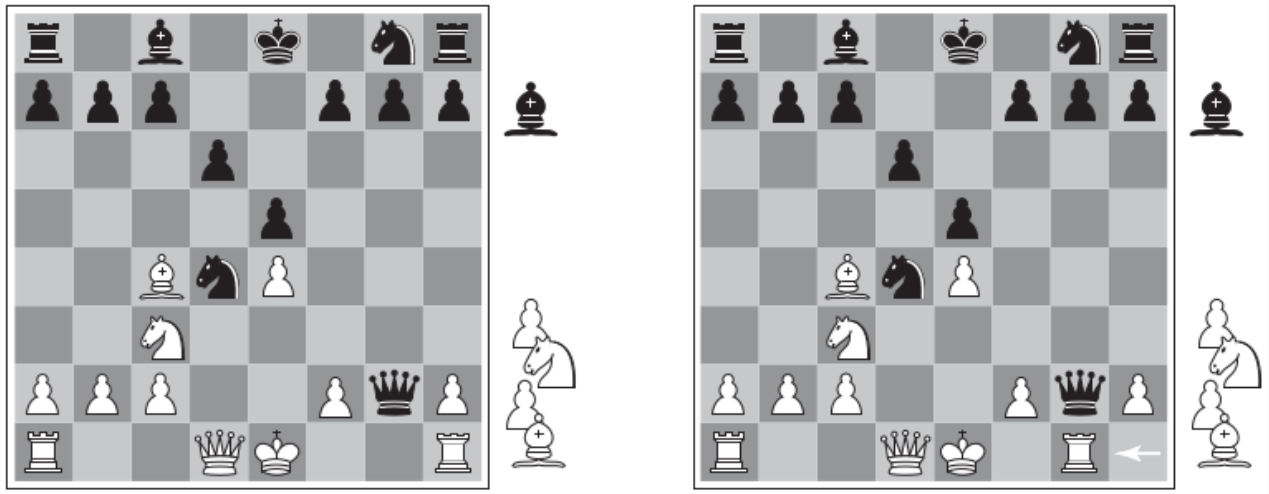
Ressources limitées v.2

- `eval(s)` une heuristique pour l'état s :
  - Ex. Dames : pièces blanches - pièces noires
  - Ex. Échecs : valeur des pièces blanches - valeur des pièces noires
  - Transforme les nœuds non-terminaux en feuilles
- Une évaluation idéale classerait les états terminaux de la même façon que l'utilité mais doit être rapide
- Conception classique : somme linéaire de différentes caractéristiques

Ressources limitées v.2

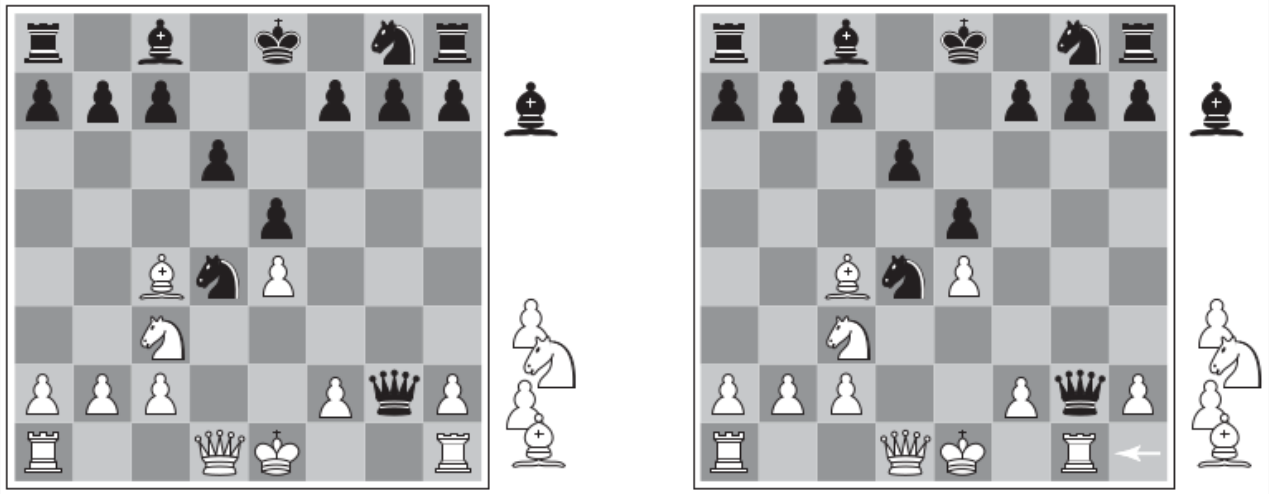
- `eval(s)` une heuristique pour l'état s :
  - Ex. Dames : pièces blanches - pièces noires
  - Ex. Échecs : valeur des pièces blanches - valeur des pièces noires
  - Transforme les nœuds non-terminaux en feuilles
- Une évaluation idéale classerait les états terminaux de la même façon que l'utilité mais doit être rapide
- Conception classique : somme linéaire de différentes caractéristiques
- Apport de la connaissance du domaine pour concevoir les meilleurs caractéristiques

