

IA02 : Résolution de Problèmes et Programmation Logique

Algorithmes pour les Jeux de Stratégie

Sylvain Lagrue

sylvain.lagrue@hds.utc.fr – <https://www.hds.utc.fr/~lagruesy>.

Plan du cours

- Introduction
- L'algorithme min-max
- L'algorithme α - β
- Les algorithmes de type Monte Carlo pour les jeux
- Implémentation et structures de données
- Le *General Game Playing*

Définition

“

*Un jeu est un système dans lequel des entités **joueurs** s'engagent dans un conflit artificiel, défini par des **règles** et menant à un **gain** quantifiable.*

— Katie Salen et Eric Zimmerman

Les jeux de stratégie

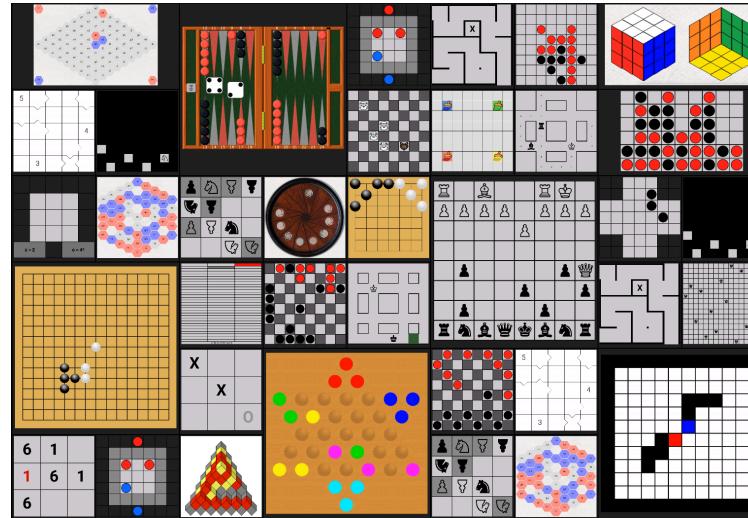
- Archéotype de l'intelligence humaine : dans les jeux de stratégie, les capacités physiques ne sont pas nécessaires, seules l'intelligence, la concentration, les connaissances et l'expérience le sont

Applications

- Loisirs
- Comportement d'agent économique (théorie des jeux)
- Systèmes d'aide à la décision
- Éducation (ex. les jeux sérieux)
- Systèmes multi-agents

Exemples de jeux et taxonomie

- Jeux à information complète : Échecs, Go, Dames, Xiangqi
- Jeux de hasard : Backgammon, Risk, Yam's
- Jeux à information incomplète : Poker, Bridge, Coinche
- Jeux simultanés : pierre/papier/ciseaux
- Jeux à somme nulle : tous les jeux cités précédemment
- Mais aussi : jeux asymétriques, indéterministes, collaboratifs, à somme non-nulles, etc.



Dans ce cours aujourd'hui : **jeux séquentiels, à information complète et à somme nulle.**

Pour les chercheurs en IA

“

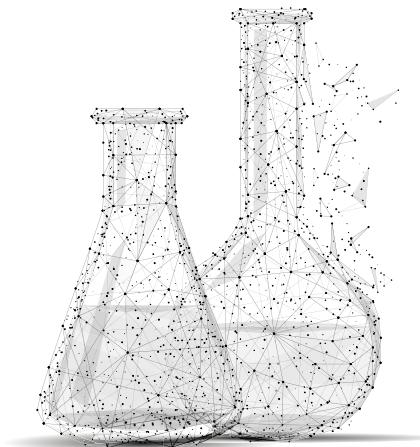
Les échecs sont la drosophile de l'Intelligence Artificielle.

— Alexander Kronrod (1921-1986)



Jeux de stratégie

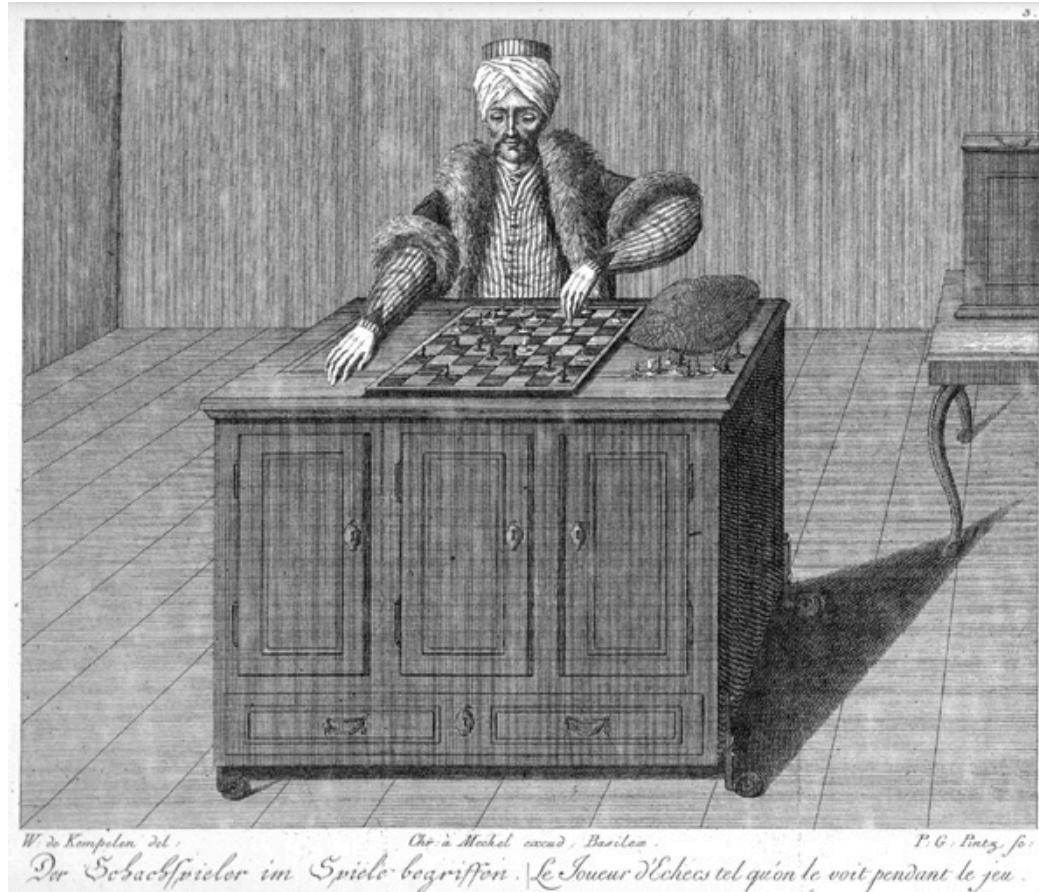
- Environnement contrôlé (pas de contraintes physiques, règles fixées, joueurs "rationnels")
- Bac à sable pour expérimenter des algorithmes et des architectures
- Vitrine technologique



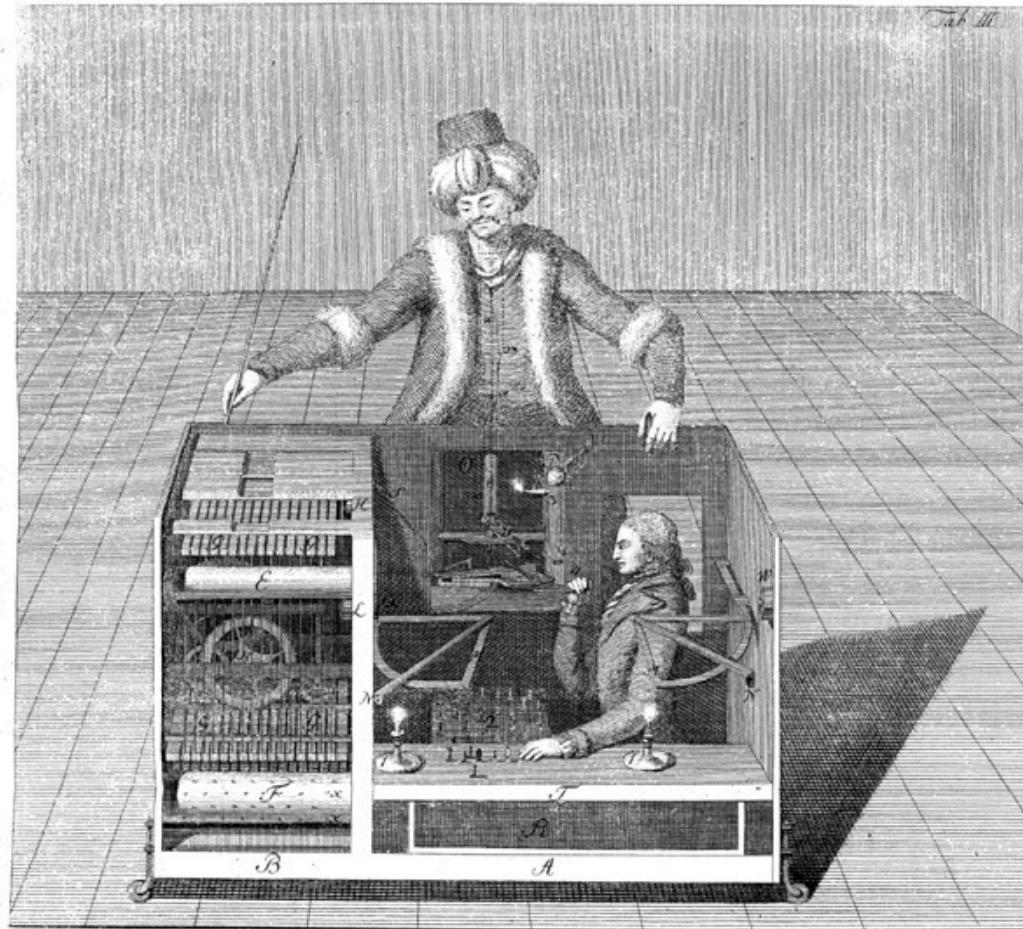
Pour le grand public : fascination...



