

Reinforcement Learning

Clément Romac (Hugging Face & Inria)

clement.romac@inria.fr

https://github.com/ClementRomac/Teaching/tree/main/ENSC3A_RL_2024-2025

Contenu

- Markov Decision Processes
 - Bellman Equations
- Dynamic Programming
 - Value Iteration
 - Policy Iteration
- Model-free RL
 - Q-Learning / SARSA
- Model-Based RL
- Function approximators
 - DQN
 - Policy Gradient

A retenir

- Markov Decision Processes
- Exploration vs Exploitation
- Value / Q-value functions
- Model-free vs Model-based
- Q-learning & DQN

Ressources

Cours:

- David Silver (UCL): <https://www.davidsilver.uk/teaching/>
- Olivier Slgaud (Sorbonne):
<https://www.youtube.com/playlist?list=PLe5mY-Da-ksWV330WbfazLUyOuR59sers>
- Emmanuel Rachelson (MVA): https://erachelson.github.io/RLclass_MVA/

Lectures:

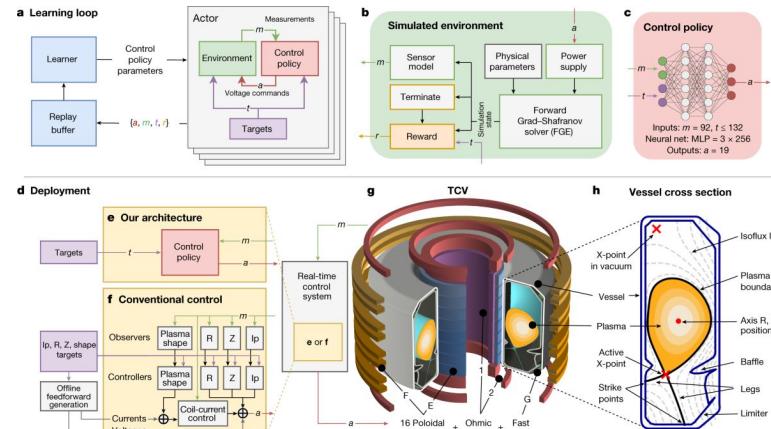
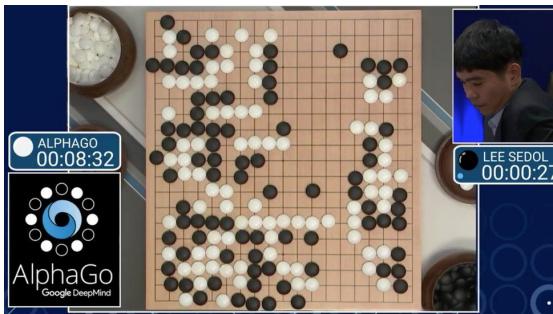
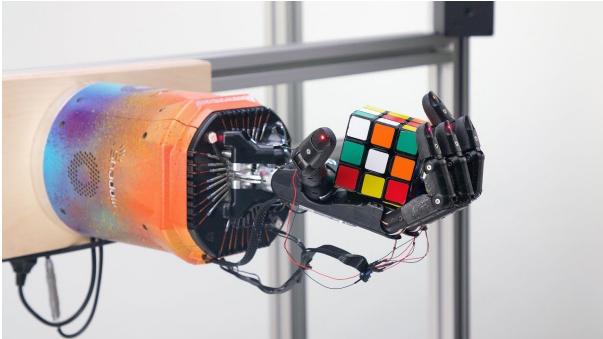
- <http://incompleteideas.net/book/RLbook2020.pdf>
- <https://lilianweng.github.io/posts/2018-02-19-rl-overview/>
- <https://lilianweng.github.io/posts/2018-04-08-policy-gradient/>



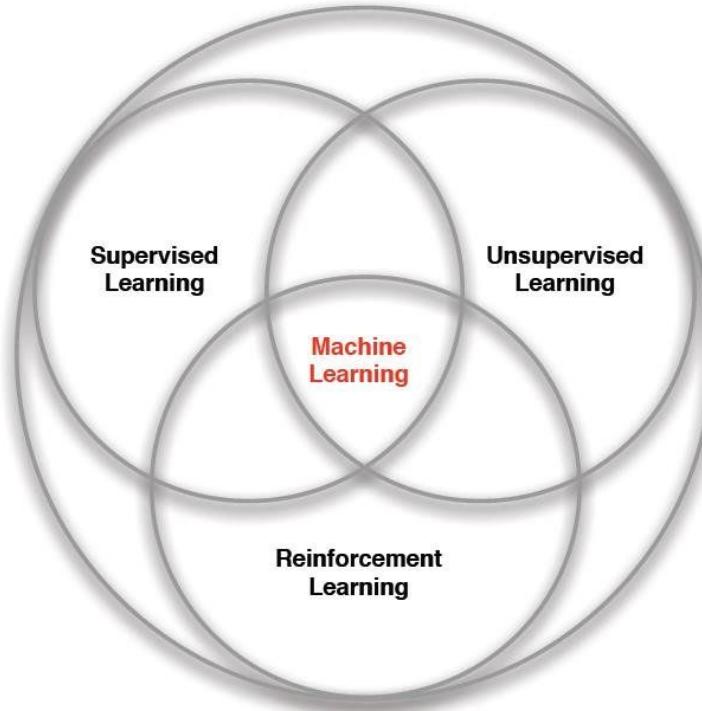
[https://colab.research.google.com/drive/1r-MrVihnqBkWI
ADa-jJZTtMGydpExC9m?usp=sharing](https://colab.research.google.com/drive/1r-MrVihnqBkWIADa-jJZTtMGydpExC9m?usp=sharing)

Introduction

Applications



Une branche du ML



Les caractéristiques d'un problème de RL

- **Problème décisionnel séquentiel**
 - Suite d'**actions** et **observations**
 - Objectif: trouver la suite d'actions optimale
 - Exemples:
 - Jeux (échecs, Mario Kart...)
 - Contrôle (robotique...)
 - Objectif mal défini (RLHF)

Les caractéristiques d'un problème de RL

- **Problème décisionnel séquentiel**
 - Suite d'**actions** et **observations**
 - Objectif: trouver la suite d'actions optimale
 - Exemples:
 - Jeux (échecs, Mario Kart...)
 - Contrôle (robotique...)
 - Objectif mal défini (RLHF)
- **Récompense / Pénalité donnée à chaque actions**
 - Pas de supervision (i.e. **action optimale non-connue**)
 - **Feedback** potentiellement différent
 - Exemples:
 - Echecs: +100 si victoire / - 100 si défaite
 - Atari: Score du jeu

