

# Apprentissage par renforcement

Grégoire Passault

February 4, 2022

## 1 Introduction

## 2 Formalisme

## 3 Politique optimale, value iteration

## 4 Apprentissage sans modèle

## 5 Monte Carlo

## 6 TD et Q-Learning

# Motivation

La robotique industrielle est gouvernée par des robots:

- Dotés d'encodeurs très précis,
- De réducteurs sans jeu (par exemple harmonique),
- En somme, très fidèles à leur **modèle physique**.



# Motivation

Cependant:

- Il n'est pas toujours facile d'avoir le modèle complet d'un robot (robot à bas coût, jeu, encodeurs imprécis, pièces tordues...)
- Même en connaissant parfaitement le modèle, ça ne résout pas le problème de planification (exploitation) du modèle.

## *Model-based ou Data-based*

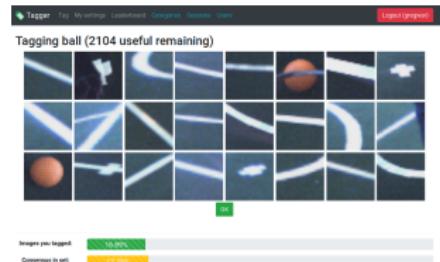
Un robot peut interagir avec son environnement. On aimerait pouvoir exploiter ces interactions pour améliorer son comportement



# Catégories d'apprentissage automatique

Il existe plusieurs catégories d'apprentissage:

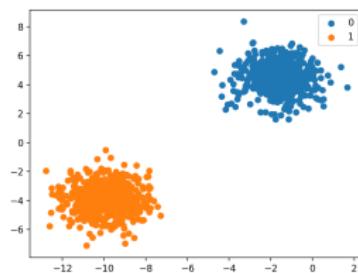
- **Apprentissage supervisé:** On apprend sur des données étiquetées avec une vérité terrain
- **Apprentissage non supervisé:** On dispose uniquement des données, et on essaie d'en comprendre des caractéristiques
- **Apprentissage par renforcement:** On veut maximiser un signal de futures récompenses (nous allons en reparler!)



# Catégories d'apprentissage automatique

Il existe plusieurs catégories d'apprentissage:

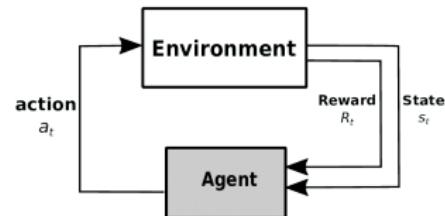
- **Apprentissage supervisé:** On apprend sur des données étiquetées avec une vérité terrain
- **Apprentissage non supervisé:** On dispose uniquement des données, et on essaie d'en comprendre des caractéristiques
- **Apprentissage par renforcement:** On veut maximiser un signal de futures récompenses (nous allons en reparler!)



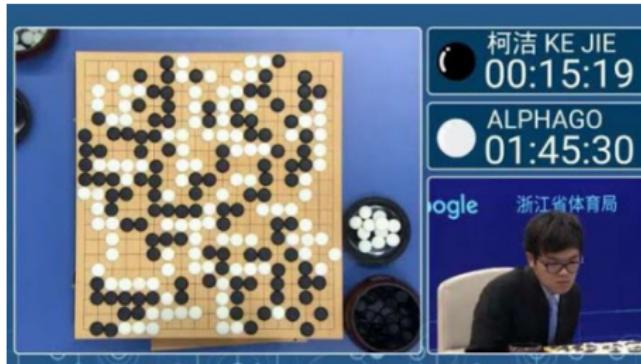
# Catégories d'apprentissage automatique

Il existe plusieurs catégories d'apprentissage:

- **Apprentissage supervisé:** On apprend sur des données étiquetées avec une vérité terrain
- **Apprentissage non supervisé:** On dispose uniquement des données, et on essaie d'en comprendre des caractéristiques
- **Apprentissage par renforcement:** On veut maximiser un signal de futures récompenses (nous allons en reparler!)



# Success story

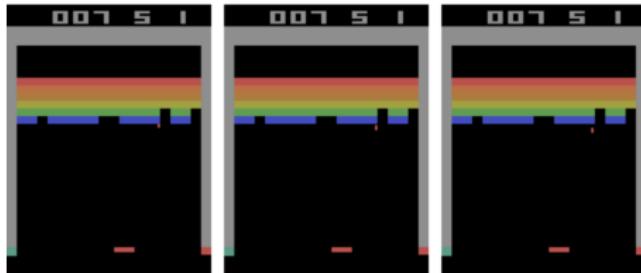


## AlphaGo, puis AlphaGo zero (DeepMind)

Capable de battre le grand maître humain au jeu de Go en utilisant une base de données de parties pour l'entraînement.

Variante "Zero", capable d'apprendre en jouant uniquement contre elle-même.

# Success story



## Apprentissage automatique de jeux d'Atari (DeepMind)

En utilisant l'apprentissage par renforcement (mariée au deep learning), le même algorithme était capable d'apprendre à jouer à une multitude de jeux, à partir de l'image, la manette et le score.

# Success story



## Victoire contre des grand maître StarCraft (DeepMind)

Dans un des jeux les plus compétitifs du monde, l'IA de Google s'est élevée au rang de grand maître.

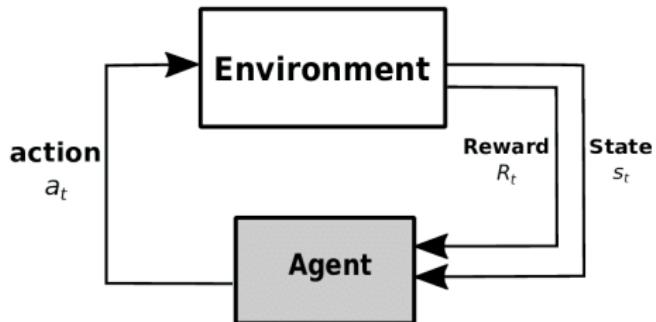
# Success story



## Résolution de Rubik's Cube avec une main robotique (OpenAI)

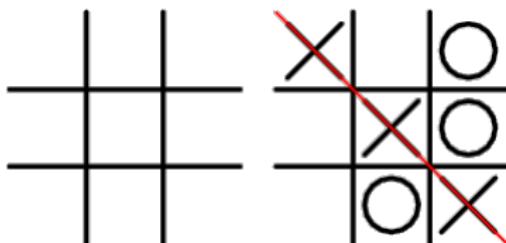
La manipulation du cube est apprise par de l'apprentissage par renforcement, à partir d'abord de simulation, puis sur le vrai robot.

# Formalisme



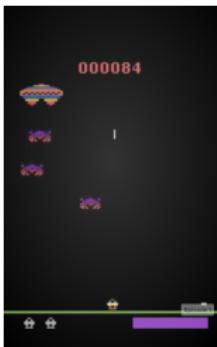
L'apprenant (qu'on appelle agent) est plongé dans un environnement avec lequel il interagit. À chaque étape, l'agent est dans un état  $s_t$ , il décide alors de son action  $a_t$  et observe une récompense  $r_{t+1}$  et un nouvel état  $s_{t+1}$ .

## Exemple: morpion



- La récompense est latente (gagné / perdu), jouer un coup change uniquement d'état
- Le jeu est déterministe, on peut explorer tous les cas, mais que se passera t-il sur un jeu de plateau plus grand? Sur un morpion 4x4, 5x5 par exemple?

## Exemple: jouer à un jeu vidéo



- Que représentent sémantiquement les éléments à l'écran ?
- Quel effet ont les boutons de la manette sur le jeu ?
- Anticiper les états futurs dans lesquels on souhaite aller ou éviter
- Certains jeux sont stochastiques (ex: jeter un dé)

## Exemple: piloter un Kart



- Difficile de connaître tous les paramètres physiques
- On peut modéliser le système par de la physique... mais aussi utiliser des expériences réelles à la place

# Motivation

Une des motivations de l'apprentissage par renforcement est la capacité d'apprendre en **interaction avec l'environnement**, et aussi le mélange de l'apprentissage et des actions visant à mieux **découvrir l'environnement** (le compromis exploration/exploitation dont nous avons parlé).

Les problèmes seront décrit selon un certain formalisme (que nous allons détailler).

1 Introduction

2 Formalisme

3 Politique optimale, value iteration

4 Apprentissage sans modèle

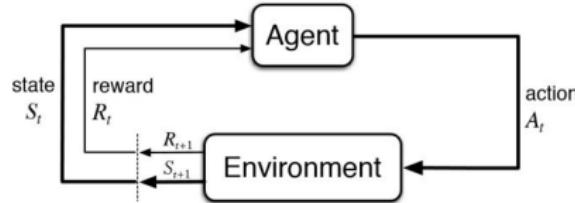
5 Monte Carlo

6 TD et Q-Learning

# Formalisme: MDP

On note:

- $S$ , un ensemble d'états,
- $A$ , un ensemble d'actions,
- $P(s'|s, a)$ , le "modèle" de l'environnement: la probabilité d'arriver dans l'état  $s'$  après avoir effectué l'action  $a$  depuis l'état  $s$  (il peut être connu ou inconnu)
- $R_{s,s'}^a$ , une fonction de récompense: obtenue en appliquant l'action  $a$  depuis l'état  $s$  et d'être arrivé dans l'état  $s'$ .



# Formalisme: catégories

Il existe des catégories de problèmes:

- **Avec / Sans terminaison**

Est-ce que l'environnement a une "fin" ou est-ce qu'il peut se poursuivre à l'infini ? (Y-a t-il des états "terminaux")

- **Totalement observable / Partiellement observables**

Notre espace d'états  $S$  suffit t-il à prendre une décision ?

- **Modèle connu / Inconnu**

Est-ce qu'on connaît le modèle de transition ? Si je suis dans un état  $s$  et que j'applique l'action  $a$ , est-ce que je peux prédire (même avec des probabilités) dans quel état j'arriverai ?