Le turc (1770, Johann Wolfgang von Kempelen)



Le turc (1770, Johann Wolfgang von Kempelen)

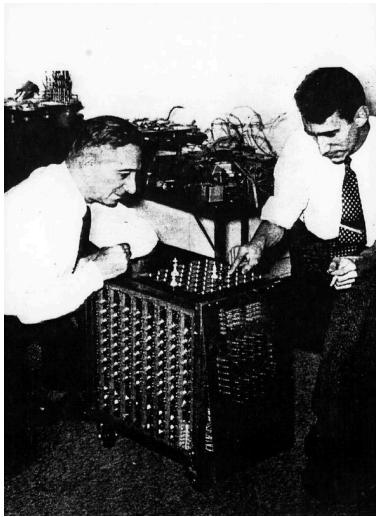


Timeline et jalons

- 1944 *Theory of Games and Economic Behavior* (Von Neumann et Morgenstern)
- 1949 Article: *Programming a Computer for Playing Chess* (Claude Shannon)
- 1951 Turing propose son programme pour jouer aux échecs
- 1950 *Équilibre de Nash* (John Nash)
- 1979 *BKG 9.8*
- 1997 *Deep Blue*
- 2007 *Checker Solved*
- 2008 *Mogo*
- 2014 *Stockfish* devient le meilleur programme d'échecs du monde
- 2016 *AlphaGo*
- 2017 *Libratus*
- 2018 *AlphaZero*

Programming a Computer for Playing Chess (1949)

- Article séminal pour la programmation de jeux d'échecs par **Claude Shannon**
 - 2 algorithmes pour jouer
 - "Type A": force brute (adaptation de min-max)
 - "Type B": sélection *fine* des branches intéressantes
 - Shannon a aussi construit un automate capable de jouer certaines fins de coups aux échecs (jusqu'à 6 pièces)



Claude Shannon and the Chessmaster Edward Laske

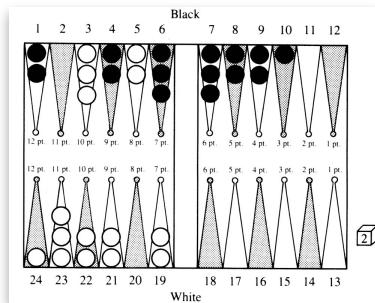
- 1951: **Alan Turing** proposa un programme (sur papier) capable de jouer aux échecs

BKG 9.8, Hans J. Berliner (1979)

- En 1979 BKG 9.8 bat le champion du monde de Backgammon, Luigi Villa, sur le score de 7–1
- *Idée principale* : utiliser la logique floue pour trouver la transition entre les 3 phases de jeu (ouverture, milieu de partie, final)



Hans J. Berliner



DEC PDP-10

Deep Blue (IBM) contre Garry Kasparov (1997)



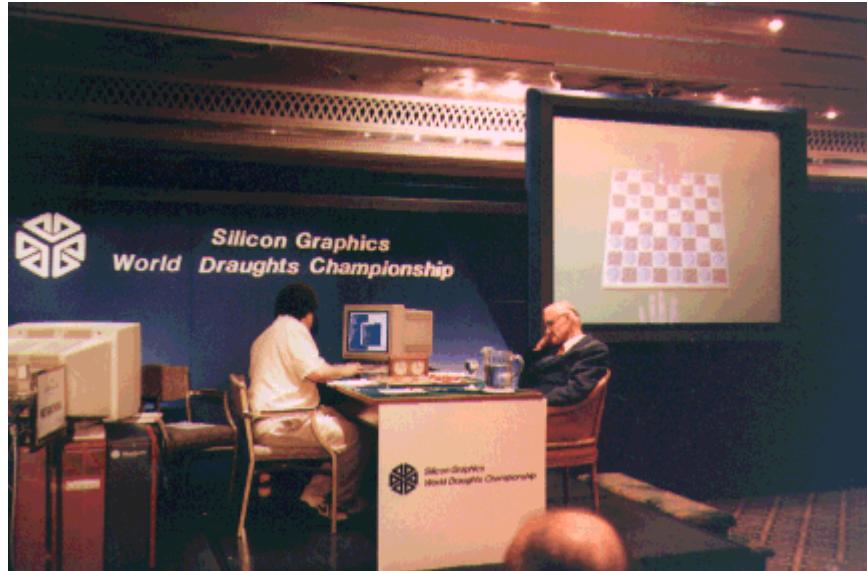
Deep Blue vs Garry Kasparov (3.5/2.5 - 2w/1w, 3 draws)

- Super-ordinateur massivement parallèle (256 CPU dédiés)
- 11.4 GFlop/s
- Évaluation de 200 millions de positions par seconde



Chinook résout les dames anglaises/checkers (2007)

- Jonathan Schaeffer et al.
- *Solved Checkers* : après une recherche exhaustive, une stratégie parfaite a été trouvée
- 18 années pendant lesquelles 50 à 200 machines (PC) ont tourné...



Contre Marion Tinsley en 1992 (4-2 et 33 nulles pour Tinsley)

Facteur de branchement:

- Checkers (8x8) = 2.8
- Échecs = 35
- Go (19x19) = 250

Monte Carlo Go (1992)

- 1ère utilisation du Monte Carlo Tree Search (MCTS) pour le Go (Bernd Brügmann)

MoGo (2008)

- Introduction de UCT (Upper bound Confidence for Tree = MCTS + UCB Upper Confidence Bounds)
- GRID-5000/UTN-Cluster (3 nodes/40 cores/100GB Ram)

Alpha Go (2016)

- Combine MCTS + réseaux de neurones profonds + apprentissage par renforcement (à partir de vraies parties et contre lui-même)
- Victoire contre le champion du monde Lee Sedol 4-1 (March 2016)
- Hardware : 1,202 CPUs et 176 GPUs

Libratus et le poker (2017)

- Tuomas Sandholm
- Victoire contre 4 joueurs professionnels de poker en **heads up no-limit Texas hold'em**
- Réseaux de neurones profonds + apprentissage par renforcement **from scratch** (en utilisant CFR+ – counterfactual regret minimization +)
- 15 millions "core hours" (1,712 années) de calcul



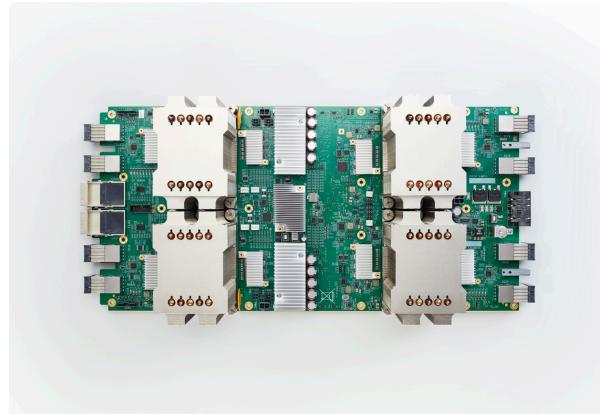
Alpha Zero (fin 2017)



Google DeepMind

Principes

- Combine MCTS + réseaux de neurones profonds + apprentissage par renforcement (**from scratch**)
- Victoire contre les meilleurs programmes aux échecs (*Stockfish*), Shogi (*elmo*) et Go (*Alpha Go*)
- Fonctionne sur un ordinateur composé de 4 TPUs (Tensor Processing Units)
- Évalue 80,000 positions par seconde vs 70,000,000 pour Stockfish 8
- Seulement 9 heures d'apprentissage pour battre Stockfish (3 jours pour battre Alpha Go Lee)



A Google Cloud TPU board

Alpha Zero (fin 2017)

Mais...

- En utilisant 5,064 TPUs (5000 de 1^{ère} gen. + 64 de 2^{nde} gen.) pour l'apprentissage
- $9 \times 5,064 \approx 5$ années de temps TPU
- $9 \times (5,000 \times 15 + 64 \times 30) \approx 79$ années de temps CPU (Intel Haswell)...