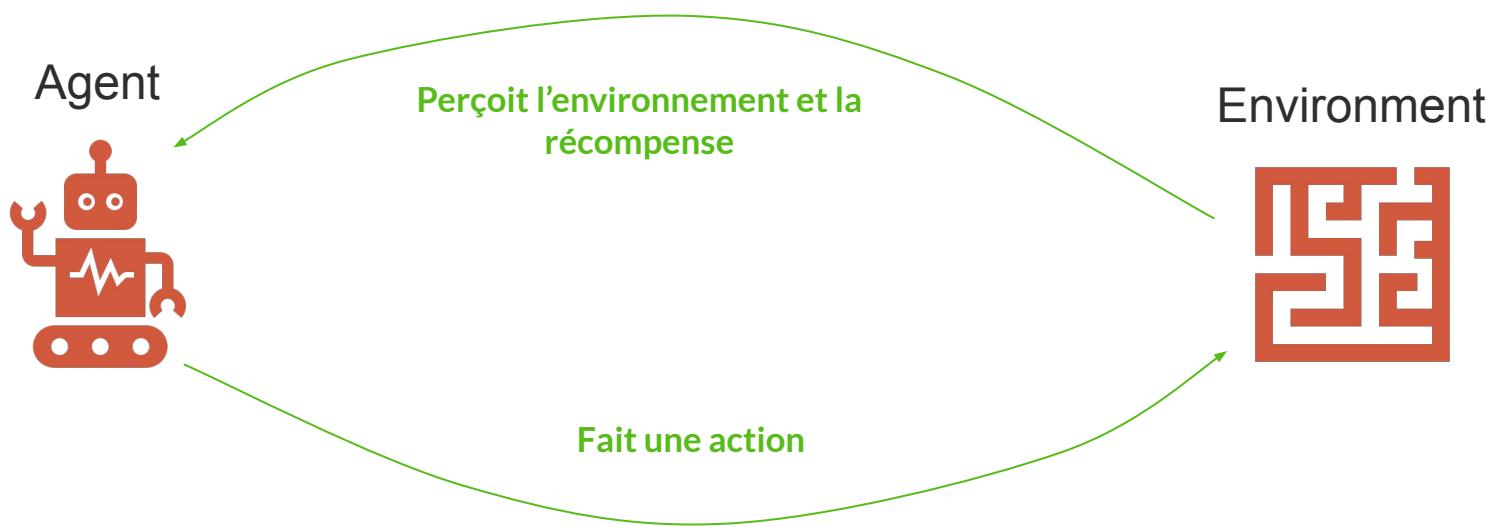
Agent et Environnement



Agent et Environnement

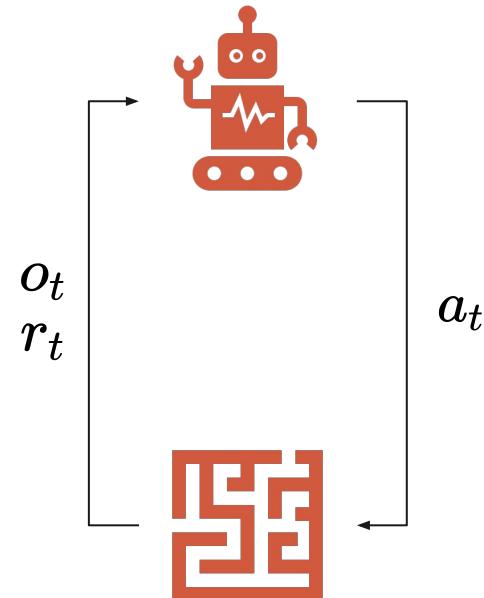


Agent et Environnement



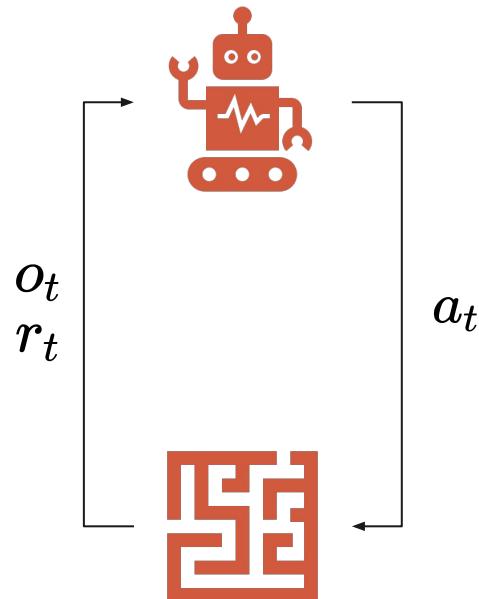
Agent et Environnement

- A chaque pas de temps t , l'agent:
 - Reçoit l'observation o_t
 - Reçoit la récompense r_t
 - Joue l'action a_t



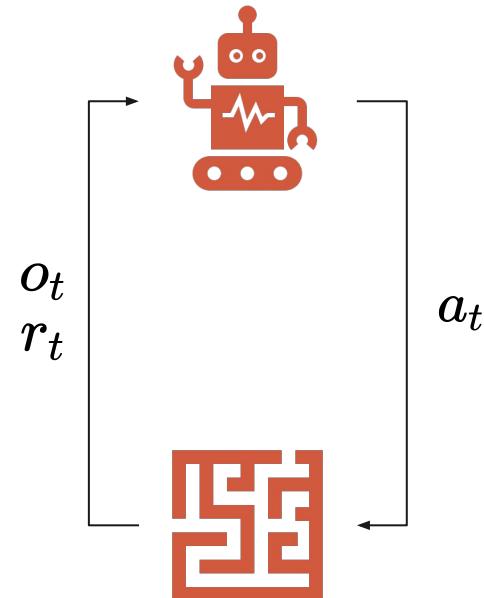
Agent et Environnement

- A chaque pas de temps t , l'agent:
 - Reçoit l'observation o_t
 - Reçoit la récompense r_t
 - Joue l'action a_t
- A chaque pas de temps t , l'environnement:
 - Reçoit l'action a_t
 - Produit l'observation o_{t+1}
 - Produit la récompense r_{t+1}



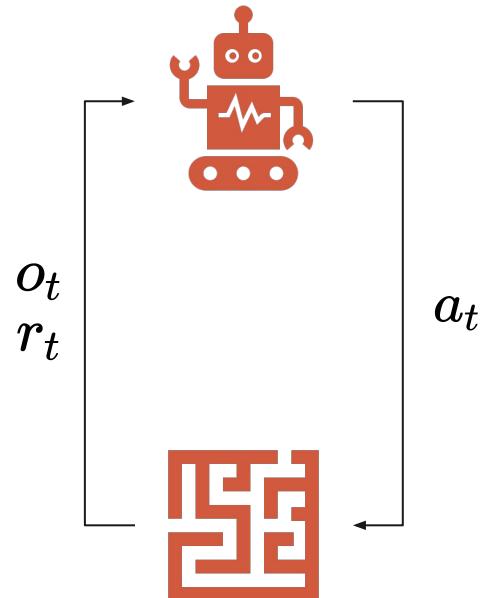
Transitions et Trajectoires

- On appelle **transition** le tuple:
 - $< o_t, a_t, r_t, o_{t+1} >$
- On appelle une **trajectoire** \mathcal{T} une séquence de transitions



Etat vs Observation

- On appelle l'état s_t les informations décrivant l'environnement
- On considère des états **Markoviens**
 - Le prochain état s_{t+1} ne dépend que de s_t
- Si $o_t \neq s_t$, on dit que l'environnement est **Partiellement Observable**
 - Exemples:
 - Un robot qui n'a accès qu'à ses capteurs
 - Un jeu à la première personne



Exemples



?



?