# Formalisme: catégories

Il existe des catégories de problèmes:

- **Avec / Sans terminaison**

Est-ce que l'environnement a une "fin" ou est-ce qu'il peut se poursuivre à l'infini ? (Y-a t-il des états "terminaux")

- **Totalement observable / Partiellement observables**

Notre espace d'états  $S$  suffit t-il à prendre une décision ?

- **Modèle connu / Inconnu**

Est-ce qu'on connaît le modèle de transition ? Si je suis dans un état  $s$  et que j'applique l'action  $a$ , est-ce que je peux prédire (même avec des probabilités) dans quel état j'arriverai ?

# Formalisme: catégories

Il existe des catégories de problèmes:

- **Avec / Sans terminaison**

Est-ce que l'environnement a une "fin" ou est-ce qu'il peut se poursuivre à l'infini ? (Y-a t-il des états "terminaux")

- **Totalement observable / Partiellement observables**

Notre espace d'états  $S$  suffit t-il à prendre une décision ?

- **Modèle connu / Inconnu**

Est-ce qu'on connaît le modèle de transition ? Si je suis dans un état  $s$  et que j'applique l'action  $a$ , est-ce que je peux prédire (même avec des probabilités) dans quel état j'arriverai ?

# Formalisme: catégories (2)

- **Déterministe / Stochastique**

Dans le cas déterministe,  $P(s'|s, a)$  vaut 1 pour un seul état cible (le modèle peut se résumer à une fonction  $s' = m(s, a)$ )

- **Discret / Continu**

Est-ce que  $S$  et  $A$  sont discrets ou continus ? (Ou un mélange des deux...)

- **En ligne / Hors ligne**

Est-ce qu'on apprend sur des expériences faites par l'agent ou par quelqu'un d'autre ? (*On-policy / Off-policy*)

# Formalisme: catégories (2)

- Déterministe / Stochastique

Dans le cas déterministe,  $P(s'|s, a)$  vaut 1 pour un seul état cible (le modèle peut se résumer à une fonction  $s' = m(s, a)$ )

- Discret / Continu

Est-ce que  $S$  et  $A$  sont discrets ou continus ? (Ou un mélange des deux...)

- En ligne / Hors ligne

Est-ce qu'on apprend sur des expériences faites par l'agent ou par quelqu'un d'autre ? (*On-policy / Off-policy*)

# Formalisme: catégories (2)

- Déterministe / Stochastique

Dans le cas déterministe,  $P(s'|s, a)$  vaut 1 pour un seul état cible (le modèle peut se résumer à une fonction  $s' = m(s, a)$ )

- Discret / Continu

Est-ce que  $S$  et  $A$  sont discrets ou continus ? (Ou un mélange des deux...)

- En ligne / Hors ligne

Est-ce qu'on apprend sur des expériences faites par l'agent ou par quelqu'un d'autre ? (*On-policy / Off-policy*)

## Formalisme: politique

On appelle politique  $\pi$  (*policy* en anglais) la "stratégie" de notre agent, qui prend un état  $s$  et fournit l'action à effectuer  $\pi(s)$ .

Notre but est de produire une politique qui donne le meilleur résultat. Pour formaliser ce résultat, on veut avoir la meilleure série de récompenses obtenues:

$$J(\pi) = \mathbb{E}_{s \sim E} \left[ \sum_{t=1}^{\infty} R_{s_t, s_{t+1}}^{\pi(s_t)} \right]$$

Où:

- $a_t = \pi(s_t)$  est la décision que l'on prend dans l'état  $s_t$ ,
- $s_{t+1}$  est l'état dans lequel on arrive après avoir pris l'action  $a_t$  dans l'état  $s_t$ .

## Formalisme: politique

On appelle politique  $\pi$  (*policy* en anglais) la "stratégie" de notre agent, qui prend un état  $s$  et fournit l'action à effectuer  $\pi(s)$ .

Notre but est de produire une politique qui donne le meilleur résultat. Pour formaliser ce résultat, on veut avoir la meilleure série de récompenses obtenues:

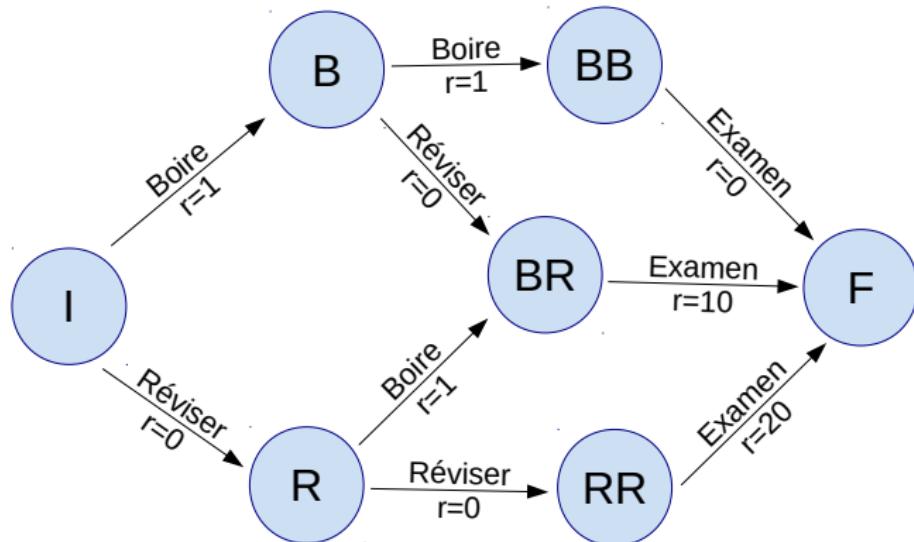
$$J(\pi) = \mathbb{E}_{s \sim E} \left[ \sum_{t=1}^{\infty} R_{s_t, s_{t+1}}^{\pi(s_t)} \right]$$

Où:

- $a_t = \pi(s_t)$  est la décision que l'on prend dans l'état  $s_t$ ,
- $s_{t+1}$  est l'état dans lequel on arrive après avoir pris l'action  $a_t$  dans l'état  $s_t$ .

# Un exemple

$$S = I, B, BB, R, RR, BR, F, A = \text{Boire}, \text{Reviser}, \text{Examen}$$



## Fonction de valeur

On appelle fonction de valeur la fonction  $V^\pi$  qui représente toutes les récompenses futures à partir d'un état  $s$  en suivant une politique  $\pi$  donnée:

$$V^\pi(s) = \mathbb{E}_{s \sim E} \left[ \sum_{t=1}^{\infty} R_{s_t, s_{t+1}}^{\pi(s_t)} | s_1 = s \right]$$

Où

- $\pi(s_t)$  est l'action sélectionnée par la politique à l'étape  $t$
- $s_{t+1}$  est l'état dans lequel on arrive après avoir sélectionné l'action  $a_t$  dans l'état  $s_t$ .

## Fonction de valeur (2)

On peut remarquer que la fonction de valeur est récursive (on l'appelle l'équation de Bellman):

$$V^\pi(s) = \mathbb{E}_{s' \sim E}[R_{s,s'}^{\pi(s_t)} + V^\pi(s')]$$

$$V^\pi(s) = \sum_{s'} P(s'|s, a)[R_{s,s'}^a + V^\pi(s')]$$

Avec  $a = \pi(s)$  choisi à l'aide de la politique

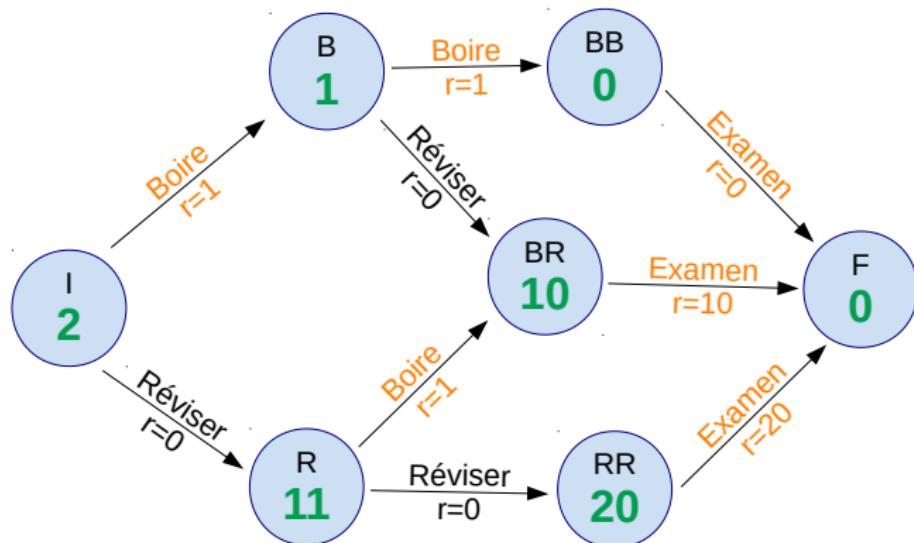
- Problème: on ne donne aucune importance à la récompense court terme
- Problème: dans un épisode sans terminaison, cette valeur peut tendre vers l'infini

Solution: introduire un terme d'escompte  $\gamma \in [0, 1]$ :

$$V^\pi(s) = \sum_{s'} P(s'|s, a)[R_{s,s'}^a + \gamma V^\pi(s')]$$

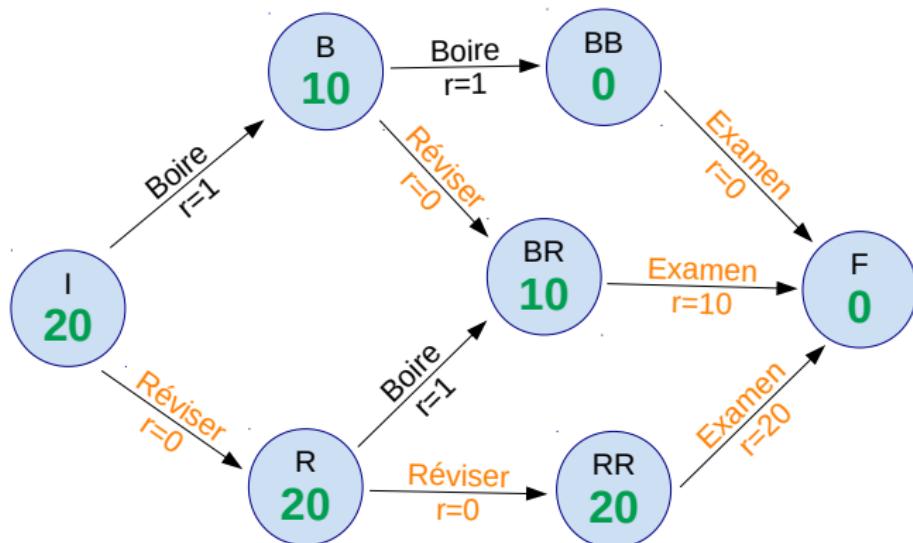
# Un exemple

Si  $\gamma = 1$ , et que la politique suit les actions en orange, la fonction de valeur est:



# Un exemple

Pour une autre politique, la fonction de valeur est différente!