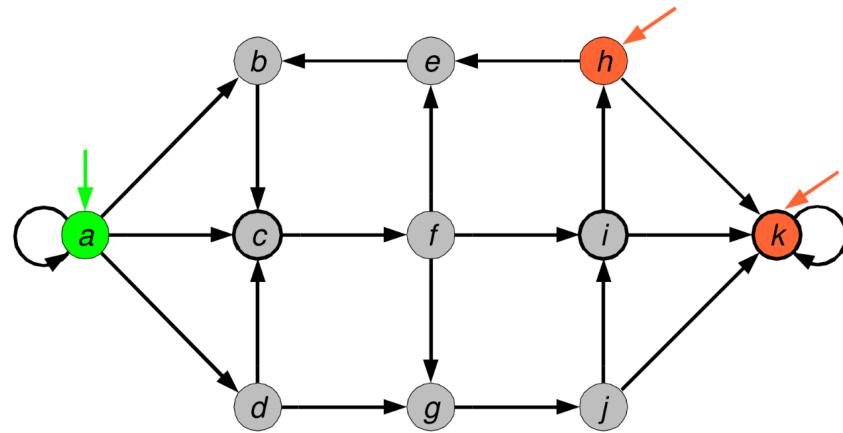
Algorithmique pour les jeux : min-max

Formalisation d'un jeu (séquentiel à 2 joueurs)

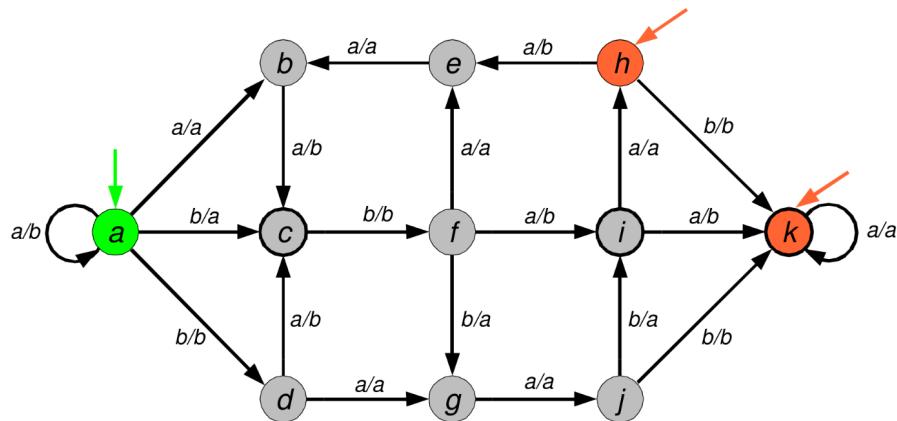
- $J = \{1, 2\}$: l'ensemble des joueurs
- Q : l'ensemble des états du jeu
- $u_0 \in Q$: l'état initial du jeu
- $S \subseteq Q$: l'ensemble des états terminaux
- $L : J \times Q \rightarrow 2^Q$: l'ensemble des états accessibles à chaque joueur à partir d'un état donné
- fonction gain $\mu : J \times Q \rightarrow \mathbb{R}$

On prendra souvent comme gains possibles $\{-1, 0, 1\}$ représentant respectivement une défaite, un nul ou une victoire pour le joueur 1 (une victoire un nul ou une défaite pour le joueur 2 dans le cas d'un jeu à somme nulle)

Représentation sous forme de système état/transition



Représentation sous forme de système état/transition (jeux simultanés)



L'algorithme min-max

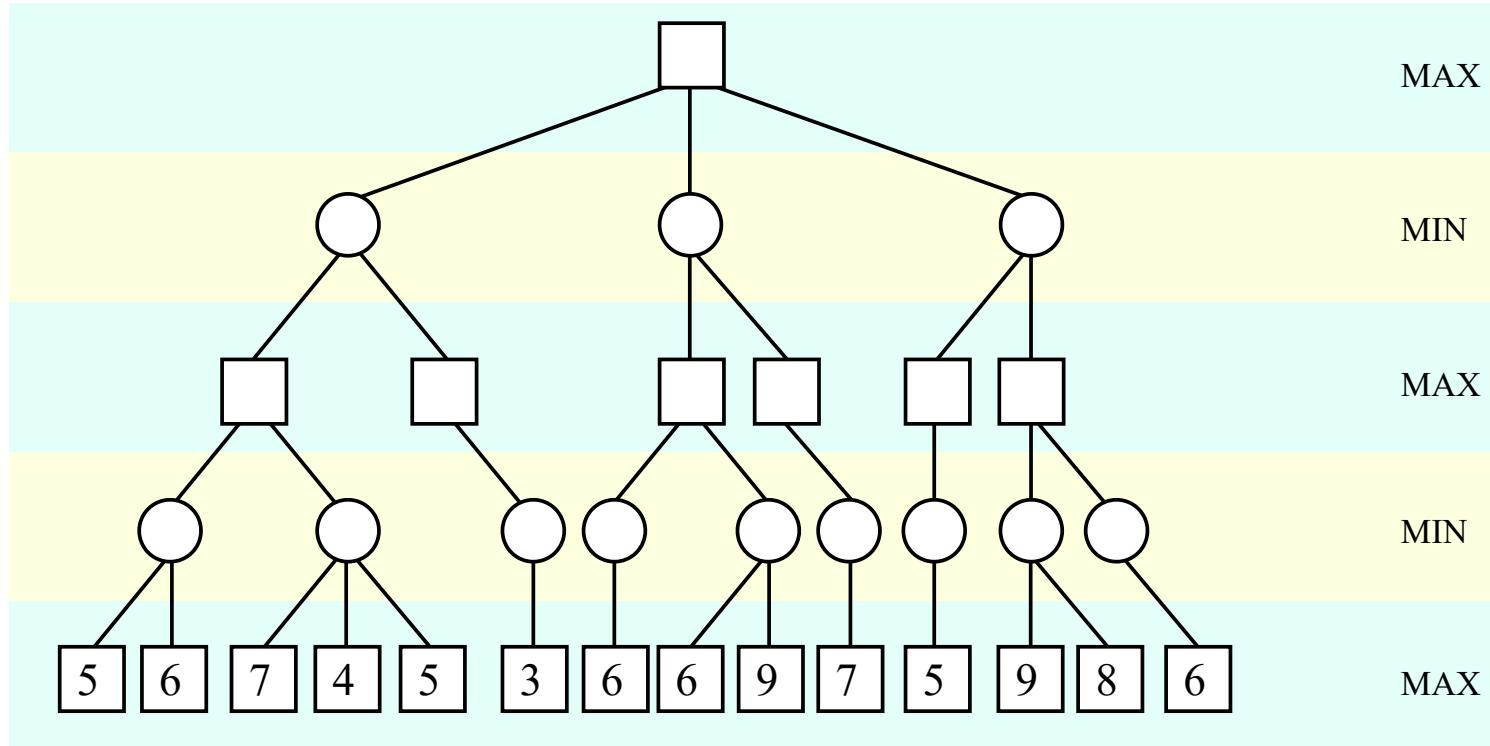
Cadre : jeux **séquentiels à 2 joueurs**, à **information complète** et à **somme nulle**
(c'est à dire un jeu dont la somme des gains des joueurs est toujours nulle)