Exemples



Complètement
Observable

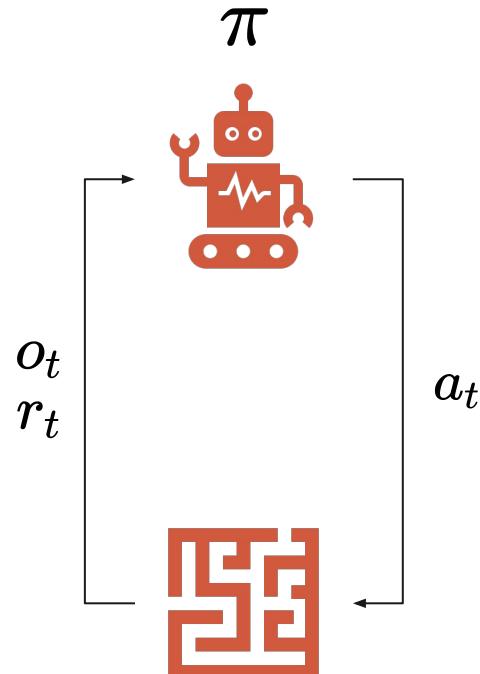


Partiellement
Observable

Politique et Retour

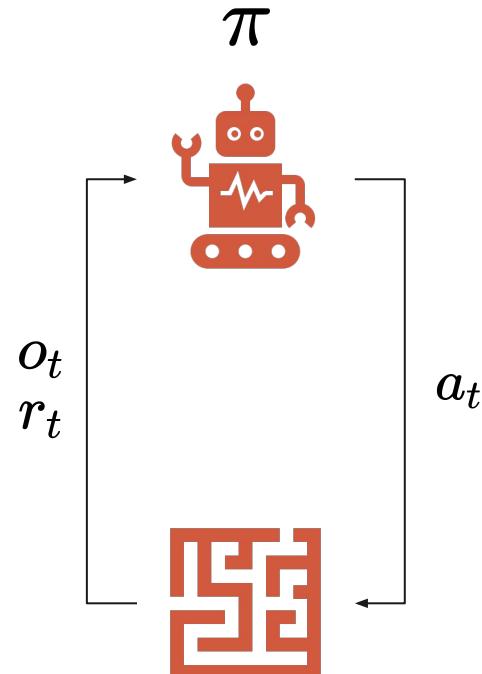
- On appelle une **politique** π la fonction définissant le comportement de l'agent
 - Elle peut être **déterministe**: $a = \pi(s)$
 - Ou **stochastique**: $\pi(a|s) = \mathbb{P}[a|s]$
- On appelle **retour** la somme des récompenses obtenues par π depuis un état s

=> On cherche la **politique optimale** i.e. celle qui **maximise le retour** depuis n'importe quel état



Politique et Retour

- On considère une **somme escompté** avec $\gamma \in [0, 1]$
$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$
 - Utilité mathématique (sommes infinies)
 - Permet de **contrôler l'importance des récompenses futures**
- En pratique la plupart du temps:
 - On considère un **nombre limite** de pas T
 - On appelle **épisode** une trajectoire allant de s_0 à s_T

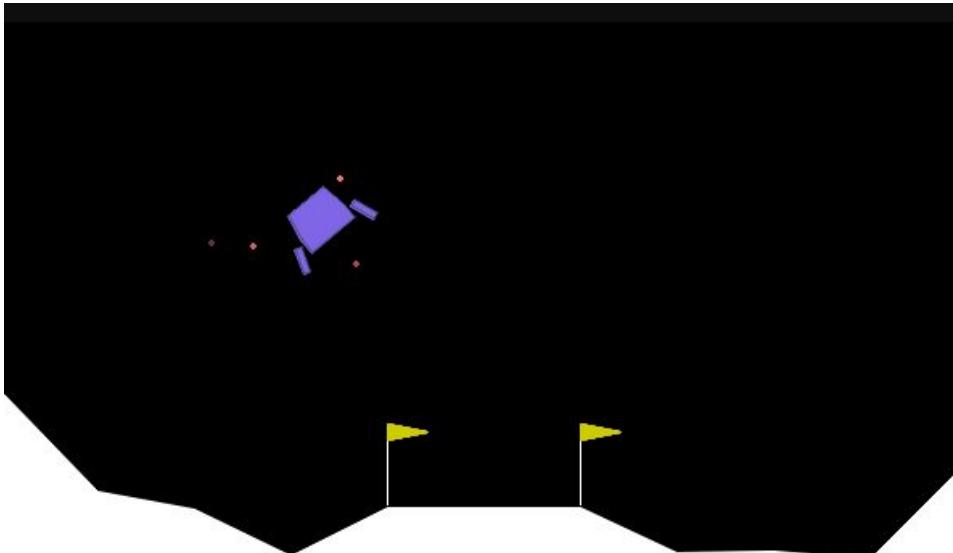


Exemples: Jeu Doom



- Observation:
 - Image
- Action:
 - Etat joysticks + boutons
- Récompense:
 - +106 à chaque monstre tué
 - -5 à chaque tir
 - +1 si toujours en vie

Exemples: Lunar Lander (contrôle)

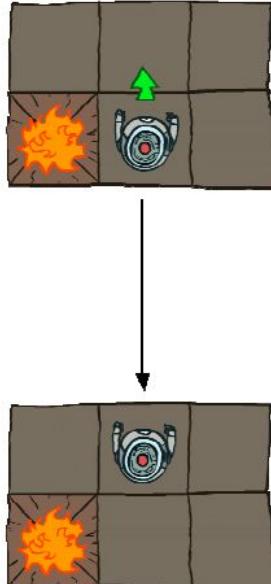


- Observation:
 - Position, angle, vitesse, contact
- Action:
 - Rien/Moteur gauche/Moteur principal/Moteur droit
- Récompense:
 - distance au sol
 - -0.3 pour chaque moteur allumé
 - -100/+100 pour crash/atterrissage

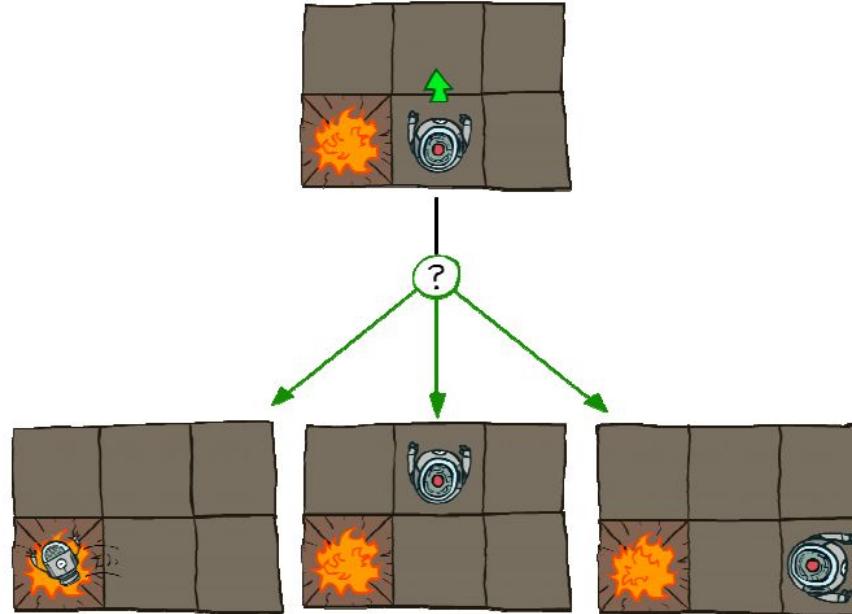
Markov Decision Processes

Une seule formalisation

Deterministic Grid World



Stochastic Grid World



Markov Decision Process (MDP)