1 Introduction

2 Formalisme

3 Politique optimale, value iteration

4 Apprentissage sans modèle

5 Monte Carlo

6 TD et Q-Learning

# Propriété de Markov

## Propriété de Markov

*"Given the present, the future does not depend on the past"*

$$P(S_{t+1}|S_t, S_{t-1}, \dots S_1) = P(S_{t+1}|S_t)$$

Quelles que soient les actions passées, l'état actuel représente totalement la situation.

# Politique optimale

Une politique  $(\pi')$  est meilleure qu'une autre  $(\pi)$  si  $\forall s, V^{\pi'}(s) \geq V^\pi(s)$ .

Il existe une politique optimale  $\pi^*$ , qui est meilleure que toutes les autres  
 $\forall \pi, \forall s, [V^{\pi^*}(s) \geq V^\pi(s)]$

- Notons  $V^*$  la fonction de valeur de cette politique optimale.
- Dans ce cas là, l'action optimale est celle qui maximise  $\sum_{s'} P(s'|s, a)[R_{s,s'}^a + V^*(s')]$ . Si on connaît  $V^*$ , on peut donc trouver  $\pi^*$ .
- À condition de connaître le modèle! (Et de pouvoir "scanner" les actions pour faire la maximisation).

# Politique optimale

Une politique  $(\pi')$  est meilleure qu'une autre  $(\pi)$  si  $\forall s, V^{\pi'}(s) \geq V^\pi(s)$ .

Il existe une politique optimale  $\pi^*$ , qui est meilleure que toutes les autres  
 $\forall \pi, \forall s, [V^{\pi^*}(s) \geq V^\pi(s)]$

- Notons  $V^*$  la fonction de valeur de cette politique optimale.
- Dans ce cas là, l'action optimale est celle qui maximise  $\sum_{s'} P(s'|s, a)[R_{s,s'}^a + V^*(s')]$ . Si on connaît  $V^*$ , on peut donc trouver  $\pi^*$ .
- À condition de connaître le modèle! (Et de pouvoir "scanner" les actions pour faire la maximisation).

# Politique optimale

Une politique  $(\pi')$  est meilleure qu'une autre  $(\pi)$  si  $\forall s, V^{\pi'}(s) \geq V^\pi(s)$ .

Il existe une politique optimale  $\pi^*$ , qui est meilleure que toutes les autres  
 $\forall \pi, \forall s, [V^{\pi^*}(s) \geq V^\pi(s)]$

- Notons  $V^*$  la fonction de valeur de cette politique optimale.
- Dans ce cas là, l'action optimale est celle qui maximise  $\sum_{s'} P(s'|s, a)[R_{s,s'}^a + V^*(s')]$ . Si on connaît  $V^*$ , on peut donc trouver  $\pi^*$ .
- À condition de connaître le modèle! (Et de pouvoir "scanner" les actions pour faire la maximisation).

# Politique optimale

Une politique  $(\pi')$  est meilleure qu'une autre  $(\pi)$  si  $\forall s, V^{\pi'}(s) \geq V^\pi(s)$ .

Il existe une politique optimale  $\pi^*$ , qui est meilleure que toutes les autres  
 $\forall \pi, \forall s, [V^{\pi^*}(s) \geq V^\pi(s)]$

- Notons  $V^*$  la fonction de valeur de cette politique optimale.
- Dans ce cas là, l'action optimale est celle qui maximise  $\sum_{s'} P(s'|s, a)[R_{s,s'}^a + V^*(s')]$ . Si on connaît  $V^*$ , on peut donc trouver  $\pi^*$ .
- À condition de connaître le modèle! (Et de pouvoir "scanner" les actions pour faire la maximisation).

# Politique optimale

Une politique  $(\pi')$  est meilleure qu'une autre  $(\pi)$  si  $\forall s, V^{\pi'}(s) \geq V^\pi(s)$ .

Il existe une politique optimale  $\pi^*$ , qui est meilleure que toutes les autres  
 $\forall \pi, s, [V^{\pi^*}(s) \geq V^\pi(s)]$

- Notons  $V^*$  la fonction de valeur de cette politique optimale.
- Dans ce cas là, l'action optimale est celle qui maximise  $\sum_{s'} P(s'|s, a)[R_{s,s'}^a + V^*(s')]$ . Si on connaît  $V^*$ , on peut donc trouver  $\pi^*$ .
- À condition de connaître le modèle! (Et de pouvoir "scanner" les actions pour faire la maximisation).

# Politique gourmande

Pour une fonction de valeur donnée  $V$ , on peut définir la politique **gourmande**  $\hat{\pi}$  qui consiste à prendre l'action  $a$  qui maximise  $\sum_{s'} P(s'|s, a)[R_{s,s'}^a + V(s')]$ .

$$\hat{\pi}^V(s) = \arg \max_a \left[ \sum_{s'} P(s'|s, a) [R_{s,s'}^a + V(s')] \right]$$

# Value iteration

- On prend  $V_0$ , en initialisant les valeurs à 0 par exemple, on peut donc en dériver une politique gourmande  $\hat{\pi}^{V_0}$
- On peut alors calculer  $V^{\hat{\pi}^{V_0}}$ , la fonction de valeur de  $\hat{\pi}^{V_0}$   
(souvenez-vous, la fonction de valeur dépend d'une politique donnée!)
- Ainsi de suite, on peut continuer d'alterner entre construction de la politique et évaluation, jusqu'à ce que cette étape de mise à jour n'ait plus aucun effet

On appelle cet algorithme (ou presque) la **value iteration**

## Value iteration

- On prend  $V_0$ , en initialisant les valeurs à 0 par exemple, on peut donc en dériver une politique gourmande  $\hat{\pi}^{V_0}$
- On peut alors calculer  $V^{\hat{\pi}^{V_0}}$ , la fonction de valeur de  $\hat{\pi}^{V_0}$   
(souvenez-vous, la fonction de valeur dépend d'une politique donnée!)
- Ainsi de suite, on peut continuer d'alterner entre construction de la politique et évaluation, jusqu'à ce que cette étape de mise à jour n'ait plus aucun effet

On appelle cet algorithme (ou presque) la **value iteration**

## Value iteration

- On prend  $V_0$ , en initialisant les valeurs à 0 par exemple, on peut donc en dériver une politique gourmande  $\hat{\pi}^{V_0}$
- On peut alors calculer  $V^{\hat{\pi}^{V_0}}$ , la fonction de valeur de  $\hat{\pi}^{V_0}$   
(souvenez-vous, la fonction de valeur dépend d'une politique donnée!)
- Ainsi de suite, on peut continuer d'alterner entre construction de la politique et évaluation, jusqu'à ce que cette étape de mise à jour n'ait plus aucun effet

On appelle cet algorithme (ou presque) la **value iteration**

# Value iteration

- On prend  $V_0$ , en initialisant les valeurs à 0 par exemple, on peut donc en dériver une politique gourmande  $\hat{\pi}^{V_0}$
- On peut alors calculer  $V^{\hat{\pi}^{V_0}}$ , la fonction de valeur de  $\hat{\pi}^{V_0}$   
(souvenez-vous, la fonction de valeur dépend d'une politique donnée!)
- Ainsi de suite, on peut continuer d'alterner entre construction de la politique et évaluation, jusqu'à ce que cette étape de mise à jour n'ait plus aucun effet

On appelle cet algorithme (ou presque) la **value iteration**

## Value iteration: algorithme

Initialize array  $V$  arbitrarily (e.g.,  $V(s) = 0$  for all  $s \in \mathcal{S}^+$ )

Repeat

$$\Delta \leftarrow 0$$

For each  $s \in \mathcal{S}$ :

$$v \leftarrow V(s)$$

$$V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$$

$$\Delta \leftarrow \max(\Delta, |v - V(s)|)$$

until  $\Delta < \theta$  (a small positive number)

Output a deterministic policy,  $\pi$ , such that

$$\pi(s) = \arg \max_a \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$$

# Value iteration

Pourquoi ça marche (intuition) ?

- La politique gourmande de la fonction de valeur optimale est la politique optimale
- La fonction de valeur de la politique optimale est la fonction de valeur optimale

Donc  $\hat{\pi}^{V^{\pi^*}} = \pi^*$ . Autrement dit  $\pi^*$  est un "point fixe" par l'opération  $f(\pi) = \hat{\pi}^{V^\pi}$ .

# Value iteration

Pourquoi ça marche (intuition) ?

- La politique gourmande de la fonction de valeur optimale est la politique optimale
- La fonction de valeur de la politique optimale est la fonction de valeur optimale

Donc  $\hat{\pi}^{V^{\pi^*}} = \pi^*$ . Autrement dit  $\pi^*$  est un "point fixe" par l'opération  $f(\pi) = \hat{\pi}^{V^\pi}$ .