- But du joueur 1 : maximiser le gain
- But du joueur 2 : minimiser le gain

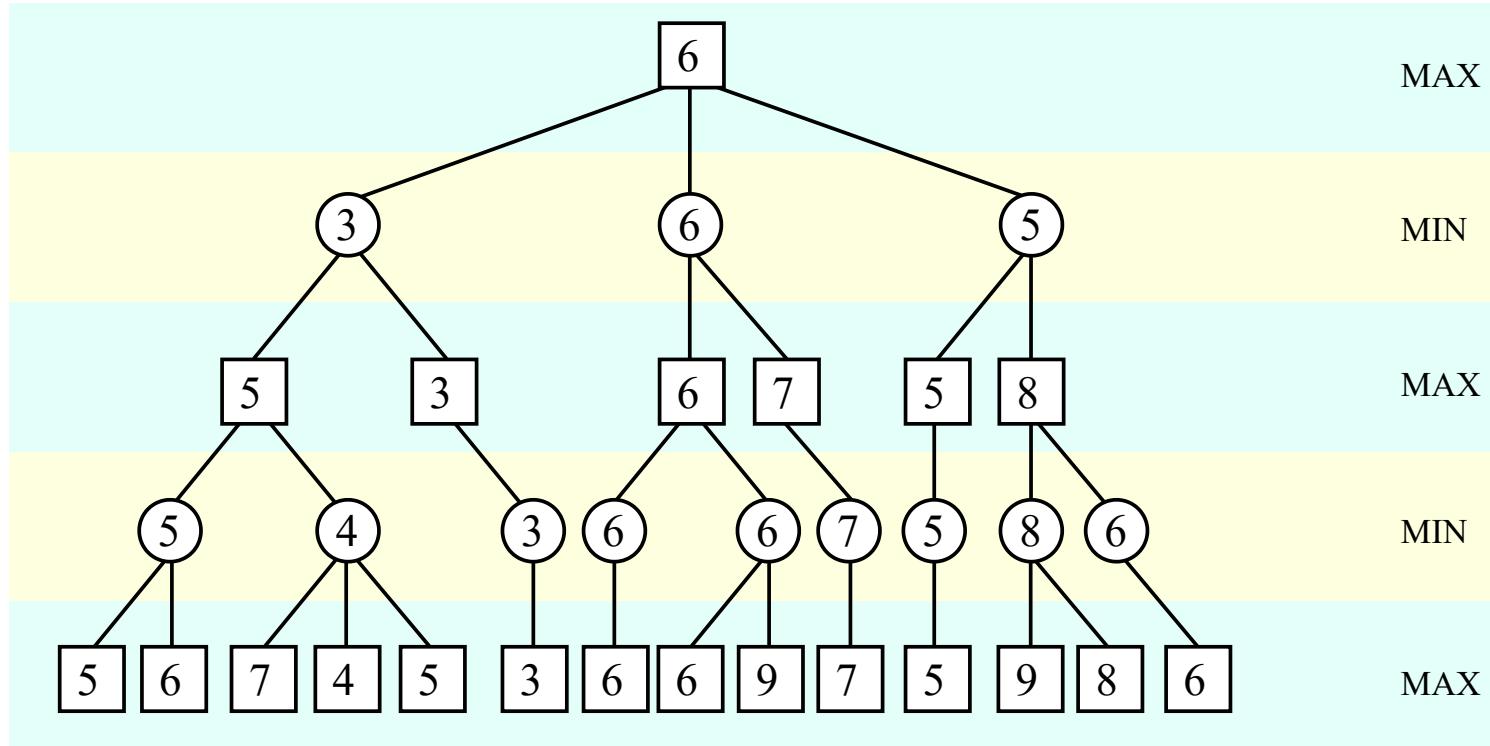
Pseudo-code

```
def minmax(node: State, maximizingPlayer: bool) -> float:  
    if node is a terminal node:  
        return  $\mu$ (node)  
  
    if maximizingPlayer:  
        bestValue =  $-\infty$   
        for each child of node  
            v = minmax(child, False)  
            bestValue = max(bestValue, v)  
        return bestValue  
  
    else: # minimizing player  
        bestValue :=  $+\infty$   
        for each child of node  
            v = minmax(child, True)  
            bestValue = min(bestValue, v)  
        return bestValue
```

Exemple



Exemple (suite)



Un autre exemple de jeu : le jeu des n allumettes

- Deux joueurs (1 et 2)
- Chaque joueur doit retirer 1, 2 ou 3 allumettes
- Les joueurs jouent à tour de rôle
- Le joueur qui retire la (ou les) dernière(s) allumette(s) a perdu
- Le gain sera de -1 (si 2 a gagné) et +1 (si 1 a gagné)

“

Pour tout jeu fini à deux joueurs, à somme nulle, il existe une valeur V et une stratégie pour chaque joueur telle que :

- *Quelle que soit la stratégie du joueur 2, le meilleur gain possible pour le joueur 1 est V*
- *Quelle que soit la stratégie du joueur 1, le meilleur gain possible pour le joueur 2 est $-V$*

Théorème 2 (von Neumann et Morgenstern)

“

Un joueur est dit rationnel s'il cherche toujours à maximiser (resp. minimiser) ses gains. Dans le cas d'un jeu fini à deux joueurs, à information complète et à somme nulle, un joueur rationnel fera toujours au moins aussi bien qu'un joueur irrationnel.

Theory of Games and Economic Behavior (1944)

- S'arrêter à une certaine profondeur $depth$
- Les états de profondeur $depth$ sont considérés comme des états buts
- Estimer la fonction de gain à cette profondeur à l'aide d'une **fonction d'évaluation**

Pseudo-code

```
def minmax(node, depth, maximizingPlayer) -> float:  
    if depth = 0 or node is a terminal node:  
        return the evaluation value of node  
  
    if maximizingPlayer:  
        bestValue = -∞  
        for each child of node:  
            v = minmax(child, depth - 1, False)  
            bestValue = max(bestValue, v)  
        return bestValue  
  
    else: # minimizing player  
        bestValue = +∞  
        for each child of node:  
            v = minmax(child, depth - 1, True)  
            bestValue = min(bestValue, v)  
        return bestValue
```

Exemples de fonction d'évaluation

Aux échecs :

- Dame = 9, Tour = 5, Fou = 3, Cavalier = 3, Pion = 1

À Othello/Reversi (Table statique) :

120	-20	20	5	5	20	-20	120
-20	-40	-5	-5	-5	-5	-40	-20
20	-5	15	3	3	15	-5	20
5	-5	3	3	3	3	-5	5
5	-5	3	3	3	3	-5	5
20	-5	15	3	3	15	-5	20
-20	-40	-5	-5	-5	-5	-40	-20
120	-20	20	5	5	20	-20	120

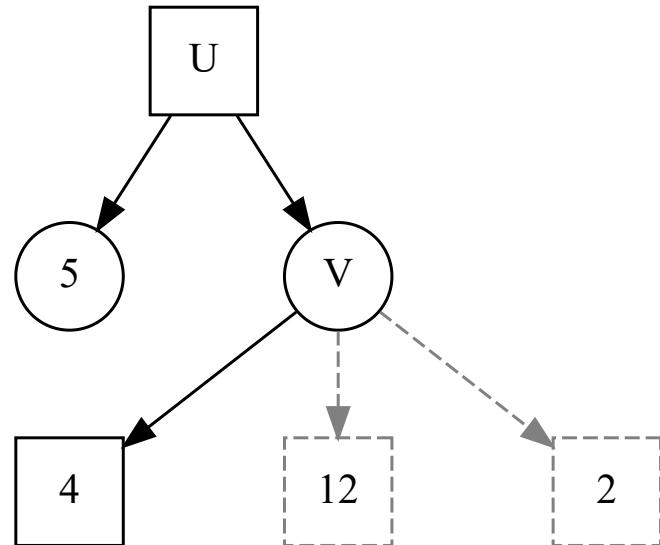
Quelques remarques...

- Effet d'horizon
- Comment concevoir une « bonne » fonction d'évaluation ?
 - Cas de Stockfish
 - Cas de Alphago

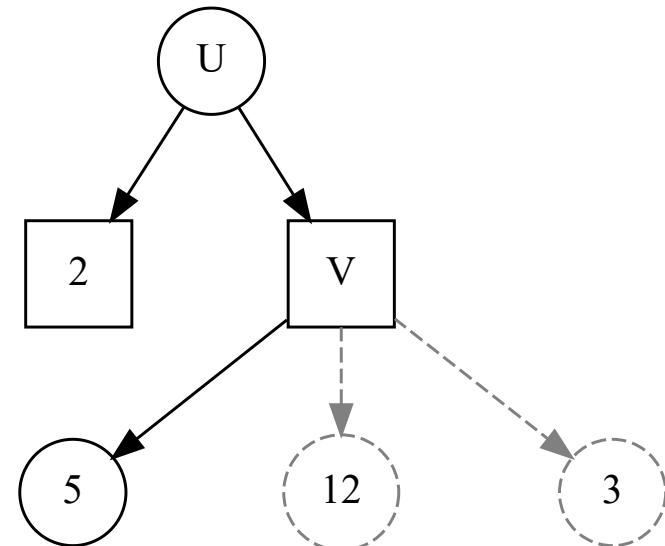
La coupure/l'élagage α - β

- Amélioration de l'algorithme *min-max*
- **Idée :** on stocke 2 valeurs
 - $\alpha =$ le plus grand des sélectionnés par le joueur Max parmi les nœuds fils
 - $\beta =$ le plus petit des sélectionnés par le joueur Min parmi les nœuds fils
 - Avec la contrainte $\alpha < \beta$