Ressources limitées v.2



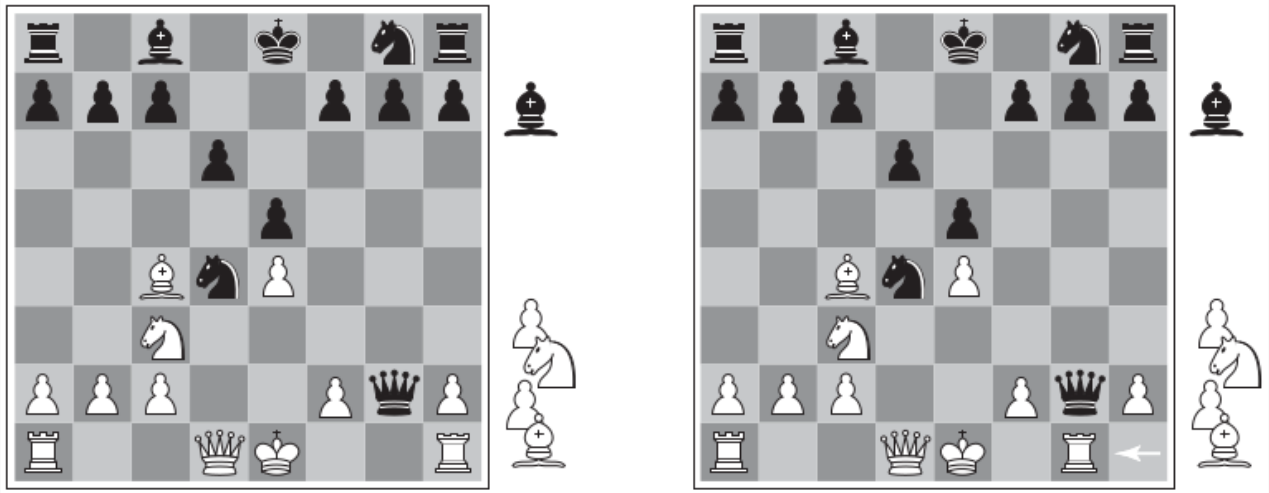
(a) White to move (b) White to move

- Aux échecs, on choisit par exemple une somme linéaire pondérée de caractéristiques



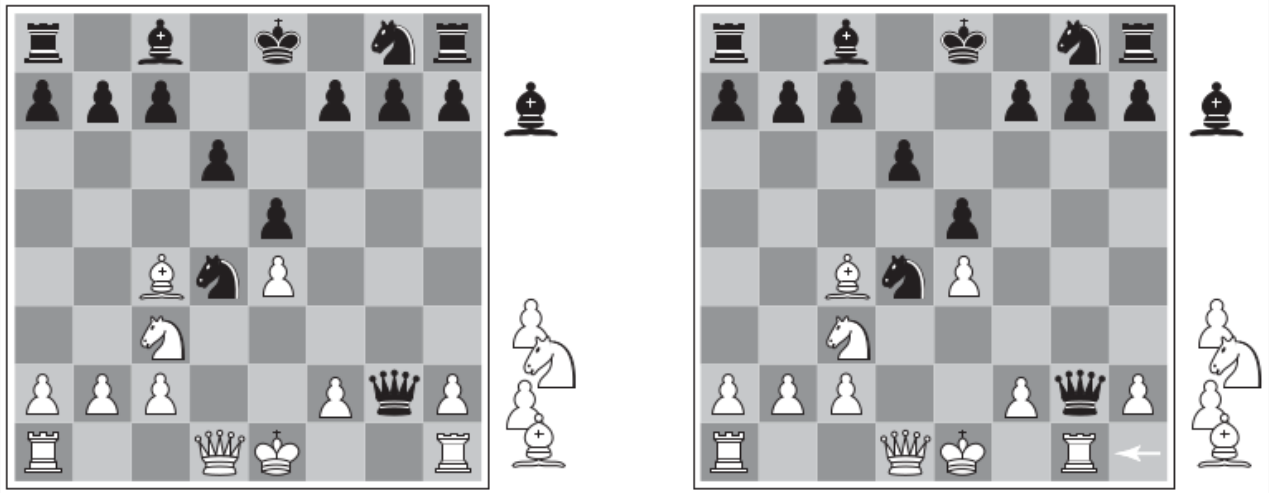
(a) White to move (b) White to move

- Aux échecs, on choisit par exemple une somme linéaire pondérée de caractéristiques
- $eval(s) = w_1 f_1(s) + w_2 f_2(s) + ... + w_n f_n(s) = \sum_{i=1}^{n} w_i f_i(s)$



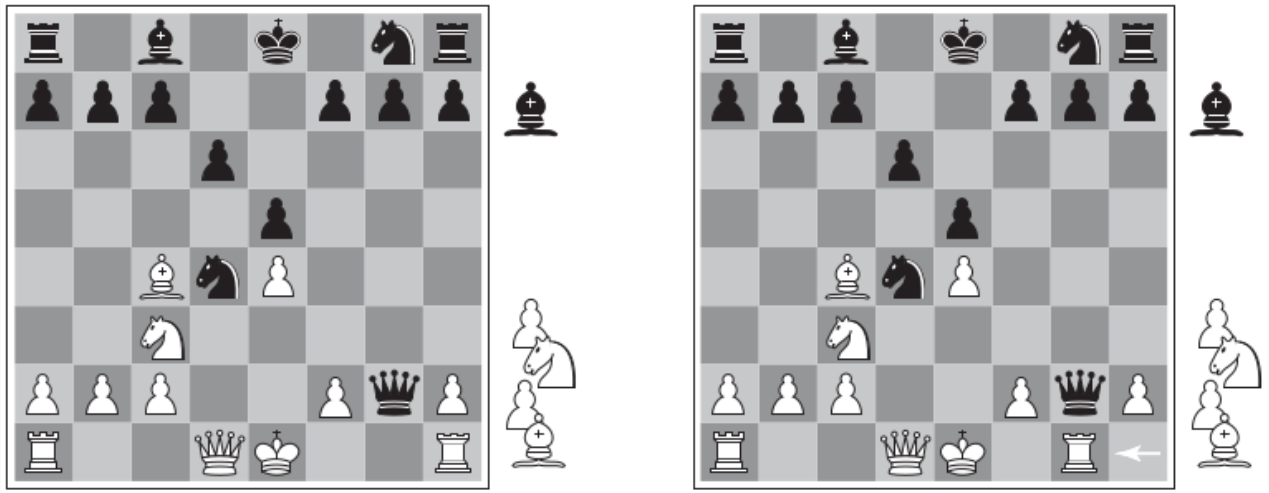
(a) White to move (b) White to move

- Aux échecs, on choisit par exemple une somme linéaire pondérée de caractéristiques
- $eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) = \sum_{i=1}^{n} w_i f_i(s)$
- Exemple : $w_1 = 10$ et
  $f_1 =$ nombre de dames blanches – nombre de dames noires