# Value iteration

Pourquoi ça marche (intuition) ?

- La politique gourmande de la fonction de valeur optimale est la politique optimale
- La fonction de valeur de la politique optimale est la fonction de valeur optimale

Donc  $\hat{\pi}^{V^{\pi^*}} = \pi^*$ . Autrement dit  $\pi^*$  est un "point fixe" par l'opération  $f(\pi) = \hat{\pi}^{V^\pi}$ .

# Value iteration

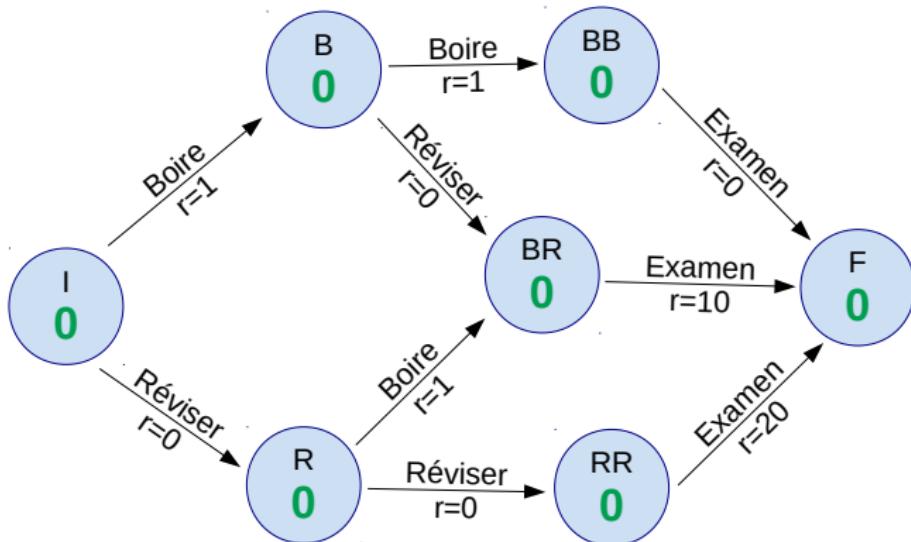
Pourquoi ça marche (intuition) ?

- La politique gourmande de la fonction de valeur optimale est la politique optimale
- La fonction de valeur de la politique optimale est la fonction de valeur optimale

Donc  $\hat{\pi}^{V^{\pi^*}} = \pi^*$ . Autrement dit  $\pi^*$  est un "point fixe" par l'opération  $f(\pi) = \hat{\pi}^{V^\pi}$ .

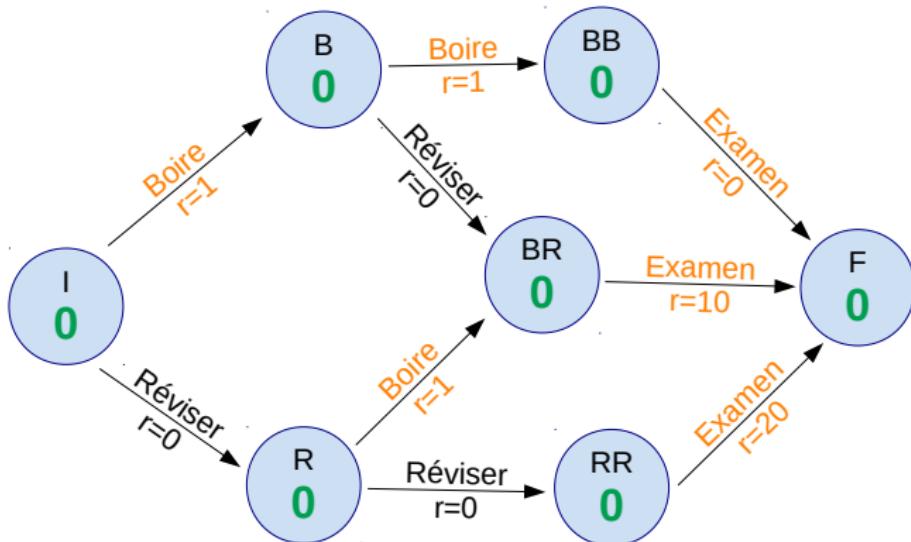
# Value iteration appliquée

Première étape, on assigne la valeur de 0 à tous les états:



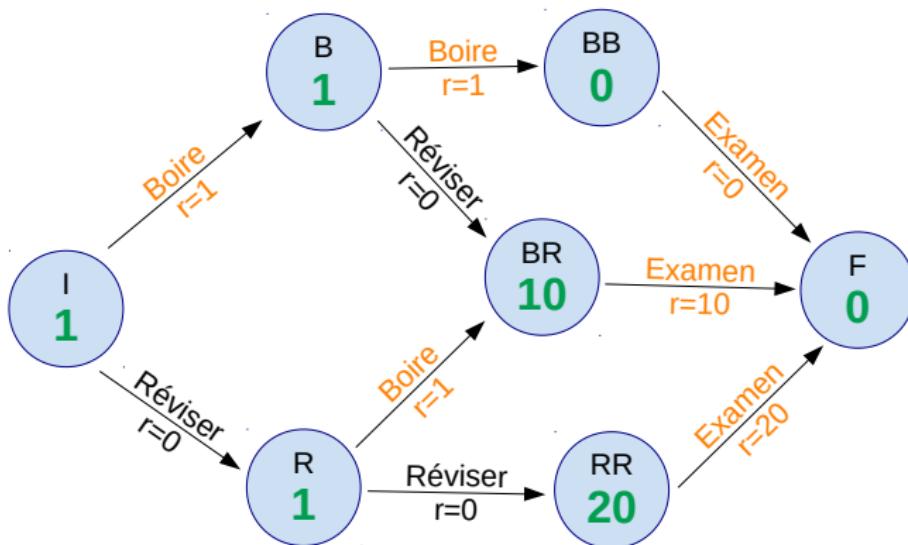
# Value iteration appliquée

Ensuite, on calcule la politique gourmande pour ces valeurs:



# Value iteration appliquée

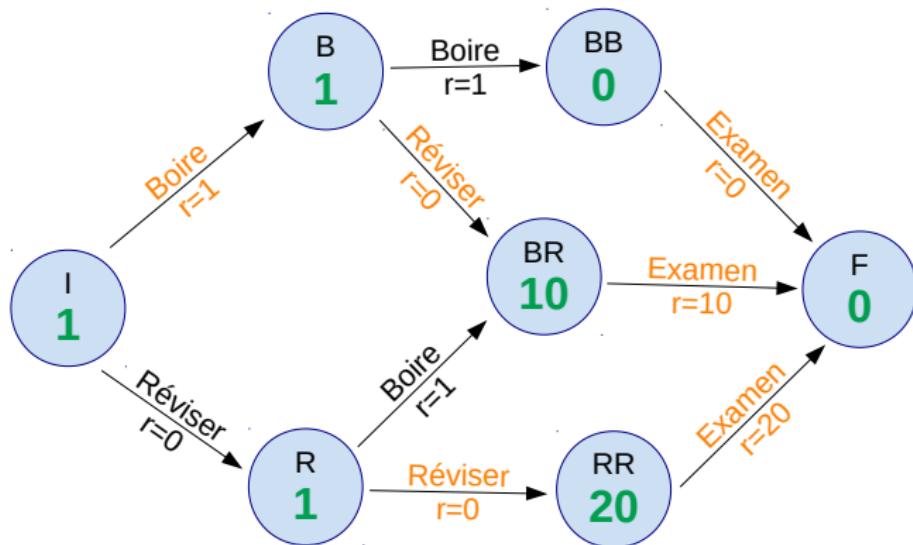
Puis on met à jour la fonction de valeur pour cette politique gourmande:



Les états sont mis à jour de "gauche à droite"

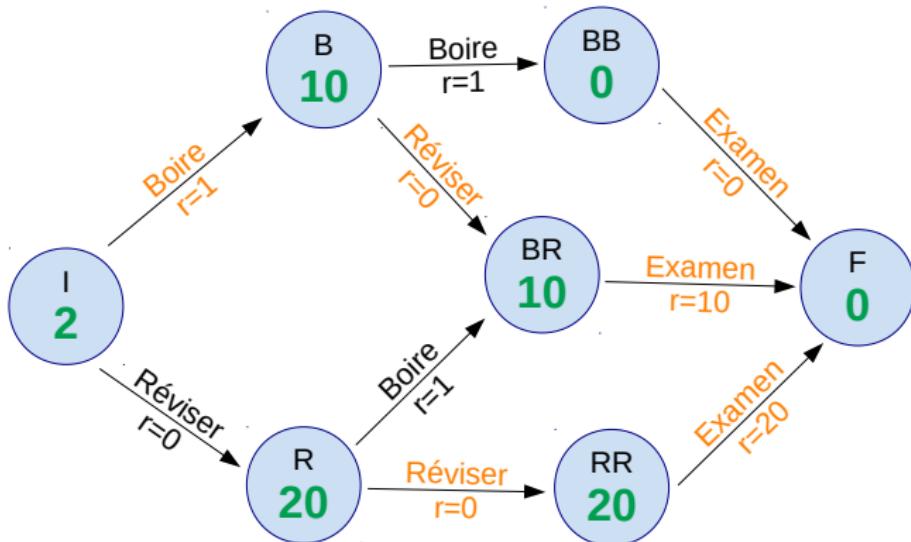
# Value iteration appliquée

On met à jour la politique gourmande pour cette nouvelle fonction de valeur:



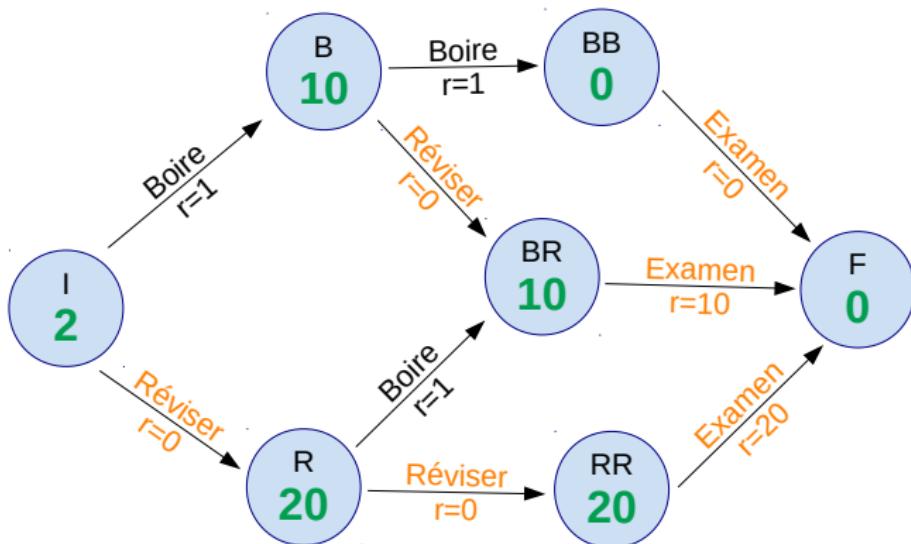
# Value iteration appliquée

Et on calcule à nouveau la fonction de valeur en suivant cette politique:



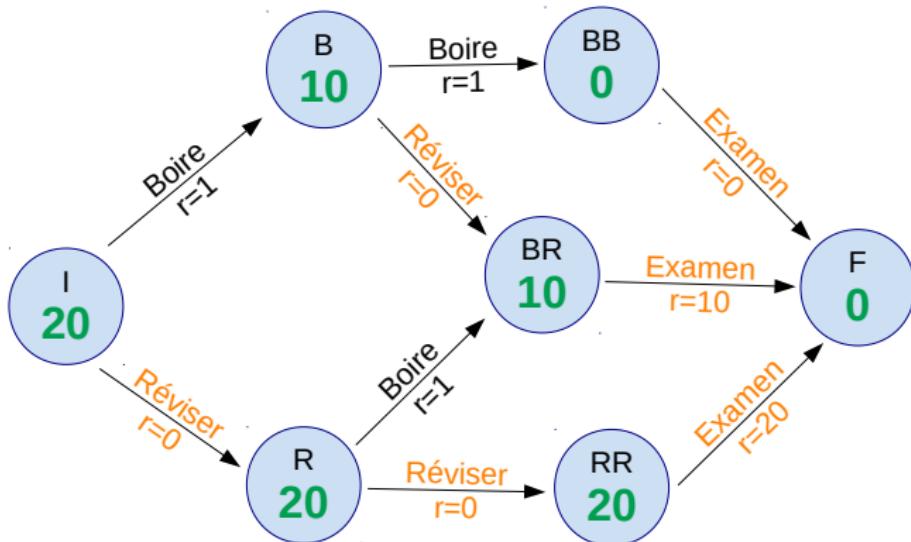
# Value iteration appliquée

Et on obtient la politique suivante:



# Value iteration appliquée

Enfin, la valuation nous donnera le résultat suivant: nous avons convergé!



## Value iteration: remarques

- La politique gourmande n'est pas construire explicitement (sauf à la fin) dans l'algorithme
- L'ordre de mise à jour des états a une importance (ici on les met à jour de gauche à droite): est-ce qu'un autre ordre serait mieux ?
- Une fois l'algorithme terminé, on a alors une politique qui nous donne l'action optimale pour chaque état!

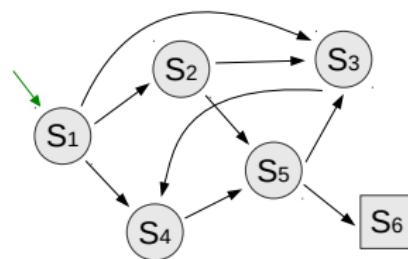
## Value iteration: remarques

- La politique gourmande n'est pas construire explicitement (sauf à la fin) dans l'algorithme
- L'ordre de mise à jour des états a une importance (ici on les met à jour de gauche à droite): est-ce qu'un autre ordre serait mieux ?
- Une fois l'algorithme terminé, on a alors une politique qui nous donne l'action optimale pour chaque état!

## Value iteration: remarques

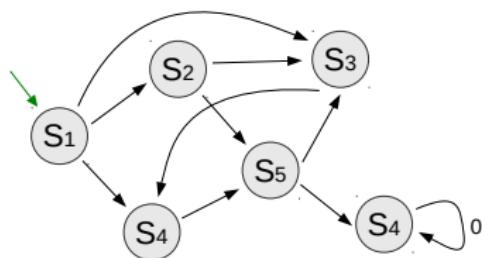
- La politique gourmande n'est pas construire explicitement (sauf à la fin) dans l'algorithme
- L'ordre de mise à jour des états a une importance (ici on les met à jour de gauche à droite): est-ce qu'un autre ordre serait mieux ?
- Une fois l'algorithme terminé, on a alors une politique qui nous donne l'action optimale pour chaque état!

## État initial et final



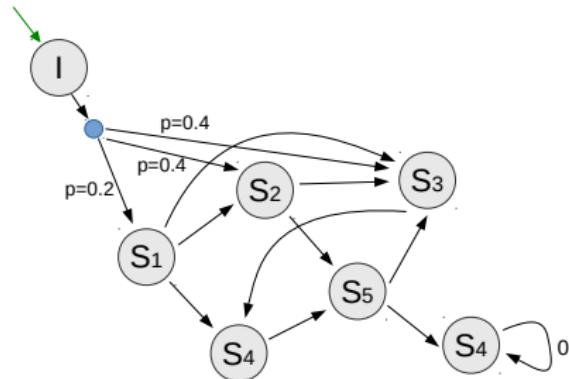
- Il est possible d'écrire sous forme continue un environnement avec états terminaux
- L'état initial n'a pas d'effet sur la politique optimale

## État initial et final



- Il est possible d'écrire sous forme continue un environnement avec états terminaux
- L'état initial n'a pas d'effet sur la politique optimale

## État initial et final



- Il est possible d'écrire sous forme continue un environnement avec états terminaux
- L'état initial n'a pas d'effet sur la politique optimale

# Cas du continu

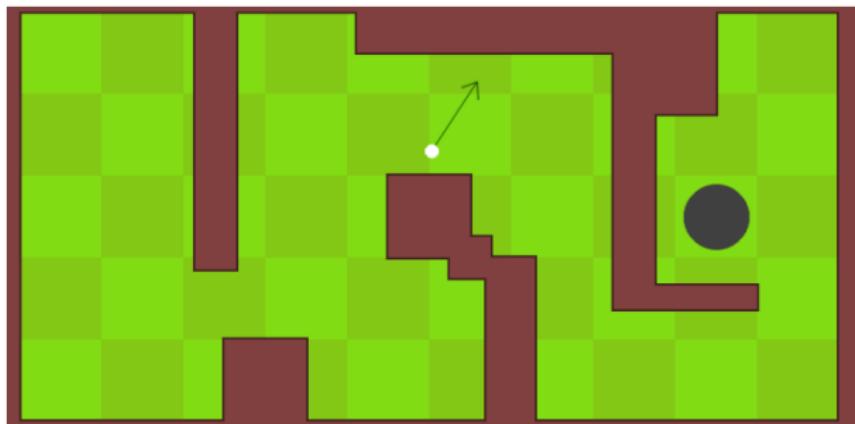
Nous avons dit qu'un espace d'action ou d'états pouvait être **discret** ou **continu**.

Comment traiter les cas continus ? Deux solutions:

- **Discrétiliser** le problème: simple et efficace, mais peut vite exploser!
- Utiliser des **approximateurs de fonctions** (comme des réseaux de neurones): **hors du périmètre de ce cours**

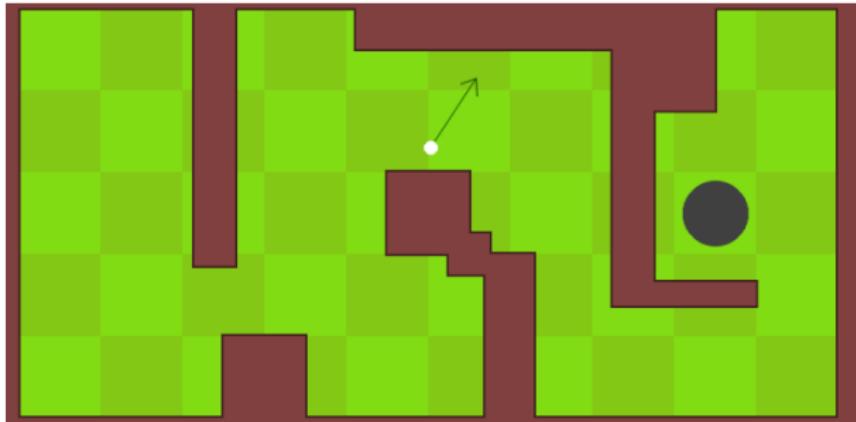
# Exemple mini-golf

On joue au mini-golf, sur un terrain de 10m x 5m:



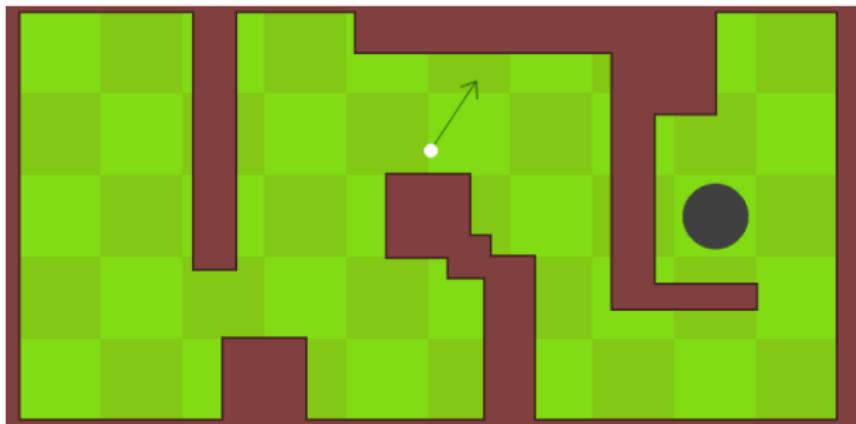
Quel est l'espace d'état ? L'espace d'action ?