Coupure α



Coupure β

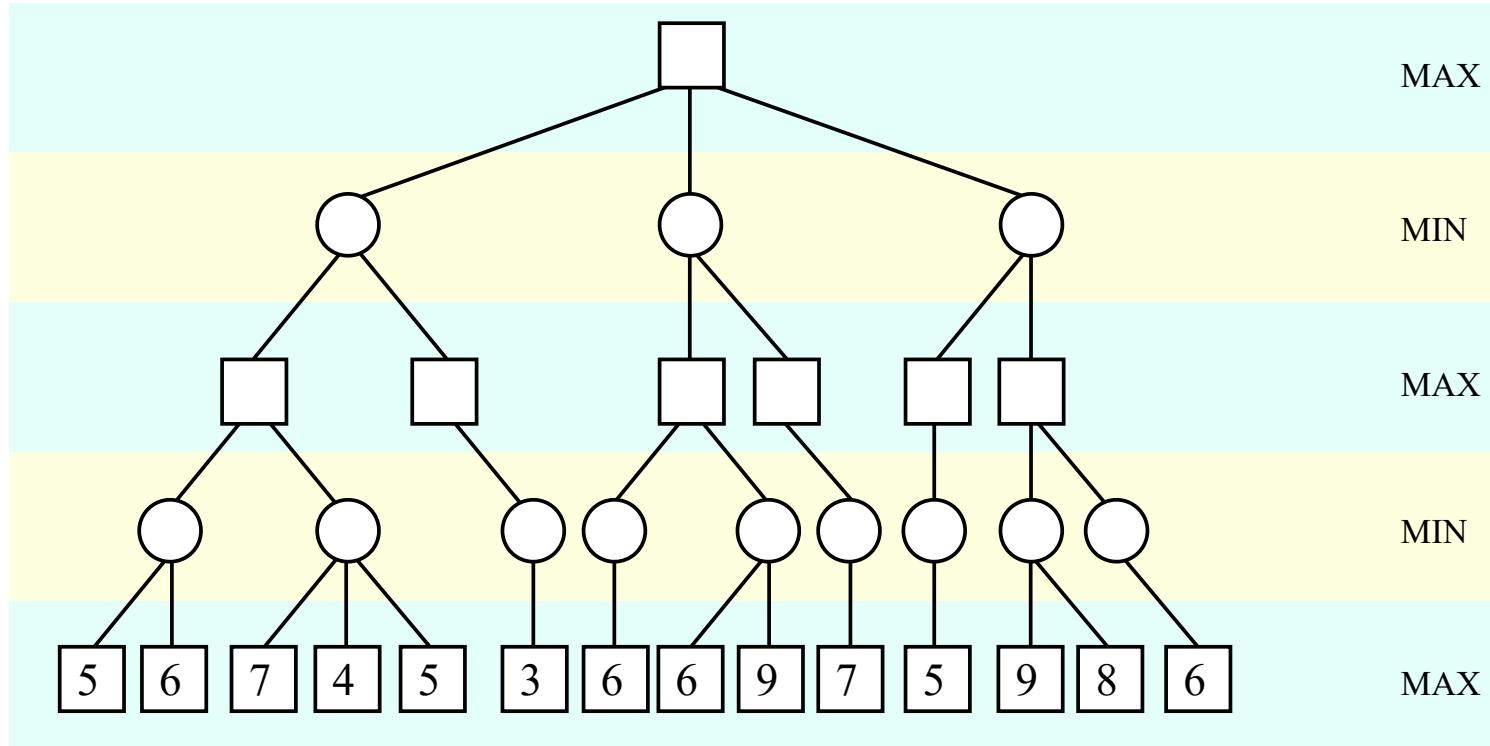


Pseudo-code (profondeur illimitée)

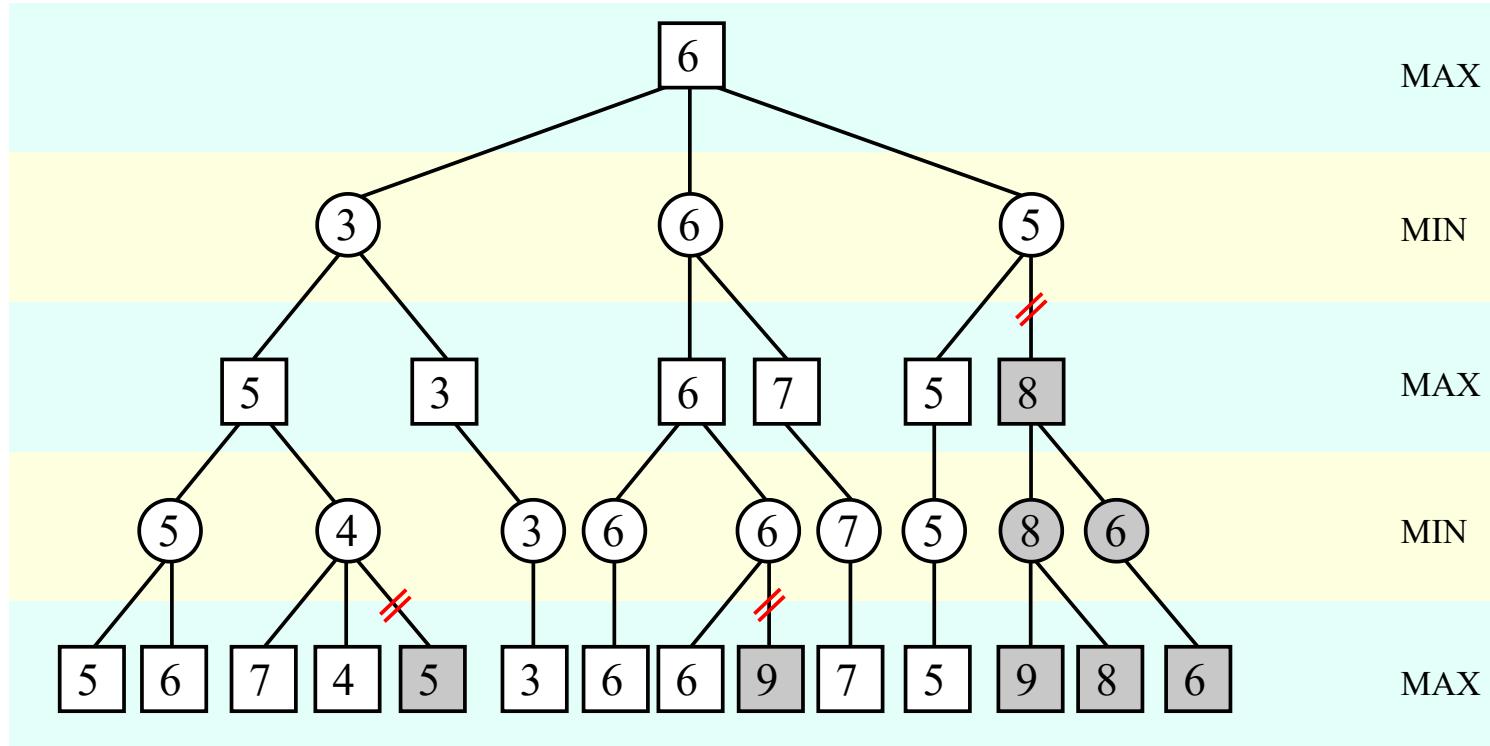
```
def alphabeta(node: State, α: float, β: float, maximizingPlayer: bool) -> float:
    if node is a terminal node:
        return μ(node)
    if maximizingPlayer:
        value = -∞
        for each child of node:
            value = max(value, alphabeta(child, α, β, FALSE))
            α = max(α, value)
            if α ≥ β:
                break # β cut-off
        return value
    else: # minimizing player
        value = +∞
        for each child of node:
            value = min(value, alphabeta(child, α, β, TRUE))
            β = min(β, value)
            if β ≤ α:
                break # α cut-off
        return value
```

```
alphabeta(u0, -∞, +∞, True)
```

Exemple



Exemple (suite)



Pseudo-code (profondeur limitée)

```
def alphabeta(node: State, depth: int, a: float,
              β: float, maximizingPlayer: bool) -> float:
    if depth == 0 or node is a terminal node:
        return the heuristic value of node
    if maximizingPlayer:
        value = -∞
        for each child of node:
            value = max(value, alphabeta(child, depth - 1, a, β, False))
            a = max(a, value)
            if a ≥ β:
                break # β cut-off
        return value
    else: # minimizing player
        value = +∞
        for each child of node:
            value = min(value, alphabeta(child, depth - 1, a, β, True))
            β = min(β, value)
            if a ≥ β:
                break # a cut-off
        return value
```

```
alphabeta(u0, depth, -∞, +∞, True)
```

IA02 : Résolution de Problèmes et Programmation Logique

Algorithmes de Monte-Carlo pour le jeu

Sylvain Lagrue

sylvain.lagrue@hds.utc.fr – <https://www.hds.utc.fr/~lagruesy>.

Algorithmes de Monte-Carlo pour le jeu

Introduction

- Algorithme de hachage de Zobrist
- Les bandits manchots
- Recherche arborescente de type Monte-Carlo MCTS

Introduction

“

Algorithmes de Monte-Carlo : algorithmes faisant intervenir le hasard

- L'aléatoire sur une machine n'existe pas (sauf dispositifs quantiques...)
 - On utilise des suites **pseudo-aléatoires**
 - Utilisées pour les simulations et le chiffrement
 - Besoins pour la simulation \neq besoins pour le chiffrement

Introduction

Générateurs aléatoires congruentiels linéaires

- Inventés en 1948 par D.H Lehmer.

$$u_{n+1} = (a \cdot u_n + b) \mod c$$

- On utilisera de préférence b et c premiers entre eux et c une puissance de 2.
- Les propriétés du générateur dépendent du choix des valeurs a , b et c
- u_0 est la graine (seed)

“

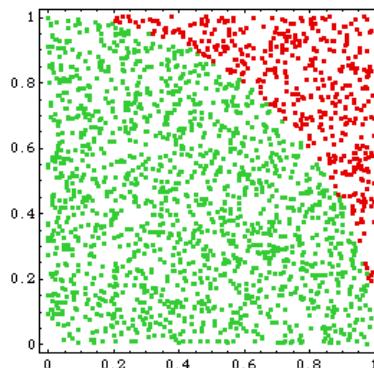
"Les générateurs de nombres aléatoires ne doivent pas être choisis au hasard." – D. Knuth

Algorithmes de Monte-Carlo pour le jeu

Introduction

Exemple : comment trouver la valeur de π avec un dé ?