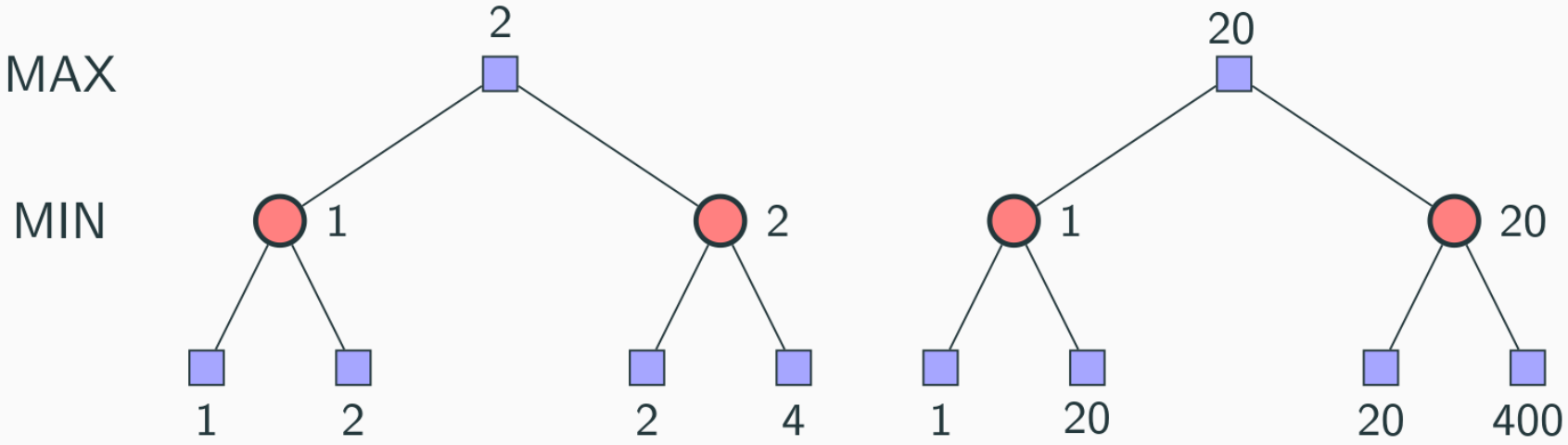
(a) White to move (b) White to move

- Aux échecs, on choisit par exemple une somme linéaire pondérée de caractéristiques
- $eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) = \sum_{i=1}^{n} w_i f_i(s)$
- Exemple : $w_1 = 10$ et
  $f_1 =$ nombre de dames blanches – nombre de dames noires
- Apprentissage des valeurs de w_i à partir d'exemples



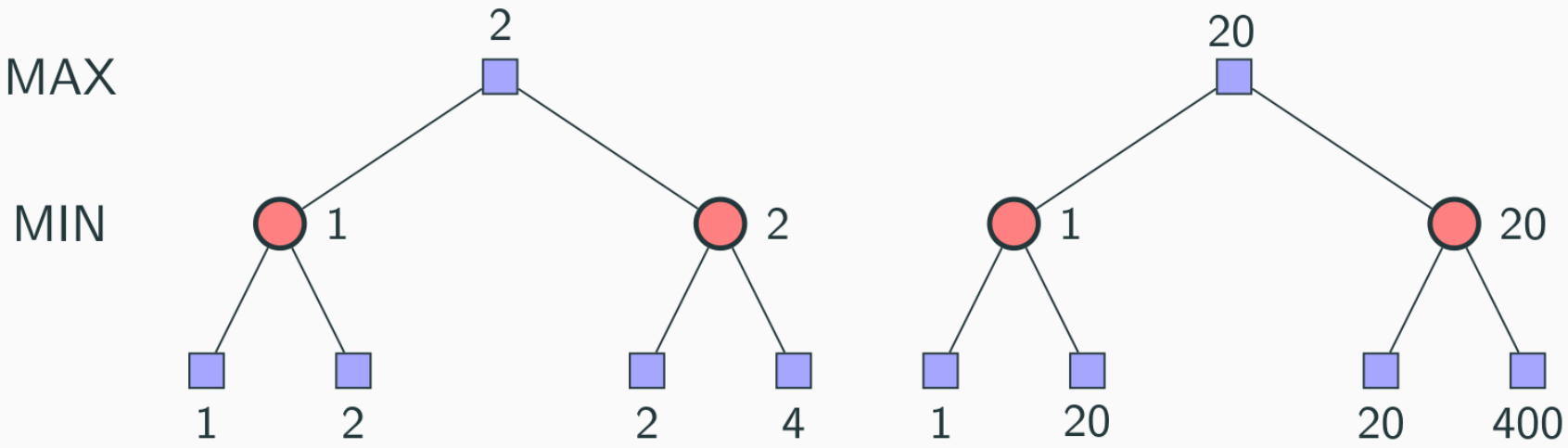
(a) White to move (b) White to move

- Aux échecs, on choisit par exemple une somme linéaire pondérée de caractéristiques
- $eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) = \sum_{i=1}^{n} w_i f_i(s)$
- Exemple : $w_1 = 10$ et
  $f_1 =$ nombre de dames blanches – nombre de dames noires
- Apprentissage des valeurs de w_i à partir d'exemples
- Deep Blue utilisait environ 6 000 caractéristiques