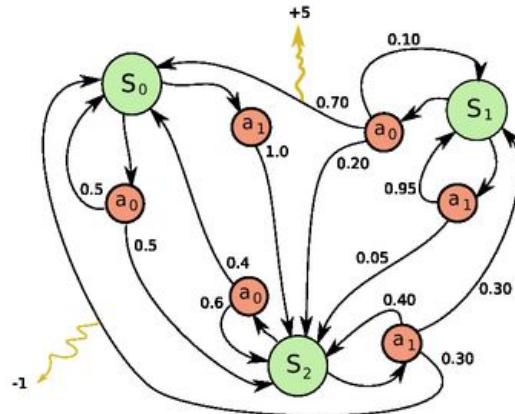
$$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R \rangle$$

- Espace des états: $s \in \mathcal{S}$
- Espace des actions: $a \in \mathcal{A}$
- Fonction de transition: $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$
$$P(s'|s, a) = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$$
- Fonction de récompense: $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{R}$
$$r = R(s, a, s'), r \in \mathcal{R}$$

Markov Decision Process (MDP)

$$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R \rangle$$

- Espace des états: $s \in \mathcal{S}$
- Espace des actions: $a \in \mathcal{A}$
- Fonction de transition: $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$
 $P(s'|s, a) = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$
- Fonction de récompense: $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{R}$
 $r = R(s, a, s'), r \in \mathcal{R}$



$$\mathcal{S} = \{S_0, S_1, S_2\} \quad \mathcal{A} = \{a_0, a_1\}$$

$$P(S_0|a_0, S_1) = 0.7 \quad P(S_2|a_0, S_1) = 0.2$$

$$P(S_1|a_0, S_1) = 0.1$$

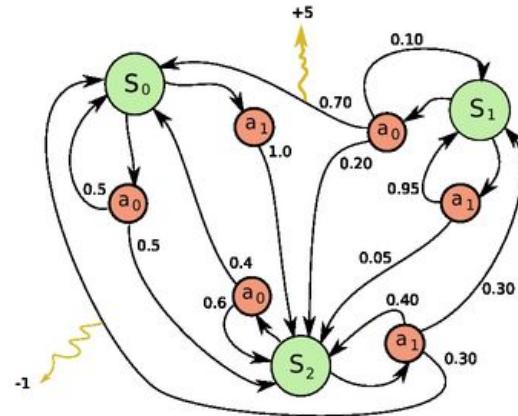
$$R(S_1, a_0, S_0) = 5 \quad R(S_2, a_1, S_0) = -1$$

Markov Decision Process (MDP)

$$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R \rangle$$

- Espace des états: $s \in \mathcal{S}$
- Espace des actions: $a \in \mathcal{A}$
- Fonction de transition: $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$
 $P(s'|s, a) = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$
- Fonction de récompense: $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{R}$
 $r = R(s, a, s'), r \in \mathcal{R}$

$$R(s, a) = \mathbb{E}[R_{t+1} | S_t = s, A_t = a] = \sum_{s' \in \mathcal{S}} P(s', r | s, a) R(s, a, s')$$



$$\mathcal{S} = \{S_0, S_1, S_2\} \quad \mathcal{A} = \{a_0, a_1\}$$

$$P(S_0|a_0, S_1) = 0.7 \quad P(S_2|a_0, S_1) = 0.2$$

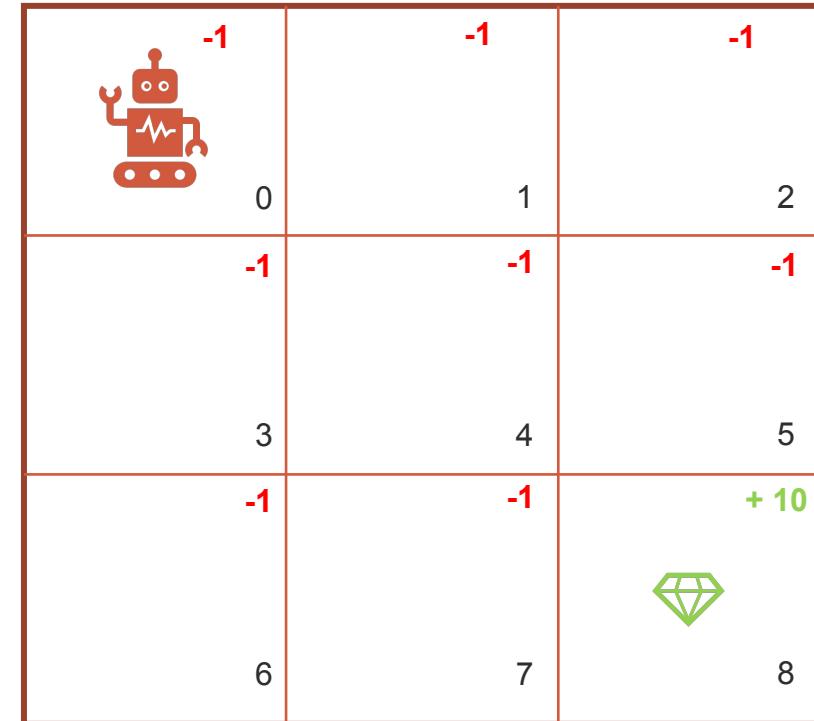
$$P(S_1|a_0, S_1) = 0.1$$

$$R(S_1, a_0, S_0) = 5 \quad R(S_2, a_1, S_0) = -1$$

Exemple: Grille déterministe

$$\mathcal{S} = \{S_0, S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8\}$$

$$\mathcal{A} = \{\text{up, down, right, left}\}$$



Exemple: Grille déterministe

$$\mathcal{S} = \{S_0, S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8\}$$