- Considérons un quart de cercle de rayon 1
- On choisit un point au hasard (x, y) dans le carré $[1, 1]$
- On détermine s'il est dans le quart de cercle de rayon 1 c-à-d $x^2 + y^2 \leq 1$
- On recommence n fois



$$\pi = 4 \cdot \frac{m}{n}$$

Avec m le nombre de points à l'intérieur du cercle. ([Demo en ligne](#))

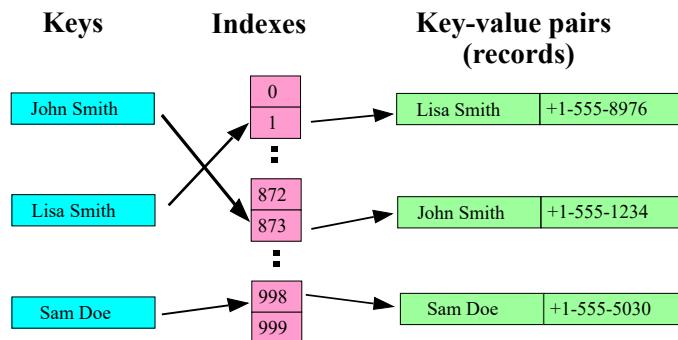
L'algorithme de hachage de Zobrist

Objectif

- Créer une fonction de hachage pour stocker efficacement les valeurs obtenues pour les états déjà parcourus

Quelques définitions (rappel)

- *Fonction de hachage* : fonction permettant de passer efficacement d'une donnée quelconque à un nombre (idéalement unique)
- *Table de hachage* : structure de données (généralement un tableau) utilisant une fonction de hachage pour accéder efficacement à un contenu indiqué par cette donnée (peut être utilisé comme dictionnaire, tableau associatif, etc.)
- *Collision* : même nombre pour 2 données différentes



Source : Wikipédia

L'algorithme de hachage de Zobrist

Utilisation dans les jeux

- Stocker et accéder efficacement aux valeurs de récompense pour les états déjà parcourus...

Principe de l'algorithme de hachage de Zobrist (1970)

- Combiner à l'aide de xor bits à bits des nombres pseudo-aléatoires

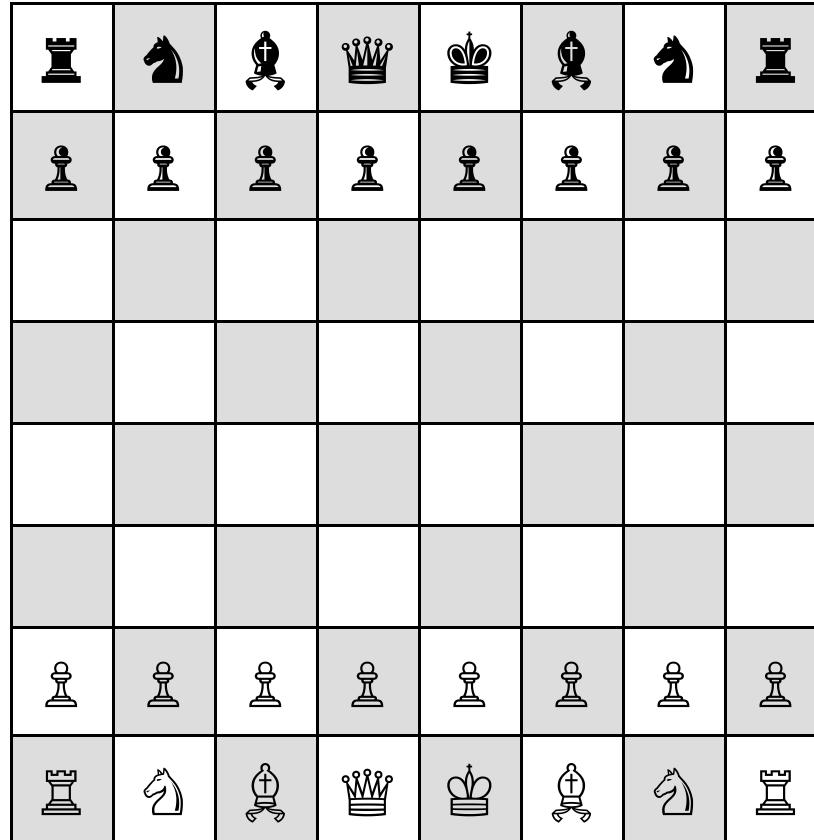
Exemple de xor bit à bit

```
1000 1100
xor 1101 0110 (clef)
-----
0101 1010
```

```
0101 1010
xor 1101 0110 (clef)
-----
1000 1100
```

L'algorithme de hachage de Zobrist : exemple des échecs

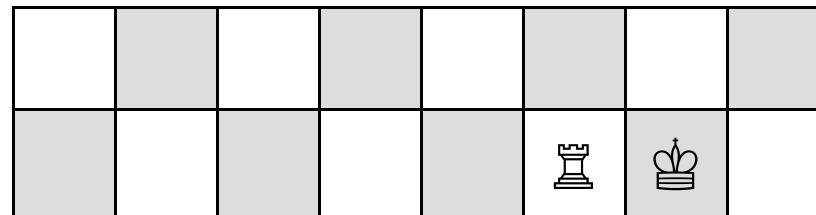
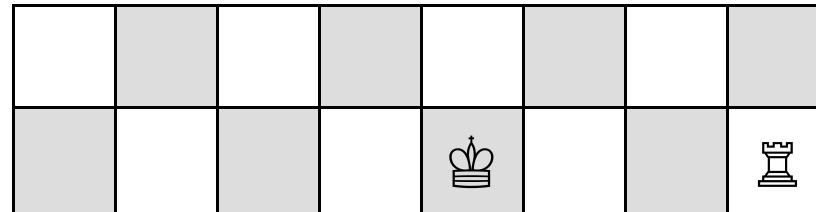
Combien de cases et de pièces différentes ?



⇒ 64 cases et 12 pièces différentes (      et     )

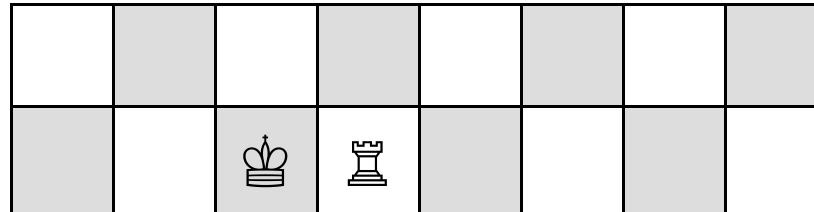
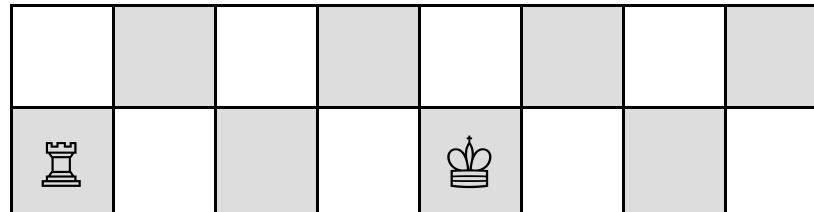
L'algorithme de hachage de Zobrist : exemple des échecs

Le petit roque



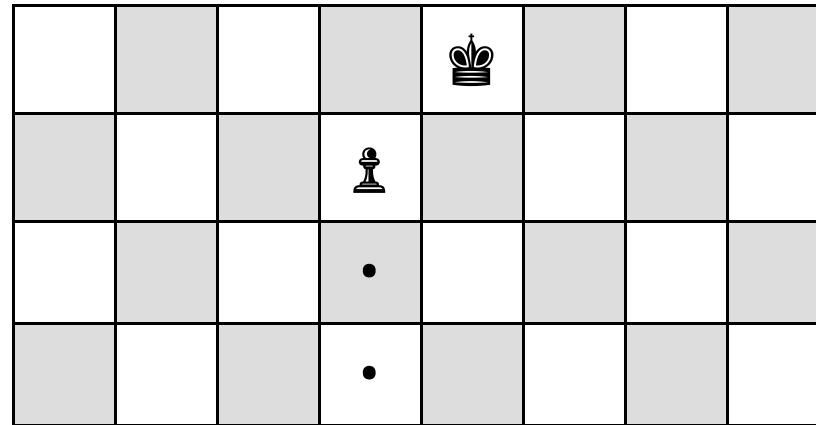
L'algorithme de hachage de Zobrist : exemple des échecs