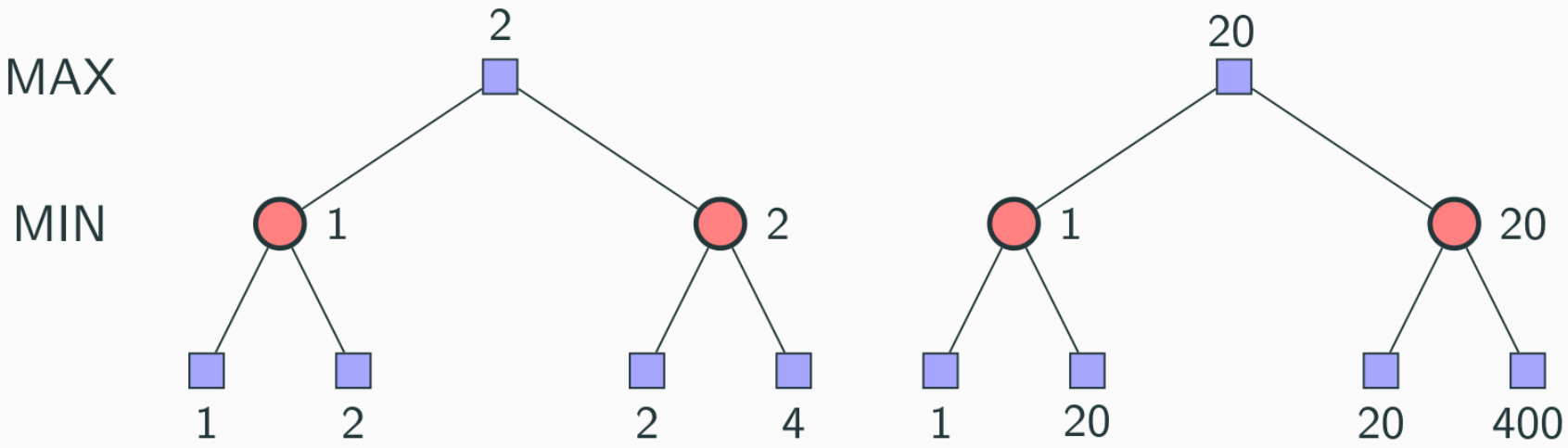
- Pour $\alpha - \beta$, la valeur exacte des nœuds n'est pas importante :

Ressources limitées v.2

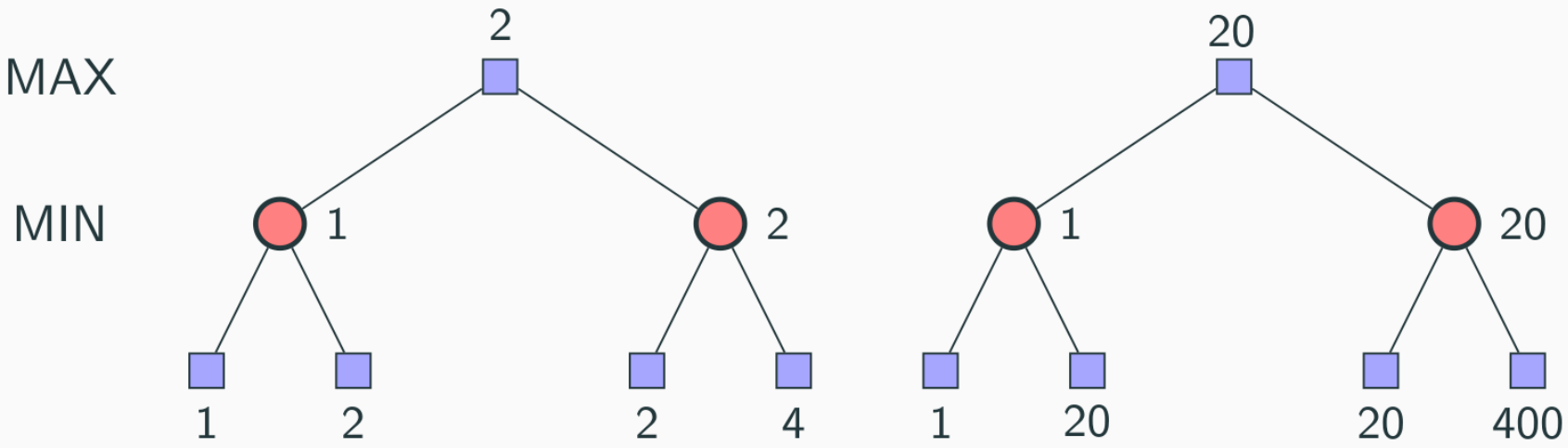


- Pour $\alpha - \beta$, la valeur exacte des nœuds n'est pas importante :
  - Seul l'ordre dans lequel on visite les nœuds est important

Ressources limitées v.2



- Pour $\alpha - \beta$, la valeur exacte des nœuds n'est pas importante :
  - Seul l'ordre dans lequel on visite les nœuds est important
  - Le comportement est préservé pour chaque transformation monotone de la fonction `eval`



- Pour $\alpha - \beta$, la valeur exacte des nœuds n'est pas importante :
  - Seul l'ordre dans lequel on visite les nœuds est important
  - Le comportement est préservé pour chaque transformation monotone de la fonction `eval`
  - Pour les jeux déterministes, la fonction de résultat se comporte comme une fonction d'utilité ordinale

$\alpha - \beta$: Stratégies

- L'ordre dans lequel on visite les nœuds fils est important

$\alpha - \beta$: Stratégies

- L'ordre dans lequel on visite les nœuds fils est important
- Si on trouve rapidement une bonne valeur, on élague plus de nœuds

$\alpha - \beta$: Stratégies

- L'ordre dans lequel on visite les nœuds fils est important
- Si on trouve rapidement une bonne valeur, on élague plus de nœuds
- On peut trier les fils par leur utilité

$\alpha - \beta$: Stratégies

- L'ordre dans lequel on visite les nœuds fils est important
- Si on trouve rapidement une bonne valeur, on élague plus de nœuds
- On peut trier les fils par leur utilité
- D'autres améliorations sont possible

$\alpha - \beta$: Stratégies

- L'ordre dans lequel on visite les nœuds fils est important
- Si on trouve rapidement une bonne valeur, on élague plus de nœuds
- On peut trier les fils par leur utilité
- D'autres améliorations sont possible
- Aux échecs on utilise :

$\alpha - \beta$: Stratégies

- L'ordre dans lequel on visite les nœuds fils est important
- Si on trouve rapidement une bonne valeur, on élague plus de nœuds
- On peut trier les fils par leur utilité
- D'autres améliorations sont possible
- Aux échecs on utilise :
    1. au début du jeu des bases de données d'ouverture

$\alpha - \beta$: Stratégies

- L'ordre dans lequel on visite les nœuds fils est important
- Si on trouve rapidement une bonne valeur, on élague plus de nœuds
- On peut trier les fils par leur utilité
- D'autres améliorations sont possible
- Aux échecs on utilise :
  1. au début du jeu des bases de données d'ouverture
  2. au milieu $\alpha - \beta$