# Exemple mini-golf



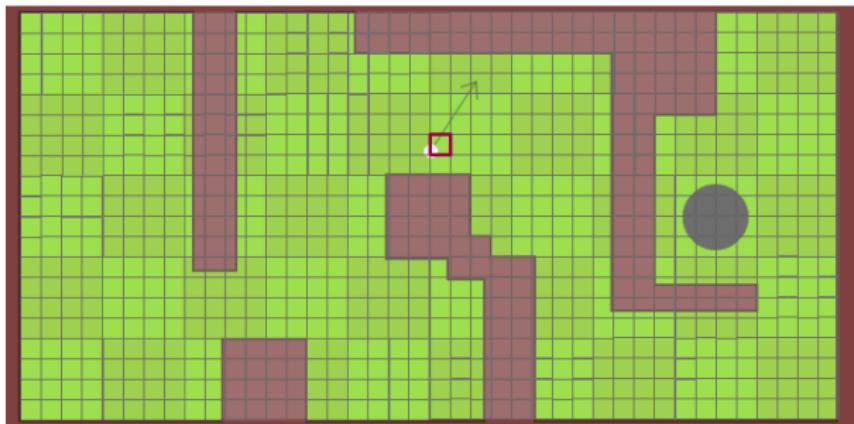
- État: position de la balle
- Action: orientation du tir
- Récompense: ?

## Exemple mini-golf



- État: position de la balle
- Action: orientation du tir
- Récompense: -1 pour chaque tir, l'épisode se termine quand on rentre la balle dans le trou (la récompense est un "coût")

## Exemple mini-golf



On peut choisir un "pas" de discrétisation (par exemple 0.25 m), et considérer que la balle est dans une "case". Pour l'orientation des tirs, on choisit un pas sur les angles.

$$\frac{10}{0.25} \times \frac{5}{0.25} = 800 \text{ états}$$

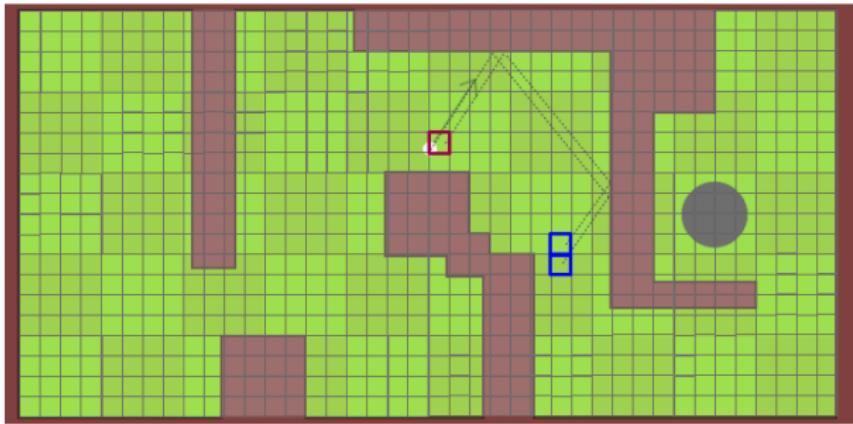
# Problème d'aliasing



Attention: le fait de discréteriser regroupe plusieurs états en un!

Selon la position réelle de la balle dans la case d'origine, la case d'arrivée ne sera pas la même. Le problème continu **déterministe** est devenu **stochastique car partiellement observable**.

# Environnement stochastique



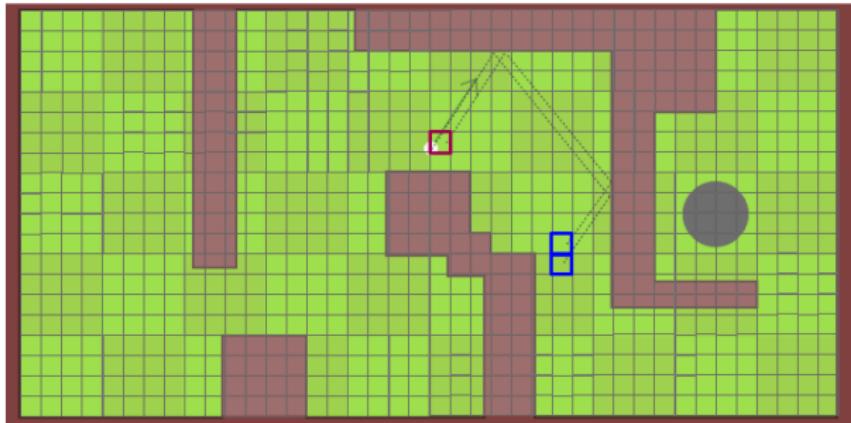
Si le modèle  $m$  était déterministe:

$$V(s) = \max_a [R_{s,m(s,a)}^a + \gamma V(m(s,a))]$$

On utilise alors une répartition de probabilité:

$$V(s) = \max_a \left[ \sum_{s' \in S} P(s'|s, a) [R_{s,s'}^a + \gamma V(s')] \right]$$

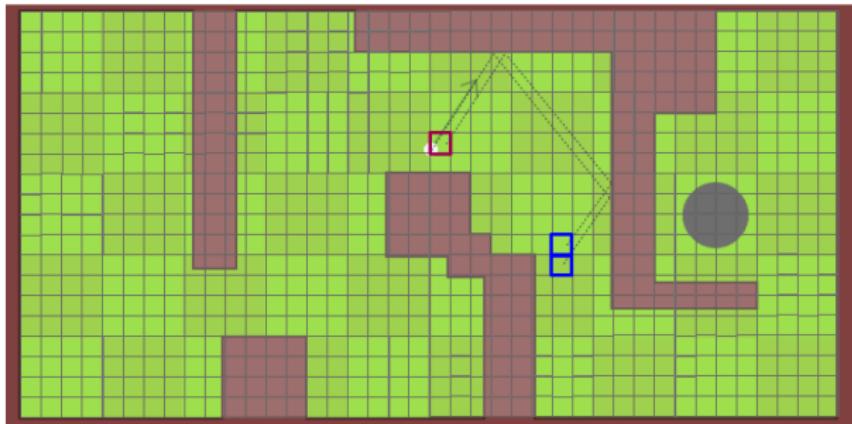
# Environnement stochastique



Comment calculer  $P(s'|s, a)$  ?

- On peut avoir des formules closes (ex: lancé de dés)
- Ou alors échantillonner!

# Environnement stochastique



Comment calculer  $P(s'|s, a)$  ?

- On peut avoir des formules closes (ex: lancé de dés)
- Ou alors échantillonner!

# Environnement stochastique



Comment calculer  $P(s'|s, a)$  ?

- On peut avoir des formules closes (ex: lancé de dés)
- Ou alors échantillonner!

1 Introduction

2 Formalisme

3 Politique optimale, value iteration

4 Apprentissage sans modèle

5 Monte Carlo

6 TD et Q-Learning

## Value iteration: limitations

- Que faire si on ne connaît pas  $P(s'|s, a)$  ?
  - On ne sait pas dans quel état on arrivera si on applique l'action  $a$  à partir de l'état  $s$
  - Par exemple, si vous apprenez à conduire un vrai Kart, ou jouez à un jeu vidéo
- Idée: Sur un problème réel, on aurait plutôt envie de réaliser des expériences, c'est à dire des trajectoires d'états/actions/récompense:  $(s_1, a_1, r_1, s_2, a_2, r_2, \dots, s_n, a_n, r_n, s_{n+1})$  et d'utiliser ces données pour apprendre!

## Value iteration: limitations

- Que faire si on ne connaît pas  $P(s'|s, a)$  ?
  - On ne sait pas dans quel état on arrivera si on applique l'action  $a$  à partir de l'état  $s$
  - Par exemple, si vous apprenez à conduire un vrai Kart, ou jouez à un jeu vidéo
- **Idée:** Sur un problème réel, on aurait plutôt envie de réaliser des expériences, c'est à dire des trajectoires d'états/actions/récompense:  $(s_1, a_1, r_1, s_2, a_2, r_2, \dots, s_n, a_n, r_n, s_{n+1})$  et d'utiliser ces données pour apprendre!

# Fonction de valeur état/action

- Si on ne connaît pas le modèle, même en connaissant  $V^*$ , on ne peut pas trouver la politique optimale, car la politique gourmande cherche à maximiser  $\sum_{s' \in S} P(s', r|s, a)[r + \gamma V(s')]$
- **Solution:** apprendre plutôt une fonction qui prend en paramètre  $s$  l'état actuel et  $a$  la prochaine action et vaut  $Q^\pi(s, a) = R_{s,s'}^a + \gamma V(s')$  (toutes les récompenses que l'on obtiendra dans le futur à partir de cette action).
- On appelle cette fonction  $Q^\pi$  la fonction valeur état/action. Si on appelle  $Q^*$  la fonction de valeur état/action optimale, la politique optimale produit donc le  $a$  qui maximise  $Q^*(s, a)$ !

# Fonction de valeur état/action

- Si on ne connaît pas le modèle, même en connaissant  $V^*$ , on ne peut pas trouver la politique optimale, car la politique gourmande cherche à maximiser  $\sum_{s' \in S} P(s', r|s, a)[r + \gamma V(s')]$
- **Solution:** apprendre plutôt une fonction qui prend en paramètre  $s$  l'état actuel et  $a$  la prochaine action et vaut  $Q^\pi(s, a) = R_{s,s'}^a + \gamma V(s')$  (toutes les récompenses que l'on obtiendra dans le futur à partir de cette action).
- On appelle cette fonction  $Q^\pi$  la fonction valeur état/action. Si on appelle  $Q^*$  la fonction de valeur état/action optimale, la politique optimale produit donc le  $a$  qui maximise  $Q^*(s, a)$ !