Le grand roque

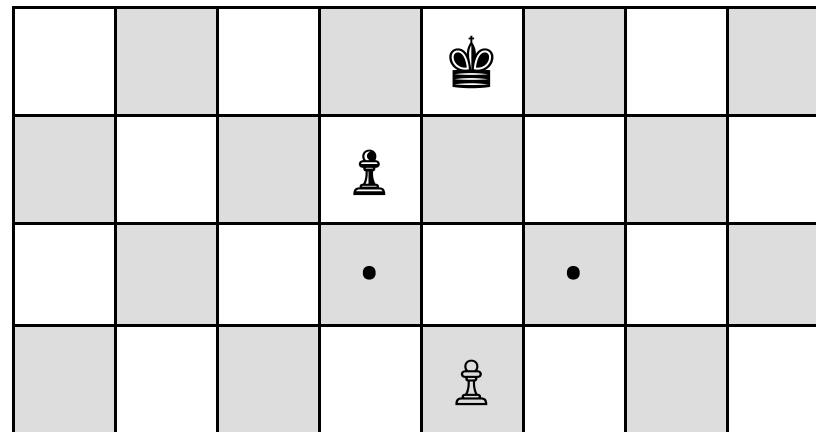


L'algorithme de hachage de Zobrist : exemple des échecs

Déplacement du pion

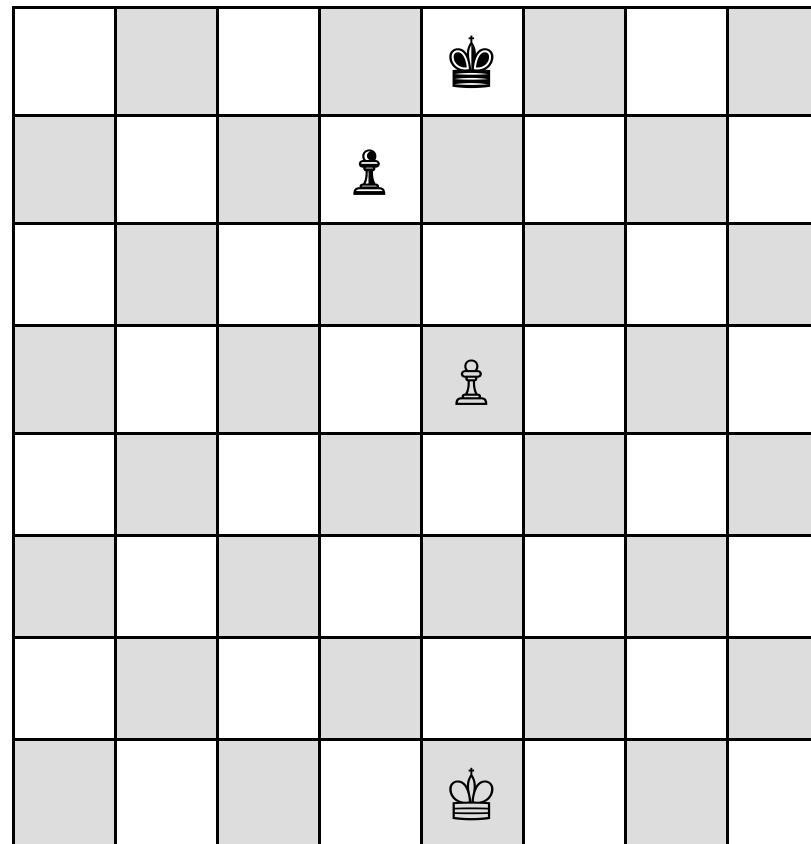


Prise du pion



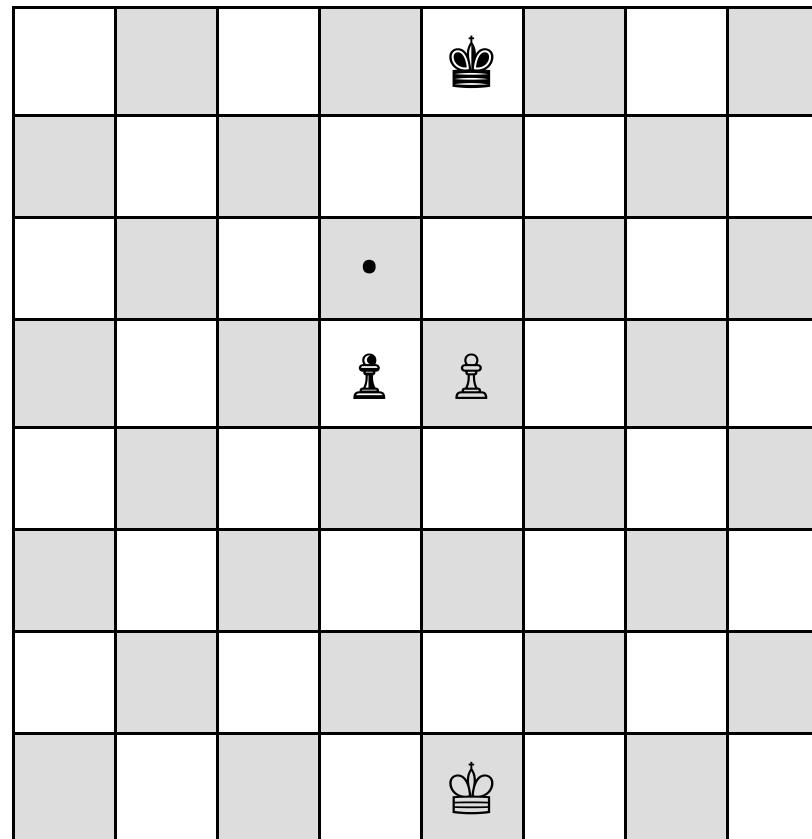
L'algorithme de hachage de Zobrist : exemple des échecs

Prise en passant



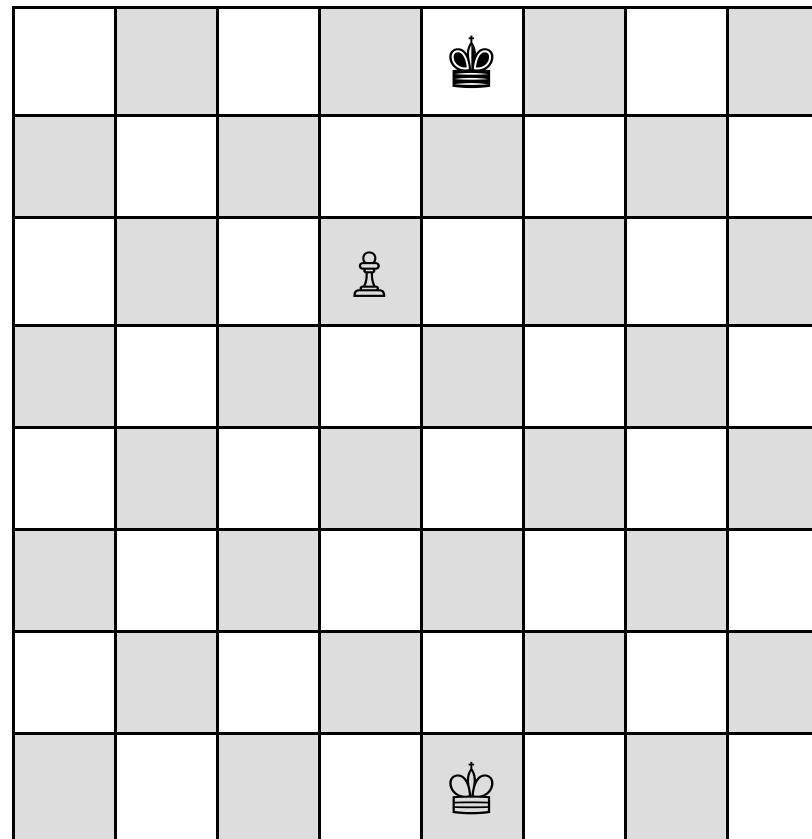
L'algorithme de hachage de Zobrist : exemple des échecs

Prise en passant



L'algorithme de hachage de Zobrist : exemple des échecs

Prise en passant



L'algorithme de hachage de Zobrist : exemple des échecs

- On va générer un tableau de nombres pseudo-aléatoires :
 - 1 pour chaque pièce sur chaque case (♔ ♜ ♕ ♖ ♗ ♘ ♙ ♚ et ♔ ♜ ♕ ♖ ♗ ♘ ♙ ♚)
 - 1 nombre qui précise si c'est aux noirs de jouer
 - 4 nombres représentant les possibilités de roque
 - 8 nombres représentant les prises en passant