$\alpha - \beta$: Stratégies

- L'ordre dans lequel on visite les nœuds fils est important
- Si on trouve rapidement une bonne valeur, on élague plus de nœuds
- On peut trier les fils par leur utilité
- D'autres améliorations sont possible
- Aux échecs on utilise :
  1. au début du jeu des bases de données d'ouverture
  2. au milieu $\alpha - \beta$
  3. à la fin des algorithmes spéciaux

- $b > 300$, donc la plupart des programmes utilisaient des bases de connaissances permettant de suggérer les coups les plus probables

Retour sur le Go

- $b > 300$, donc la plupart des programmes utilisaient des bases de connaissances permettant de suggérer les coups les plus probables
- Algorithme de AlphaGo :

Retour sur le Go

- $b > 300$, donc la plupart des programmes utilisaient des bases de connaissances permettant de suggérer les coups les plus probables
- Algorithme de AlphaGo :
  - apprentissage par imitation d'une fonction qui à une situation associe plusieurs coups possibles avec leurs probabilités

Retour sur le Go

- $b > 300$, donc la plupart des programmes utilisaient des bases de connaissances permettant de suggérer les coups les plus probables
- Algorithme de AlphaGo :
  - apprentissage par imitation d'une fonction qui à une situation associe plusieurs coups possibles avec leurs probabilités
  - apprentissage par jeu contre soi-même : le réseau joue contre lui même et augmente peu à peu (par montée de gradient sur la probabilité de gagner) les probabilités de jouer les meilleurs coups

Retour sur le Go

- $b > 300$, donc la plupart des programmes utilisaient des bases de connaissances permettant de suggérer les coups les plus probables
- Algorithme de AlphaGo :
  - apprentissage par imitation d'une fonction qui à une situation associe plusieurs coups possibles avec leurs probabilités
  - apprentissage par jeu contre soi-même : le réseau joue contre lui même et augmente peu à peu (par montée de gradient sur la probabilité de gagner) les probabilités de jouer les meilleurs coups
  - apprentissage de fonction d'évaluation

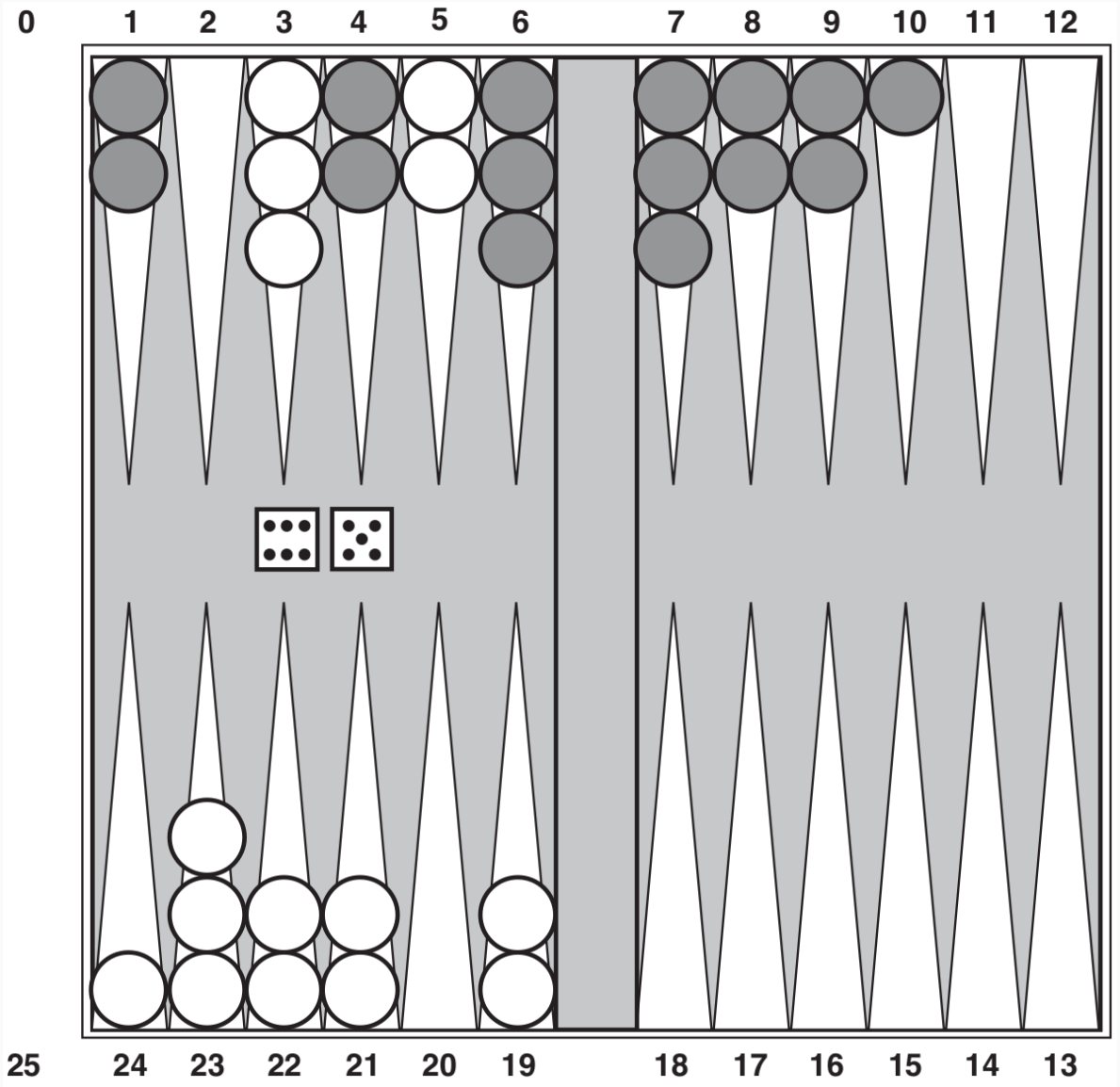
Retour sur le Go

- $b > 300$, donc la plupart des programmes utilisaient des bases de connaissances permettant de suggérer les coups les plus probables
- Algorithme de AlphaGo :
  - apprentissage par imitation d'une fonction qui à une situation associe plusieurs coups possibles avec leurs probabilités
  - apprentissage par jeu contre soi-même : le réseau joue contre lui même et augmente peu à peu (par montée de gradient sur la probabilité de gagner) les probabilités de jouer les meilleurs coups
  - apprentissage de fonction d'évaluation
  - fouille d'arbre Monte Carlo