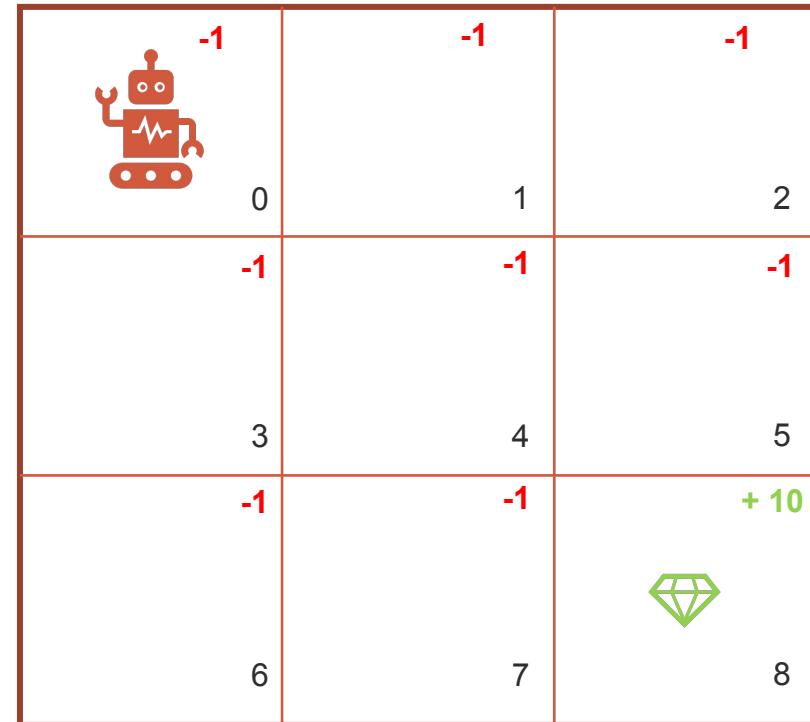
$$\mathcal{A} = \{\text{up, down, right, left}\}$$

$$P(S_0|\text{right}, S_1) = P(S_1|\text{right}, S_2) = \dots = 1$$

$$P(S_1|\text{left}, S_0) = P(S_2|\text{left}, S_1) = \dots = 1$$

$$P(S_3|\text{up}, S_0) = P(S_4|\text{up}, S_1) = \dots = 1$$

$$P(S_0|\text{down}, S_3) = P(S_1|\text{down}, S_4) = \dots = 1$$



Exemple: Grille déterministe

$$\mathcal{S} = \{S_0, S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8\}$$

$$\mathcal{A} = \{\text{up, down, right, left}\}$$

$$P(S_0|\text{right}, S_1) = P(S_1|\text{right}, S_2) = \dots = 1$$

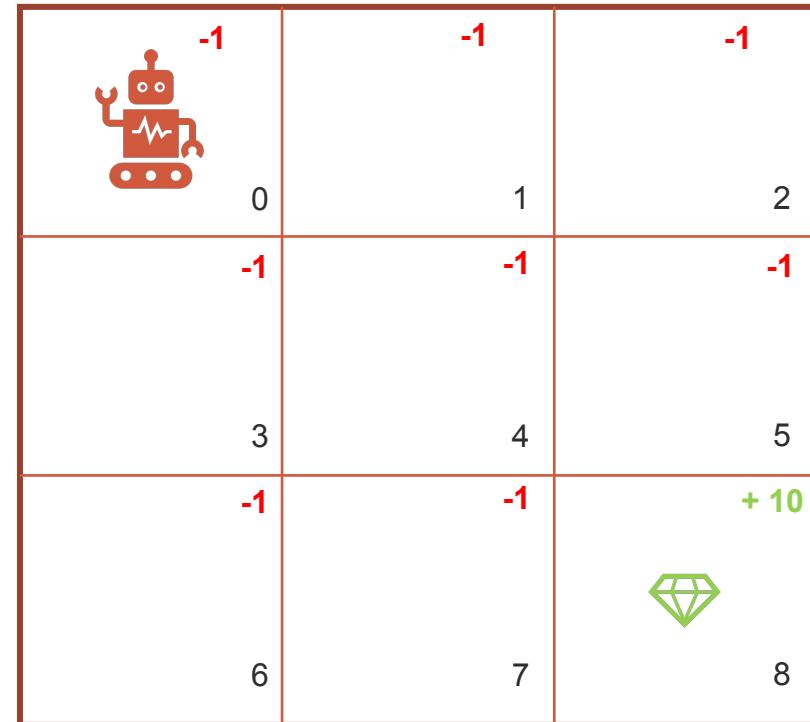
$$P(S_1|\text{left}, S_0) = P(S_2|\text{left}, S_1) = \dots = 1$$

$$P(S_3|\text{up}, S_0) = P(S_4|\text{up}, S_1) = \dots = 1$$

$$P(S_0|\text{down}, S_3) = P(S_1|\text{down}, S_4) = \dots = 1$$

$$R(S_0, \text{right}, S_1) = R(S_0, \text{down}, S_3) = \dots = -1$$

$$R(S_7, \text{right}, S_8) = R(S_5, \text{down}, S_8) = 10$$





TP: Partie 1

TP: Frozen Lake



Valeur

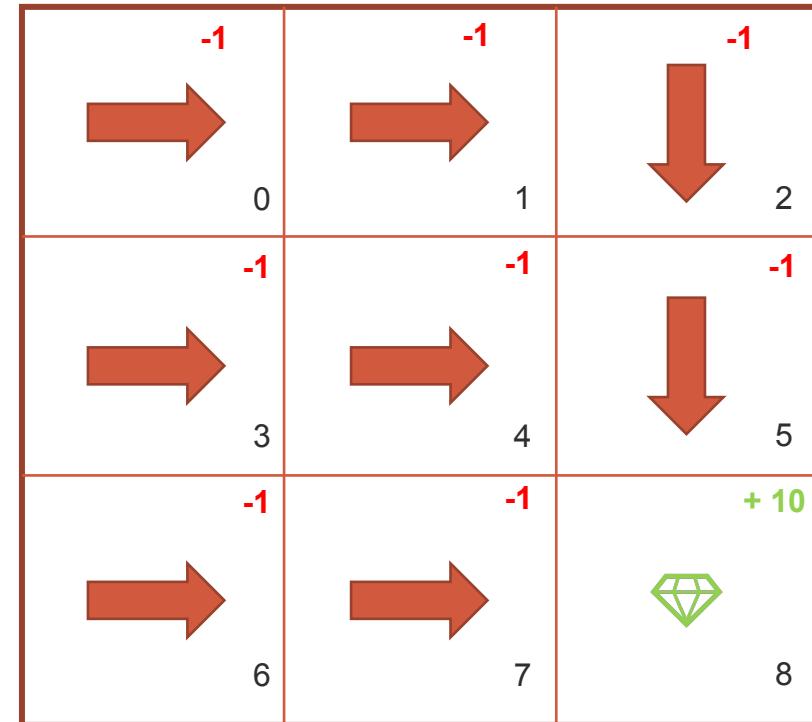
- Rappel du retour: $G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$
- On peut estimer à quel point il est “bon” pour une politique de se trouver dans un état. C'est ce qu'on appelle la **valeur**:

$$V_{\pi}(s) = \mathbb{E}_{\pi}[G_t | S_t = s]$$

Valeur

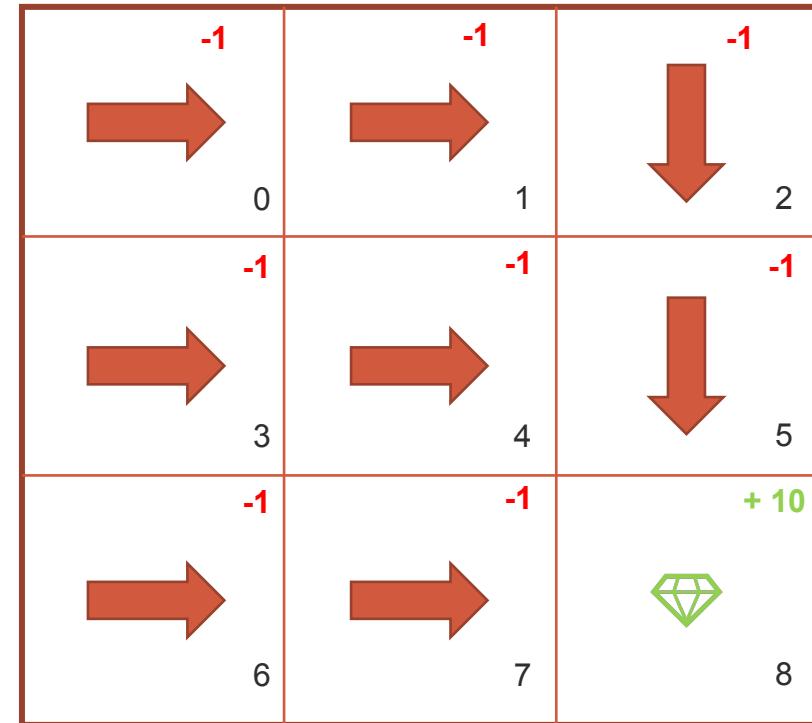
$$\begin{aligned}\pi(S_O, \text{right}) &= \pi(S_1, \text{right}) \\&= \pi(S_3, \text{right}) \\&= \pi(S_4, \text{right}) \\&= \pi(S_6, \text{right}) \\&= \pi(S_7, \text{right}) \\&= 1\end{aligned}$$

$$\pi(S_2, \text{down}) = \pi(S_5, \text{down}) = 1$$



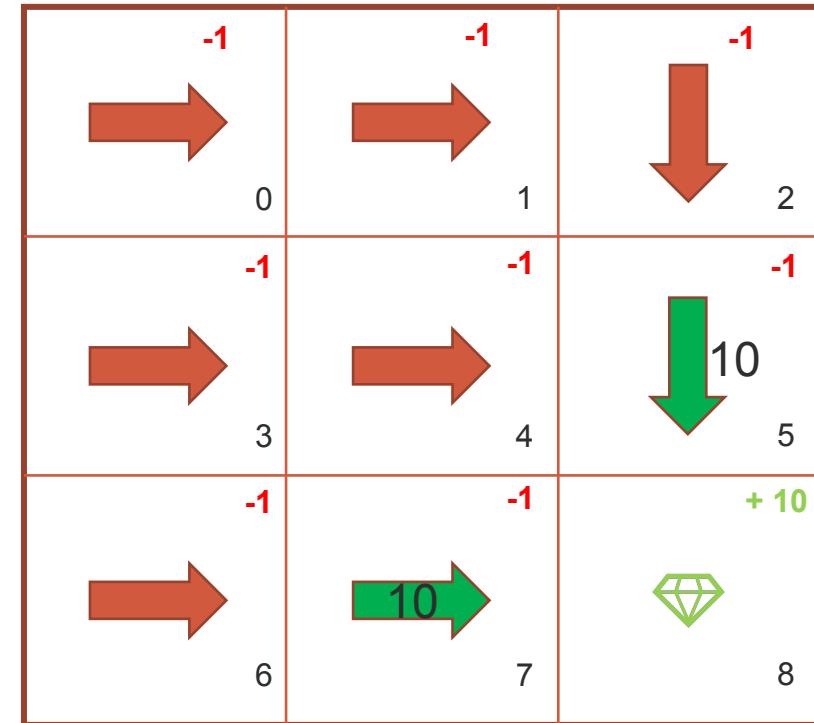
Valeur

$$V_{\pi}(S_7) = V_{\pi}(S_5) = \text{ ?}$$



Valeur

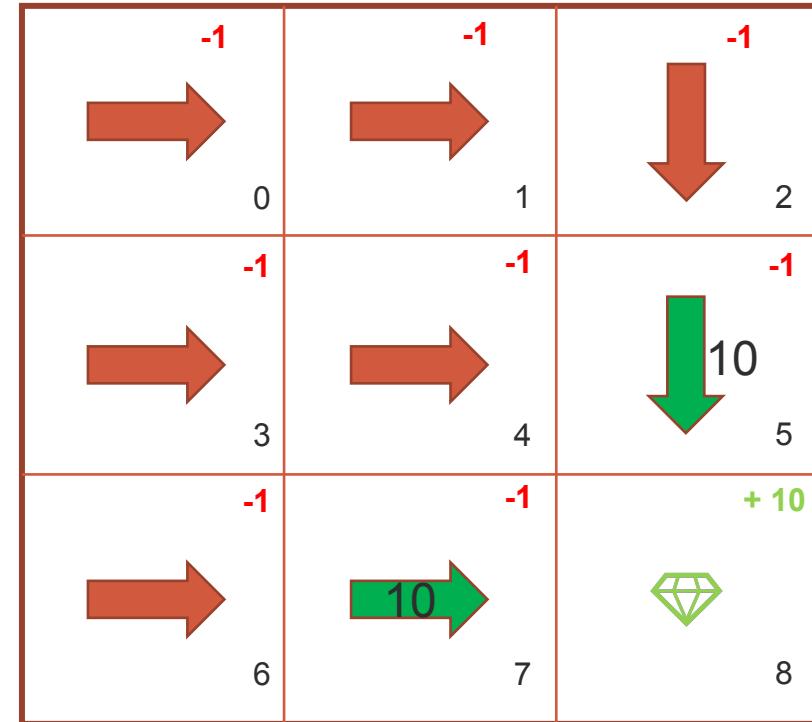
$$V_{\pi}(S_7) = V_{\pi}(S_5) = 10$$



Valeur

$$V_{\pi}(S_7) = V_{\pi}(S_5) = 10$$

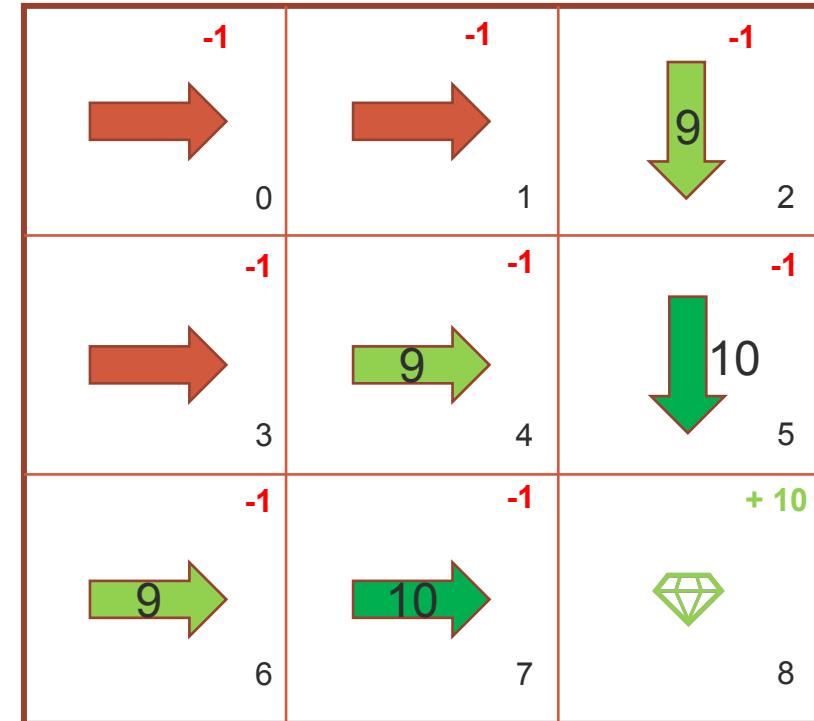
$$V_{\pi}(S_6) = V_{\pi}(S_4) = V_{\pi}(S_2) = ?$$