Soit $64 \times 6 \times 2 + 1 + 4 + 8 = 781$ nombres aléatoires

- On utilise un `xor` bit à bit pour coder un état du jeu avec l'ensemble des nombres
- Pour mettre à jour l'état du jeu, il suffit d'appliquer un `xor` pour enlever ou pour ajouter une propriété

L'algorithme de hachage de Zobrist : exemple des échecs

Exemples

- État initial :

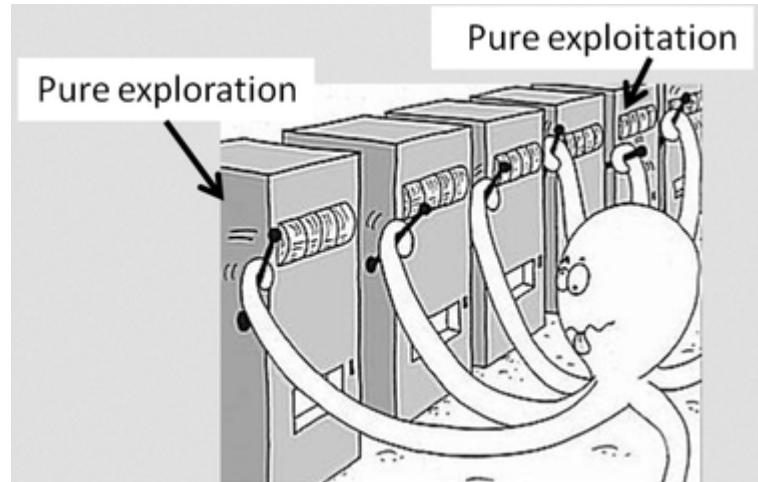
```
[Hash de la tour blanche en a1]
xor [Hash du cavalier blanc en b1]
xor [Hash du fou blanc en c1]
xor ... ( toutes les pièces )
xor [Hash du roi blanc pour le petit roque]
xor [Hash du roi blanc pour le grand roque]
xor ... (toutes les possibilités de roque)
```

- État suivant la prise du fou noir en c3 par le cavalier blanc qui était en b1

```
[Valeur de Hash de la position précédente]
xor [Hash du cavalier blanc en b1] (on supprime le cavalier blanc en b1)
xor [Hash du fou noir en c3] (on supprime le fou noir en c3)
xor [Hash du cavalier blanc en c3] (on place le cavalier blanc en c3)
xor [Hash du tour des noirs] (c'est au noir de jouer)
```

Algorithme de bandits manchots

- **Problème** : soit n machines à sous qui n'ont pas la même probabilité de gain.
Quel bras utiliser ? Comment itérer le processus ?
- Applications en apprentissage, en médecine, en radio et en jeu



- **Exploitation** : aller à son restaurant favori
- **Exploration** : essayer un nouveau restaurant

Algorithme de bandits manchots

Regret...

Le regret après n essais est la différence entre la récompense moyenne de la meilleure machine et l'espérance de récompense après n essais

$$r_n = n \mu^* - \sum_{k=1}^n \mathbb{E}(\mu_{I_k})$$

avec μ^* la récompense moyenne de la meilleure machine et μ_{I_k} la récompense obtenue à l'instant k

UCB (*Upper Confidence Bounds*)

- Proposé par P. Auer en 2002
- La moyenne empirique de chaque bandit j est :

$$X_j = \frac{1}{T_j} \sum_{i=1}^t r_i \chi_{a_j=i}$$

- t le nombre d'essais réalisés
- T_j le nombre d'essais faits sur la machine j
- r_i la récompense obtenue pour l'essai i et χ la fonction indicatrice qui indique si la machine j a été choisie à l'essai i

Algorithme de bandits manchots

- On ajoute un biais à cette moyenne afin que le regret diminue de façon logarithmique :

$$B_j = X_j + A_j$$

tel que :

$$A_j = \sqrt{\frac{2 \ln(t)}{T_j}}$$

- t le nombre d'essais réalisés
- T_j le nombre d'essais faits sur la machine j
- On choisit le bras qui maximise B_j
- X_j est le terme d'exploitation
- A_j est le terme d'exploration

Utilisation dans les jeux

- On considère que l'adversaire joue au hasard
- Chaque coup possible peut être vu comme le bras d'un bandit
- On lance une simulation pour connaître de résultat de (c.a.d. on tire au hasard toutes les actions jusqu'à ce que l'on arrive sur une récompense)
- En recommence un très grand nombre de fois
- On utilise l'action ayant la plus grande espérance

Exemples :

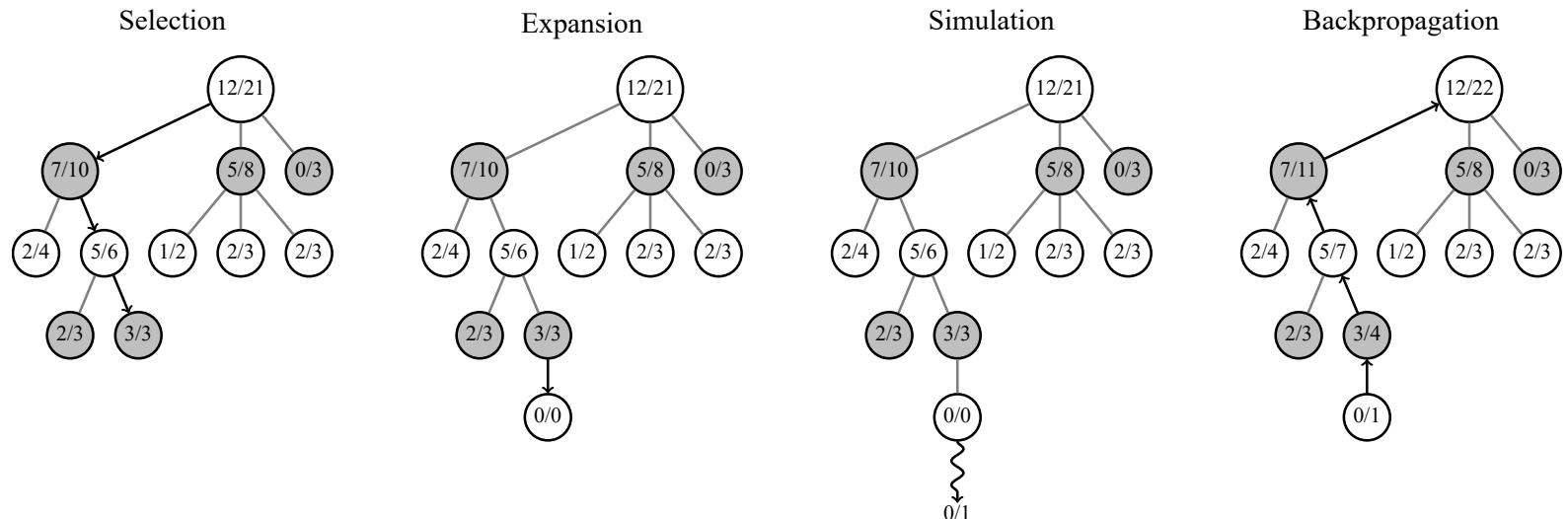
- Jouer au morpion avec un dé
- Jouer à des jeux indéterministes
- Utiliser comme fonction d'évaluation dans des jeux où la combinatoire explose (ex : MCTS)

Monte-Carlo tree search (MCTS)

Présentation

- Utilisé dans MoGo, Alpha Go et Alpha Zéro
- L'idée est de ne développer que les nœuds les plus intéressants
- On évalue l'intérêt des nœuds via des simulations

Les 4 phases



Monte-Carlo tree search (MCTS)

UCT : Upper Confidence Bounds for Trees

- Application de UCB pour MCTS
- UCT = terme d'exploitation + terme d'exploration
- On pour le choix de sélection, on maximize la valeur suivante :

$$UCT = \frac{w_i}{n_i} + C \sqrt{\frac{\ln(N_i)}{n_i}}$$

Avec :

- w_i le nombre de victoire pour le nœud considéré
- n_i nombre de fois où le nœud a été considéré
- N_i le nombre de fois où le nœud père de i a été visité
- C est le paramètre d'exploration (en théorie $\sqrt{2}$ pour une récompense de 1, en pratique adapté au jeu considéré)

Conclusion

Algorithme pour les jeux séquentiels à information complète

- Algorithme min-max
- Élagage α - β

Algorithmes stochastiques

- Algorithme de Zobrist
- UCB et MCTS

D'autres jeux, d'autres outils, d'autres challenges...

- Jeux stochastiques (avec des dés) : PDM
- Jeux à information incomplète : rendre le jeu à information complète ?
- Jeux épistémiques : logiques épistémiques
- Jeux temps réels
- *Le General Game Playing*

À propos de ce document...

Information	Valeur
Auteur	Sylvain Lagrue (sylvain.lagrue@utc.fr)
Licence	Creative Common CC BY-SA 3.0
Version document	1.2.2

Des coquilles ? sylvain.lagrue@utc.fr ou sur le [forum du cours moodle](#)