Jeux

Jeux non-déterministes

Backgammon



Jeux non-déterministes

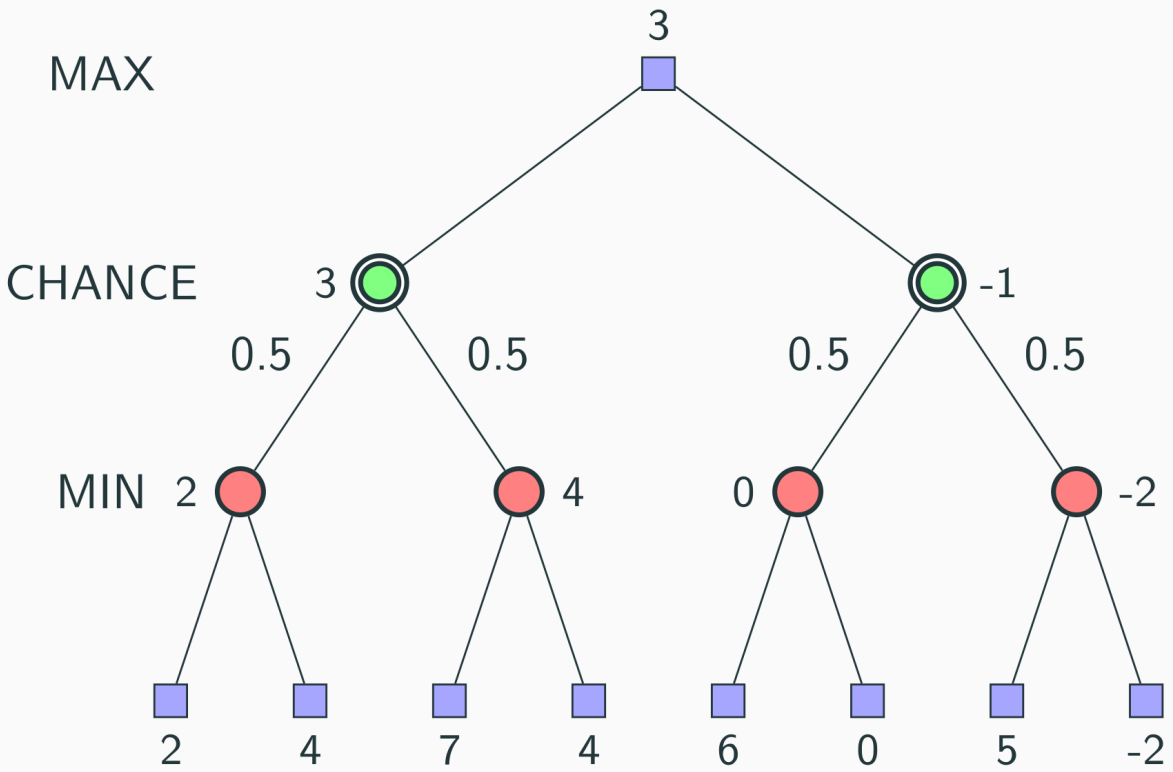
- Dans un jeu non-déterministe, la chance est introduite par un lancer de dés, une distribution de cartes, ...

Jeux non-déterministes

- Dans un jeu non-déterministe, la chance est introduite par un lancer de dés, une distribution de cartes, ...
- Exemple simplifié avec le lancer d'une pièce de monnaie :

Jeux non-déterministes

- Dans un jeu non-déterministe, la chance est introduite par un lancer de dés, une distribution de cartes, ...
- Exemple simplifié avec le lancer d'une pièce de monnaie :



Expectiminimax

- Expectiminimax généralise minimax en ajoutant la gestion de nœud de chance

Expectiminimax

- Expectiminimax généralise minimax en ajoutant la gestion de nœud de chance
- L'algorithme minimax est modifié avec les lignes suivantes :

Expectiminimax

- Expectiminimax généralise minimax en ajoutant la gestion de nœud de chance
- L'algorithme minimax est modifié avec les lignes suivantes :
  - if state is a Max node then
    return the highest Expectiminimax-Value of Successors(state)

Expectiminimax

- Expectiminimax généralise minimax en ajoutant la gestion de nœud de chance
- L'algorithme minimax est modifié avec les lignes suivantes :
  - if state is a Max node then
    return the highest Expectiminimax-Value of Successors(state)
  - if state is a Min node then
    return the lowest Expectiminimax-Value of Successors(state)