# Fonction de valeur état/action

- Si on ne connaît pas le modèle, même en connaissant  $V^*$ , on ne peut pas trouver la politique optimale, car la politique gourmande cherche à maximiser  $\sum_{s' \in S} P(s', r|s, a)[r + \gamma V(s')]$
- **Solution:** apprendre plutôt une fonction qui prend en paramètre  $s$  l'état actuel et  $a$  la prochaine action et vaut  $Q^\pi(s, a) = R_{s,s'}^a + \gamma V(s')$  (toutes les récompenses que l'on obtiendra dans le futur à partir de cette action).
- On appelle cette fonction  $Q^\pi$  la fonction valeur état/action. Si on appelle  $Q^*$  la fonction de valeur état/action optimale, la politique optimale produit donc le  $a$  qui maximise  $Q^*(s, a)$ !

# La fonction Q comme tableau à double entrées

Dans le monde discret, on peut simplement représenter  $Q$  à l'aide d'un tableau.

		actions			
states		$a_0$	$a_1$	$a_2$	...
$s_0$	$Q(s_0, a_0)$	$Q(s_0, a_1)$	$Q(s_0, a_2)$	...	
$s_1$	$Q(s_1, a_0)$	$Q(s_1, a_1)$	$Q(s_1, a_2)$	...	
$s_2$	$Q(s_2, a_0)$	$Q(s_2, a_1)$	$Q(s_2, a_2)$	...	

## Politique gourmande et valeur état/action

Une politique gourmande pour une fonction  $Q$  donnée est la politique qui choisit l'action maximisant  $Q(s, a)$  pour  $s$  l'état actuel.

La politique optimale  $\pi^*$  est également une politique gourmande sur  $Q^*$ . Dans la politique optimale, on a donc l'égalité suivante:

$$Q^*(s_t, a_t) = R_{s_t, s_{t+1}}^{a_t} + \gamma \max_a Q^*(s_{t+1}, a)$$

## Politique gourmande et valeur état/action

Une politique gourmande pour une fonction  $Q$  donnée est la politique qui choisit l'action maximisant  $Q(s, a)$  pour  $s$  l'état actuel.

La politique optimale  $\pi^*$  est également une politique gourmande sur  $Q^*$ . Dans la politique optimale, on a donc l'égalité suivante:

$$Q^*(s_t, a_t) = R_{s_t, s_{t+1}}^{a_t} + \gamma \max_a Q^*(s_{t+1}, a)$$

# Exploration et exploitation

Comment faire les expériences?

- Essayer des actions au hasard ? Efficace pour tout explorer mais très long!
- Suivre la politique optimale ? On risque d'être très vite "bloqué" sur un optimum local!

Il faut donc un compromis entre les deux, le plus simple étant  $\epsilon$ -greedy: suivre la politique optimale avec une probabilité  $1 - \epsilon$ , et choisir une action au hasard avec une probabilité  $\epsilon$ .

1 Introduction

2 Formalisme

3 Politique optimale, value iteration

4 Apprentissage sans modèle

5 Monte Carlo

6 TD et Q-Learning

## Monte Carlo: principe

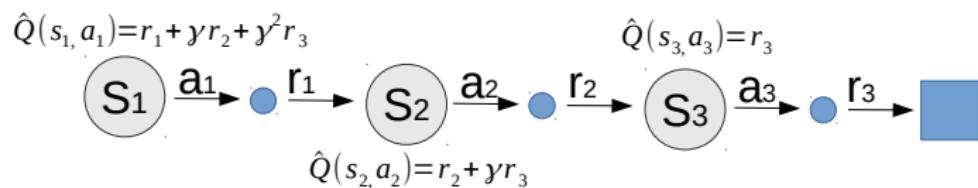
La politique optimale sélectionne l'action  $a$  qui maximise  $Q^*(s, a)$ , où  $Q^*$  est la fonction état/valeur optimale.

Sur la même idée que value iteration, on peut donc utiliser une politique  $\epsilon$ -gourmande pour notre estimation actuelle de  $Q$ , et mettre à jour  $Q$ .

## Monte Carlo: principe

Pour une politique donnée (par exemple  $\epsilon$ -greedy avec  $Q$ ) , on peut réaliser des expériences, jusqu'à atteindre un état terminal.

On peut alors, pour chaque couple  $(s, a)$  dans l'épisode, calculer la somme des récompenses obtenues à partir de cet état/action.



Notre estimation de  $Q(s, a)$  sera alors une moyenne des expériences qu'on a effectuées.

# Monte Carlo

Ce qui nous donne l'algorithme suivant:

Initialize, for all  $s \in \mathcal{S}$ ,  $a \in \mathcal{A}(s)$ :

$Q(s, a) \leftarrow$  arbitrary

$\pi(s) \leftarrow$  arbitrary

$Returns(s, a) \leftarrow$  empty list

Repeat forever:

    Choose  $S_0 \in \mathcal{S}$  and  $A_0 \in \mathcal{A}(S_0)$  s.t. all pairs have probability  $> 0$

    Generate an episode starting from  $S_0, A_0$ , following  $\pi$

    For each pair  $s, a$  appearing in the episode:

$G \leftarrow$  return following the first occurrence of  $s, a$

        Append  $G$  to  $Returns(s, a)$

$Q(s, a) \leftarrow \text{average}(Returns(s, a))$

    For each  $s$  in the episode:

$\pi(s) \leftarrow \arg \max_a Q(s, a)$

# Avantages et limitations de Monte Carlo

- Monte Carlo fournit des vrais échantillons de  $Q(s, a)$ , puisqu'on va jusqu'au bout de l'épisode pour l'obtenir
- Il faut justement attendre la fin de l'épisode pour l'obtenir
- Ne fonctionne pas dans des environnements non-épisodiques

# Avantages et limitations de Monte Carlo

- Monte Carlo fournit des vrais échantillons de  $Q(s, a)$ , puisqu'on va jusqu'au bout de l'épisode pour l'obtenir
- Il faut justement attendre la fin de l'épisode pour l'obtenir
- Ne fonctionne pas dans des environnements non-épisodiques

# Avantages et limitations de Monte Carlo

- Monte Carlo fournit des vrais échantillons de  $Q(s, a)$ , puisqu'on va jusqu'au bout de l'épisode pour l'obtenir
- Il faut justement attendre la fin de l'épisode pour l'obtenir
- Ne fonctionne pas dans des environnements non-épisodiques

1 Introduction

2 Formalisme

3 Politique optimale, value iteration

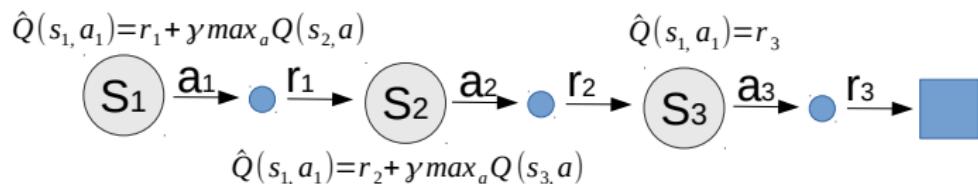
4 Apprentissage sans modèle

5 Monte Carlo

6 TD et Q-Learning

# Temporal Differences

Idée: Au lieu de prendre un "vrai" échantillon (jusqu'à bout de l'épisode), utiliser immédiatement le tuple  $(s_t, a_t, r_t, s_{t+1})$  pour faire la mise à jour à partir de notre estimation actuelle.



# Temporal Differences

Notre nouvelle estimation de  $Q(s_t, a_t)$  devient alors:

$$r_t + \gamma \max_a Q(s_{t+1}, a)$$

On peut donc mettre à jour  $Q(s_t, a_t)$  légèrement dans cette direction:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)]$$

Où  $\alpha \in [0, 1]$  est appelé le taux d'apprentissage (souvenez vous du calcul incrémental de la moyenne!)

# Temporal Differences

Notre nouvelle estimation de  $Q(s_t, a_t)$  devient alors:

$$r_t + \gamma \max_a Q(s_{t+1}, a)$$

On peut donc mettre à jour  $Q(s_t, a_t)$  légèrement dans cette direction:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)]$$

Où  $\alpha \in [0, 1]$  est appelé le taux d'apprentissage (souvenez vous du calcul incrémental de la moyenne!)

# Q-Learning

Ce qui nous donne l'algorithme du Q-Learning:

Initialize  $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

    Initialize  $S$

    Repeat (for each step of episode):

        Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)

        Take action  $A$ , observe  $R, S'$

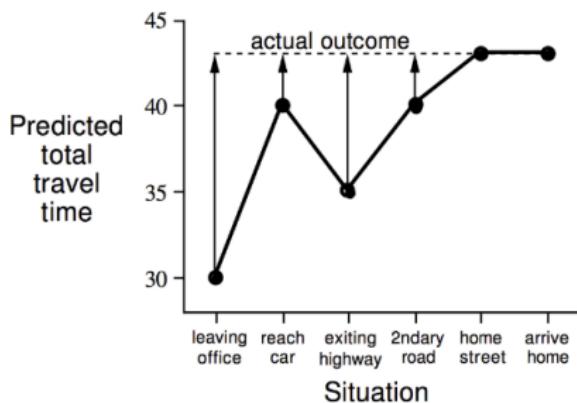
$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$$

$S \leftarrow S'$ ;