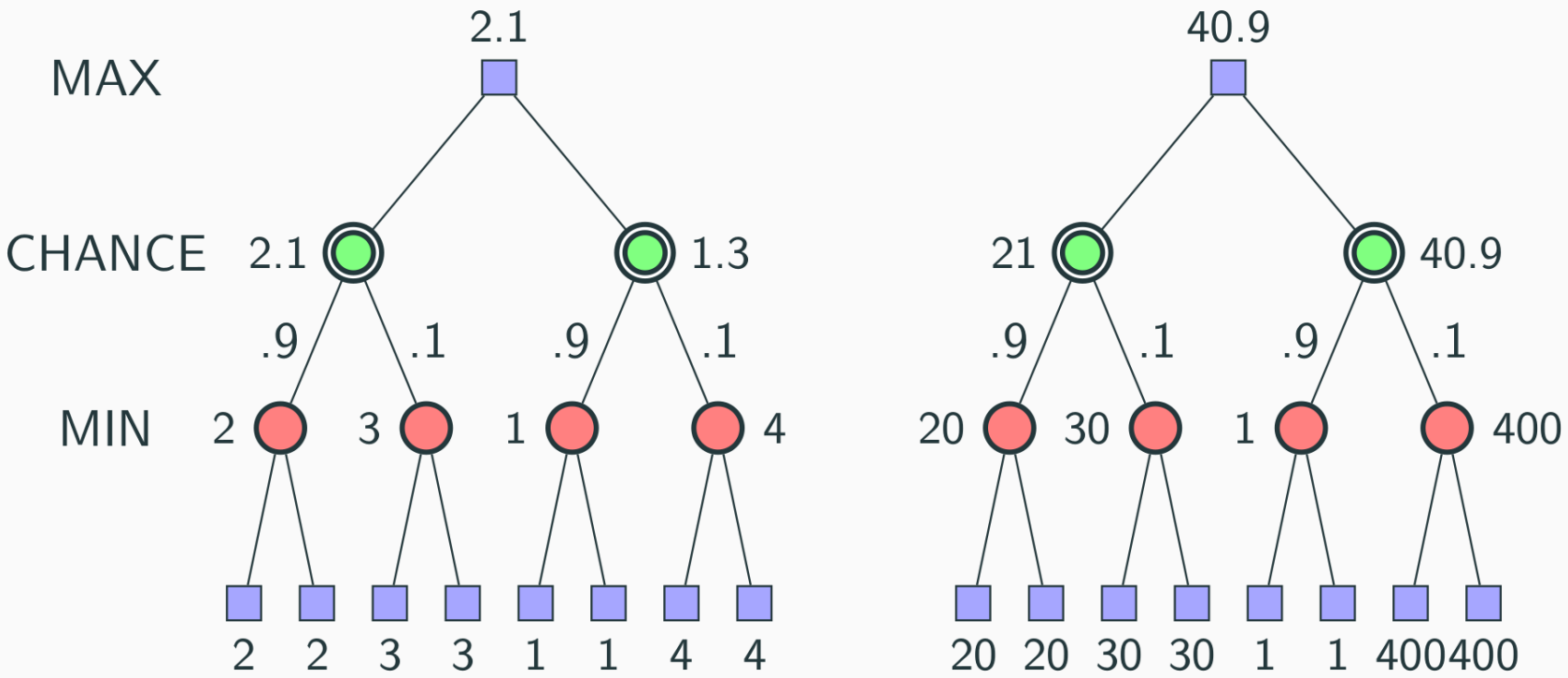
Expectiminimax

- Expectiminimax généralise minimax en ajoutant la gestion de nœud de chance
- L'algorithme minimax est modifié avec les lignes suivantes :
  - if state is a Max node then
    return the highest Expectiminimax-Value of Successors(state)
  - if state is a Min node then
    return the lowest Expectiminimax-Value of Successors(state)
  - if state is a Chance node then
    return average Expectiminimax-Value of Successors(state)

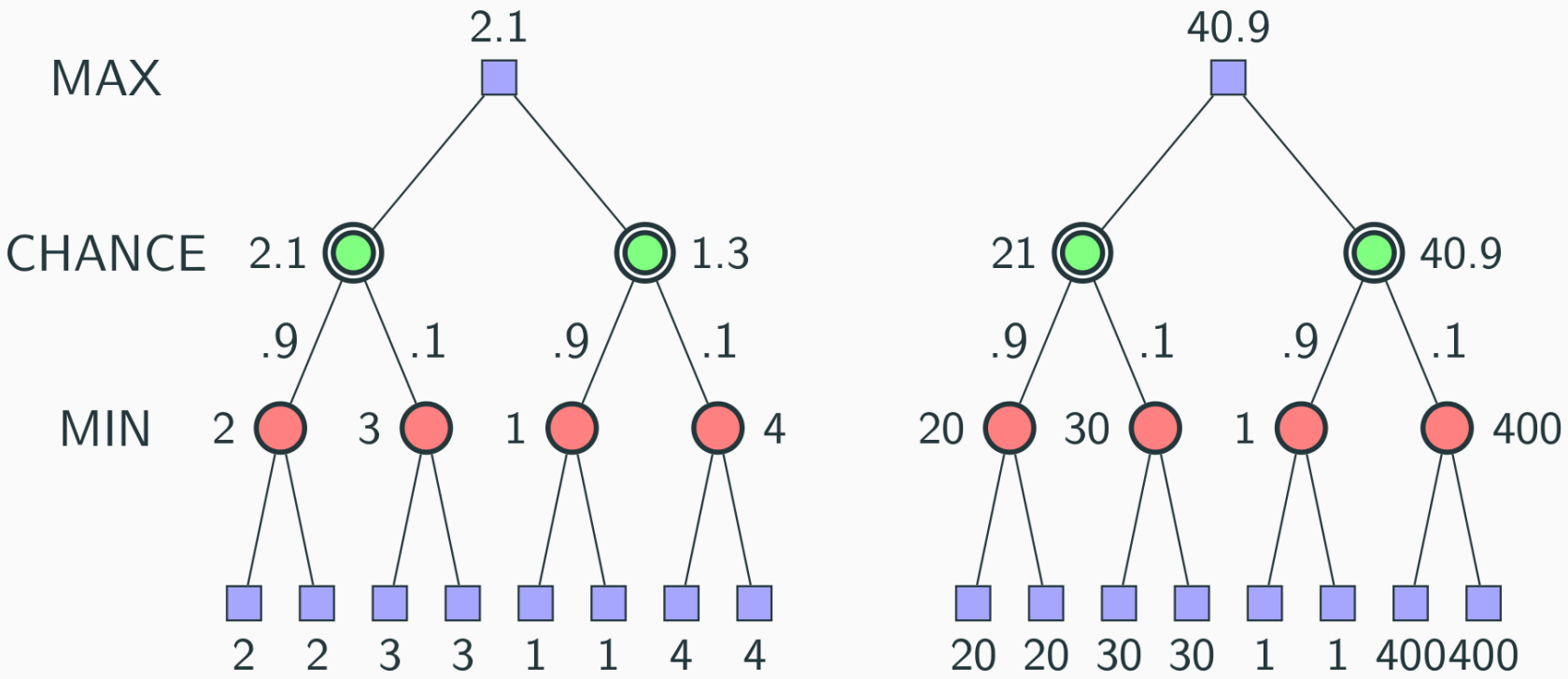
Expectiminimax

- Expectiminimax généralise minimax en ajoutant la gestion de nœud de chance
- L'algorithme minimax est modifié avec les lignes suivantes :
  - if state is a Max node then
    return the highest Expectiminimax-Value of Successors(state)
  - if state is a Min node then
    return the lowest Expectiminimax-Value of Successors(state)
  - if state is a Chance node then
    return average Expectiminimax-Value of Successors(state)
- Si `eval` est bornée, $\alpha - \beta$ peut-être adapté

La valeur exacte des nœuds est importante

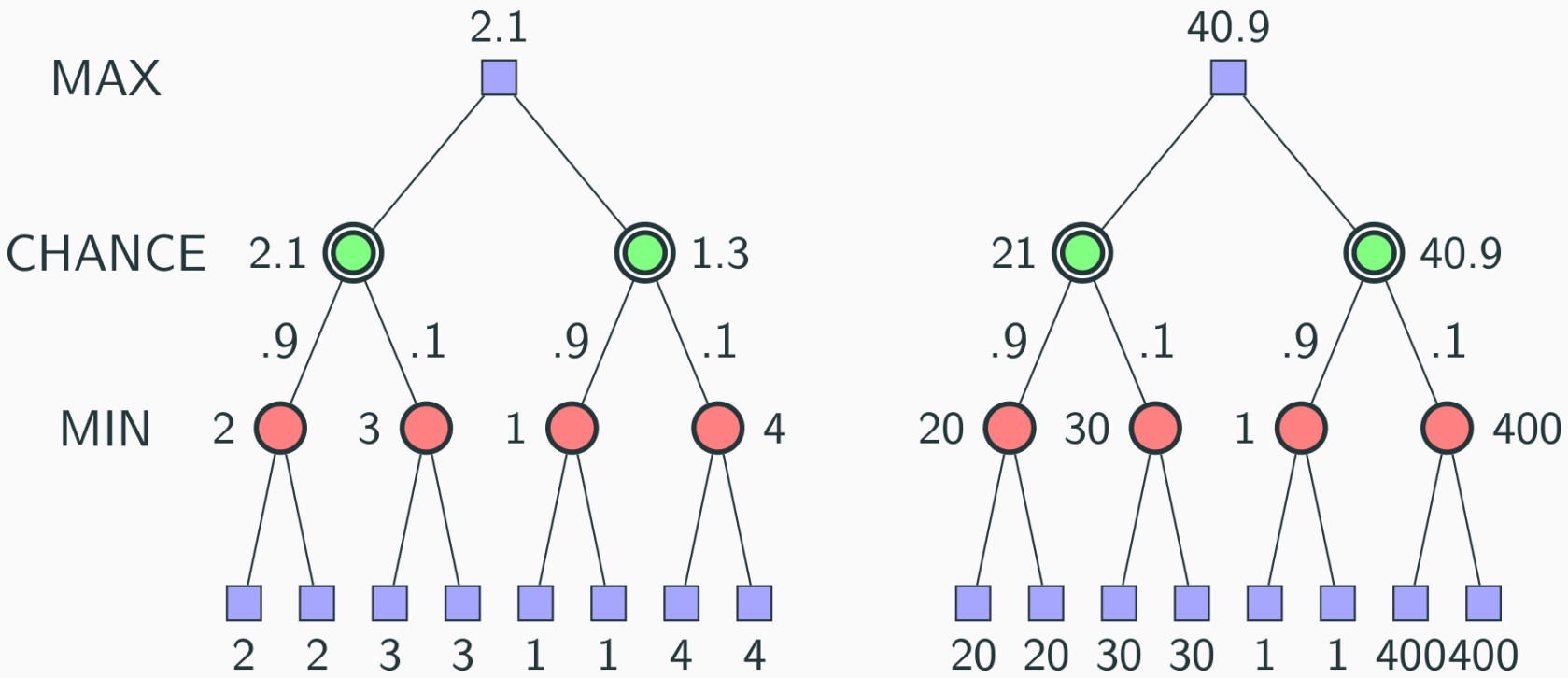


La valeur exacte des nœuds est importante



- Le comportement est préservé seulement pour chaque transformation positive et linéaire de la fonction

La valeur exacte des nœuds est importante



- Le comportement est préservé seulement pour chaque transformation positive et linéaire de la fonction
- `eval` doit être proportionnelle au gain attendu

Jeux

Jeux à information imparfaite

Jeux à information imparfaite

- Par exemple, jeux de cartes où les cartes de l'adversaire ne sont pas connues

Jeux à information imparfaite

- Par exemple, jeux de cartes où les cartes de l'adversaire ne sont pas connues
- Il est possible de calculer une probabilité pour chaque distribution de cartes possible

Jeux à information imparfaite

- Par exemple, jeux de cartes où les cartes de l'adversaire ne sont pas connues
- Il est possible de calculer une probabilité pour chaque distribution de cartes possible
- Ressemble à un gros dé lancé au début du jeu

Jeux à information imparfaite

- Par exemple, jeux de cartes où les cartes de l'adversaire ne sont pas connues
- Il est possible de calculer une probabilité pour chaque distribution de cartes possible
- Ressemble à un gros dé lancé au début du jeu
- **Idée** : calculer la valeur Minimax de chaque action dans chaque distribution possible, puis choisir l'action qui la valeur espérée maximale parmi toutes les distributions

Jeux à information imparfaite

- Par exemple, jeux de cartes où les cartes de l'adversaire ne sont pas connues
- Il est possible de calculer une probabilité pour chaque distribution de cartes possible
- Ressemble à un gros dé lancé au début du jeu
- **Idée** : calculer la valeur Minimax de chaque action dans chaque distribution possible, puis choisir l'action qui la valeur espérée maximale parmi toutes les distributions
- **Cas spécial** : si une action est optimale pour toutes les distributions, c'est une action optimale