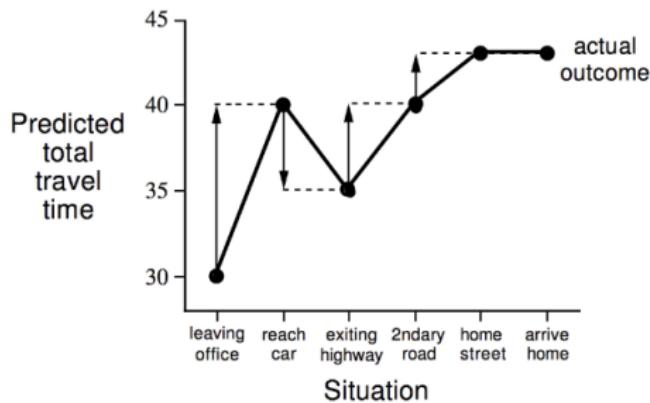
    until  $S$  is terminal

# Monte Carlo vs TD

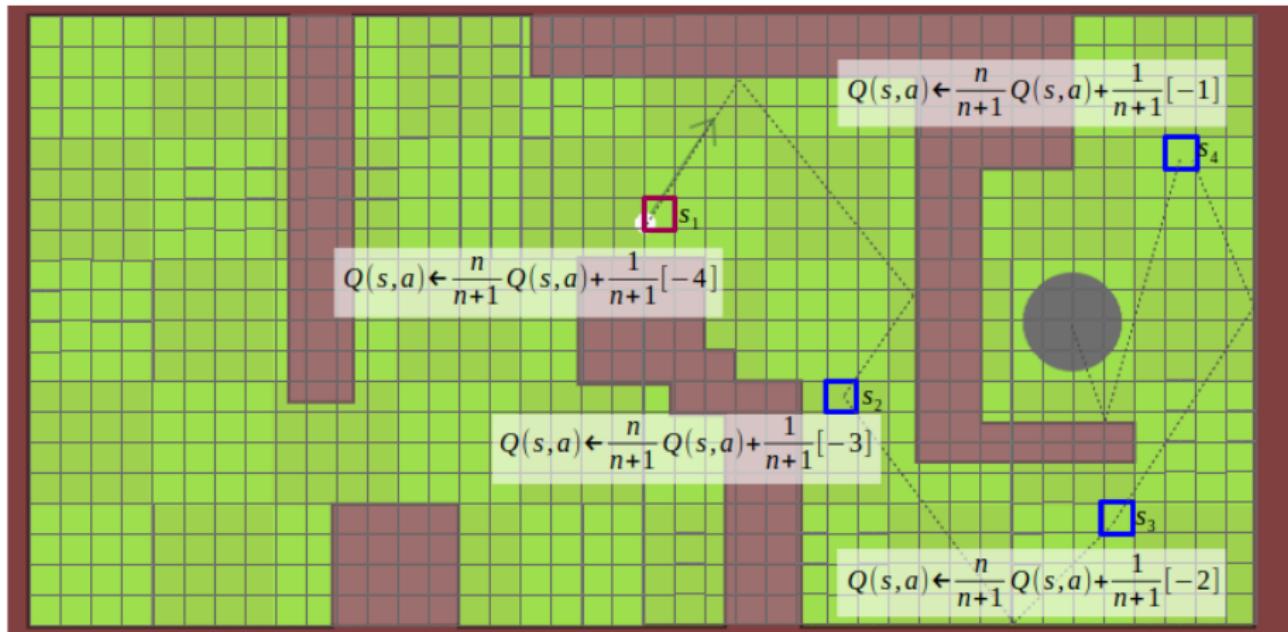
Changes recommended by Monte Carlo methods ( $\alpha=1$ )



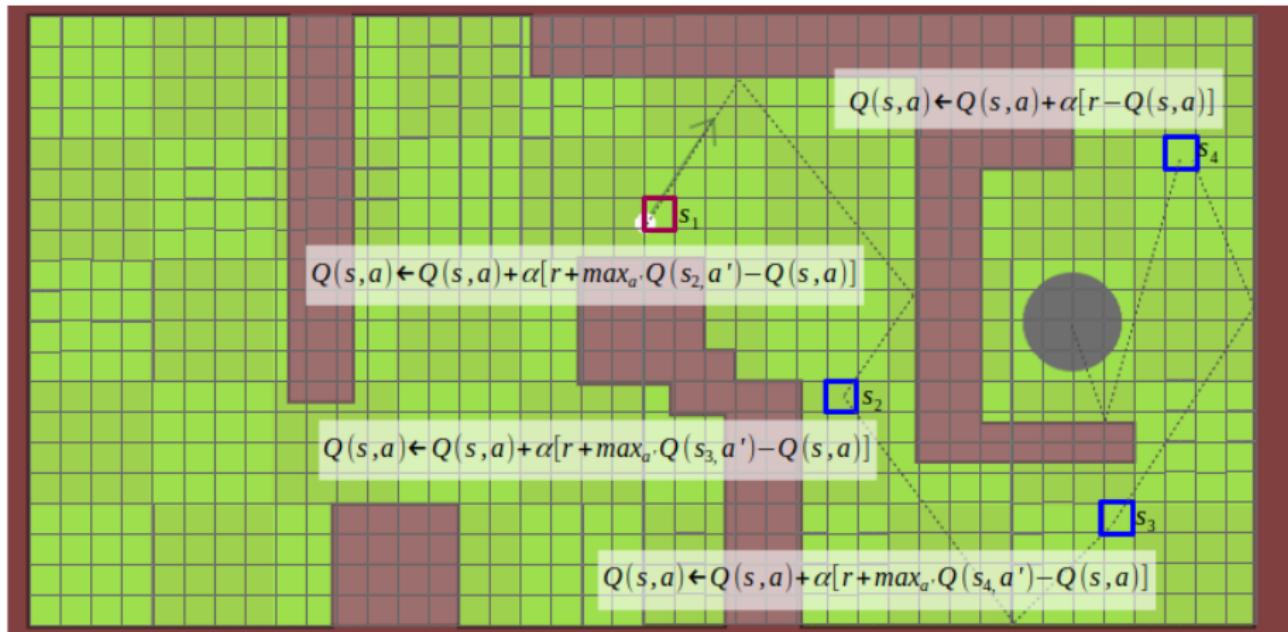
Changes recommended by TD methods ( $\alpha=1$ )



# Mise à jour Monte Carlo



# Mise à jour TD (Q-Learning)



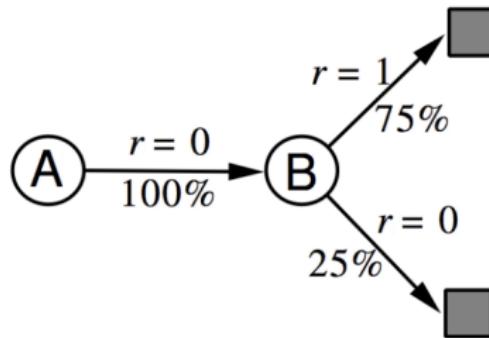
# Monte Carlo vs TD

Two states  $A, B$ ; no discounting; 8 episodes of experience

A, 0, B, 0

B, 1

B, 0



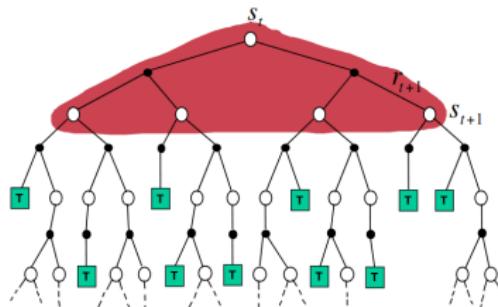
What is  $V(A), V(B)$ ?

- TD exploite la propriété de Markov, et donc  $V(A)$  dépend récursivement de  $V(B)$
- Monte Carlo n'utilisera que l'expérience qui est passée par  $A$  pour définir sa valeur, qui sera donc ici 0

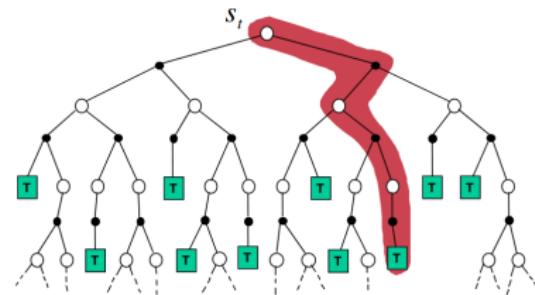
# Mises à jours selon la méthode

On peut comparer la manière de mettre à jour l'estimation dans les 3 cas (value iteration, monte carlo et temporal differences).

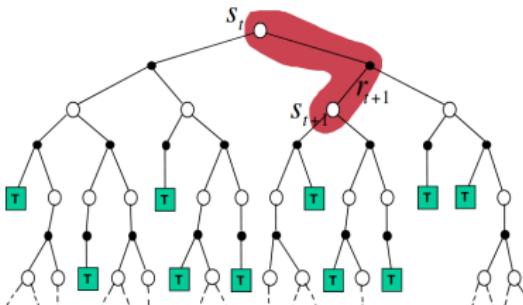
$$V(S_t) \leftarrow \mathbb{E}_\pi [R_{t+1} + \gamma V(S_{t+1})]$$



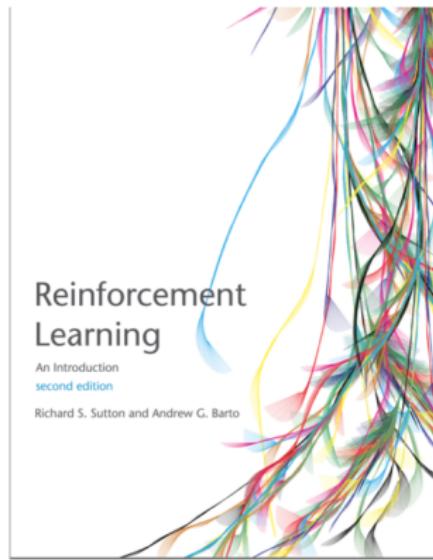
$$V(S_t) \leftarrow V(S_t) + \alpha (G_t - V(S_t))$$



$$V(S_t) \leftarrow V(S_t) + \alpha (R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$



# Le livre de référence



**R. S. Sutton and A. G. Barto (1998)**  
Reinforcement Learning : An Introduction.  
*MIT Press, 1998*