Valeur

$$V_{\pi}(S_7) = V_{\pi}(S_5) = 10$$

$$V_{\pi}(S_6) = V_{\pi}(S_4) = V_{\pi}(S_2) = -1 + 10 = 9$$

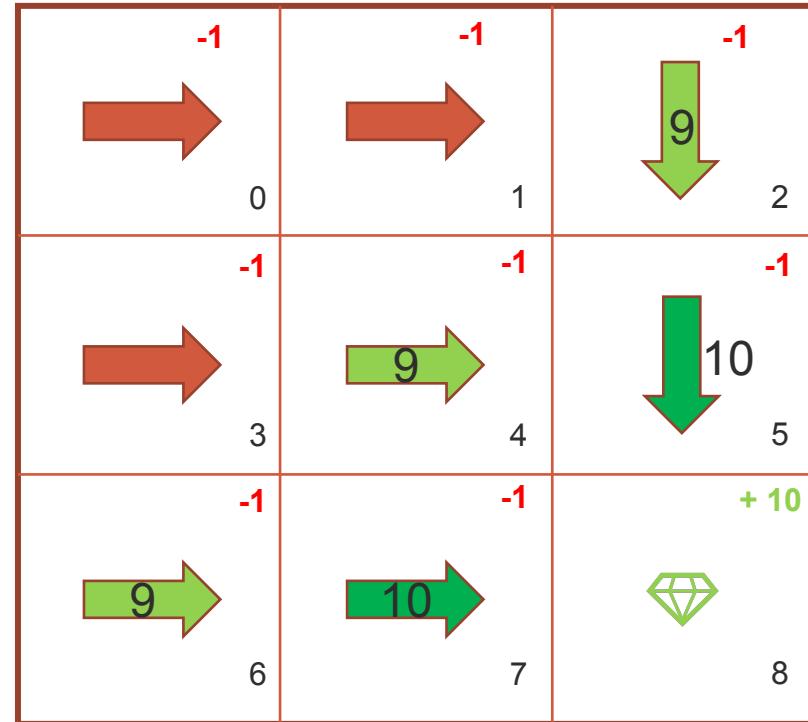


Valeur

$$V_{\pi}(S_7) = V_{\pi}(S_5) = 10$$

$$V_{\pi}(S_6) = V_{\pi}(S_4) = V_{\pi}(S_2) = -1 + 10 = 9$$

$$V_{\pi}(S_1) = V_{\pi}(S_3) = \text{ ?}$$

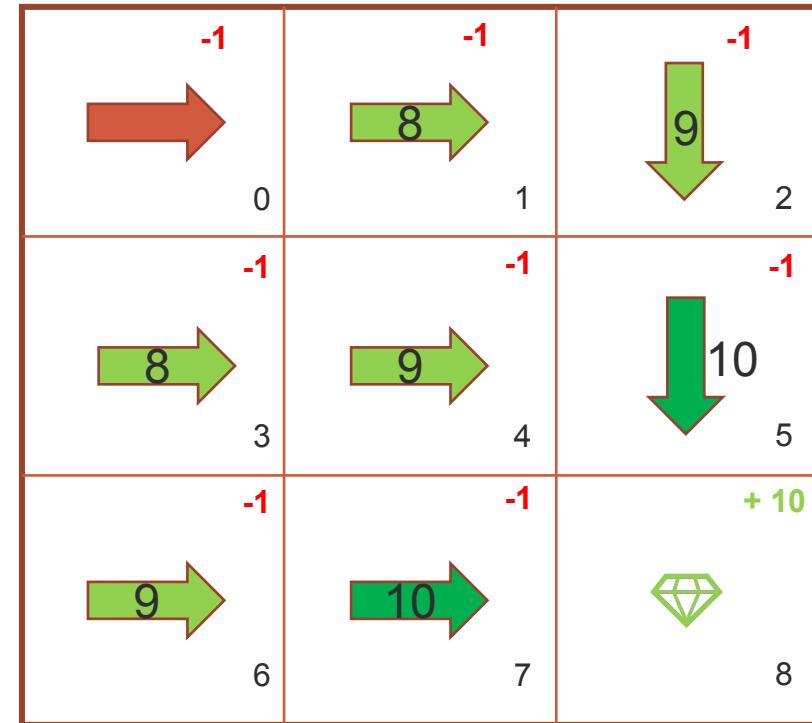


Valeur

$$V_{\pi}(S_7) = V_{\pi}(S_5) = 10$$

$$V_{\pi}(S_6) = V_{\pi}(S_4) = V_{\pi}(S_2) = -1 + 10 = 9$$

$$V_{\pi}(S_1) = V_{\pi}(S_3) = -1 - 1 + 10 = 8$$



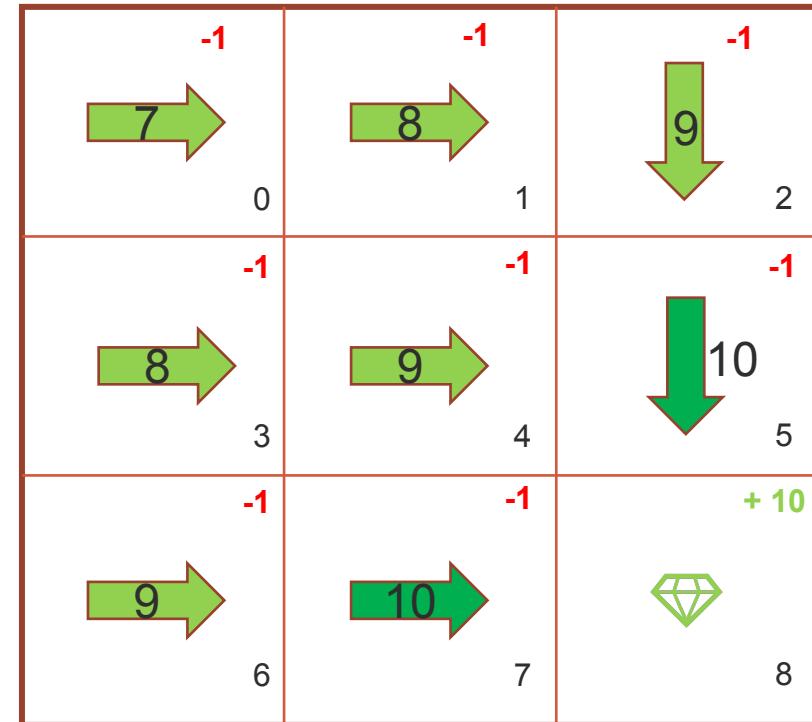
Valeur

$$V_{\pi}(S_7) = V_{\pi}(S_5) = 10$$

$$V_{\pi}(S_6) = V_{\pi}(S_4) = V_{\pi}(S_2) = -1 + 10 = 9$$

$$V_{\pi}(S_1) = V_{\pi}(S_3) = -1 - 1 + 10 = 8$$

$$V_{\pi}(S_0) = -1 + (-1) + (-1) + 10 = 7$$



Q-Valeur

- On peut aller plus loin et estimer **à quel point il est “bon”** pour une politique de choisir une **action** en particulier depuis un **état**. C'est ce qu'on appelle la **Q-valeur**:

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$

- On peut ainsi réécrire la valeur avec:

$$V_\pi(s) = \sum_{a \in \mathcal{A}} Q_\pi(s, a) \pi(a|s)$$

Bellman Equations

Bellman Expectation Equations

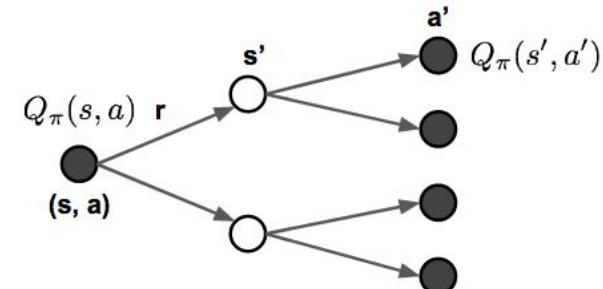
- La valeur et la Q-valeur peuvent être écrites **récursivement**:

$$\begin{aligned} V(s) &= \mathbb{E}[G_t | S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \dots) | S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma G_{t+1} | S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma V(S_{t+1}) | S_t = s] \end{aligned}$$

Bellman Expectation Equations

- La valeur et la Q-valeur peuvent être écrites **récursivement**:

$$\begin{aligned}
 V(s) &= \mathbb{E}[G_t | S_t = s] \\
 &= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s] \\
 &= \mathbb{E}[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \dots) | S_t = s] \\
 &= \mathbb{E}[R_{t+1} + \gamma G_{t+1} | S_t = s] \\
 &= \mathbb{E}[R_{t+1} + \gamma V(S_{t+1}) | S_t = s]
 \end{aligned}$$



$$\begin{aligned}
 Q(s, a) &= \mathbb{E}[R_{t+1} + \gamma V(S_{t+1}) | S_t = s, A_t = a] \\
 &= \mathbb{E}[R_{t+1} + \gamma \mathbb{E}_{a \sim \pi} Q(S_{t+1}, a) | S_t = s, A_t = a]
 \end{aligned}$$

Bellman Expectation Equations

- Plus simple:

$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) Q_\pi(s, a)$$

$$Q_\pi(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V_\pi(s')$$

Bellman Expectation Equations

- Plus simple:

$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) Q_\pi(s, a)$$

$$Q_\pi(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V_\pi(s')$$

$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V_\pi(s') \right)$$

$$Q_\pi(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) \sum_{a' \in \mathcal{A}} \pi(a'|s') Q_\pi(s', a')$$

Politique optimale

- La politique optimale est celle qui maximise la valeur / Q-valeur:

$$\pi_* = \arg \max_{\pi} V_{\pi}(s)$$

$$\pi_* = \arg \max_{\pi} Q_{\pi}(s, a)$$

- On peut aussi définir la valeur / Q-valeur de la politique optimale:

$$V_*(s) = \max_{\pi} V_{\pi}(s)$$

$$Q_*(s, a) = \max_{\pi} Q_{\pi}(s, a)$$

Bellman Optimality Equations

- Valeur / Q-valeur optimales écrites récursivement:

$$V_*(s) = \max_{a \in \mathcal{A}} Q_*(s, a)$$

$$Q_*(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V_*(s')$$

$$V_*(s) = \max_{a \in \mathcal{A}} (R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V_*(s'))$$

$$Q_*(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) \max_{a' \in \mathcal{A}} Q_*(s', a')$$

MDP connu: Dynamic Programming

MDP connu

- Etant donné un MDP il est possible de trouver la politique optimale **sans avoir besoin d'interagir avec l'environnement** !
- Comme on a accès à tout le modèle, on peut s'en servir directement

Evaluation d'une politique

- Etant donné un MDP et une politique, comment évaluer la performance de cette politique ?
=> En calculant sa valeur sur tous les états!
- Comment calculer cette valeur ?

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input π , the policy to be evaluated

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation

Initialize $V(s)$ arbitrarily, for $s \in \mathcal{S}$, and $V(\text{terminal})$ to 0

Loop:

$\Delta \leftarrow 0$

Loop for each $s \in \mathcal{S}$:

$v \leftarrow V(s)$

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

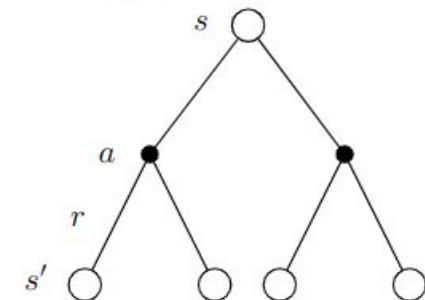
$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$

Améliorer la politique à partir de l'évaluation

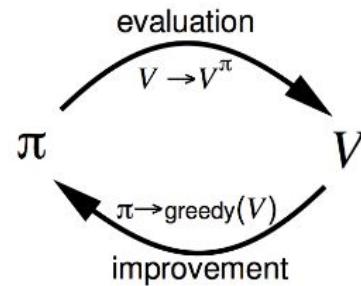
- On peut désormais utiliser notre estimation de la valeur de la politique pour modifier la politique
- Depuis chaque état, on modifie la politique de façon à ce qu'elle choisisse l'action menant à l'état avec la meilleure valeur:

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



Policy Iteration

- On alterne les phases d'évaluation et d'amélioration



Policy Iteration

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization

$V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$; $V(\text{terminal}) \doteq 0$

2. Policy Evaluation

Loop:

$$\Delta \leftarrow 0$$

Loop for each $s \in \mathcal{S}$:

$$v \leftarrow V(s)$$

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

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

until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement

policy-stable \leftarrow true

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

$$\text{old-action} \leftarrow \pi(s)$$

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

If $\text{old-action} \neq \pi(s)$, then *policy-stable* \leftarrow false

If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

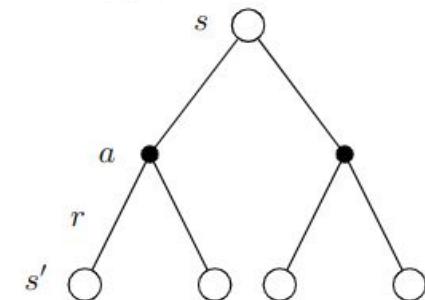
Value Iteration

- L'algorithme de Policy Iteration converge en un temps fini mais requiert à chaque itération une évaluation de la politique sur tout le MDP...
- On peut directement faire converger notre estimation de valeur vers la valeur optimale en assumant une politique greedy:

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

Au lieu de:

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



Value Iteration

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\text{terminal}) = 0$

Loop:

| $\Delta \leftarrow 0$

| Loop 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 $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that

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



TP: Partie 2

Value Iteration

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\text{terminal}) = 0$

Loop:

| $\Delta \leftarrow 0$

| Loop for each $s \in \mathcal{S}$:

| | $v \leftarrow V(s)$

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

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

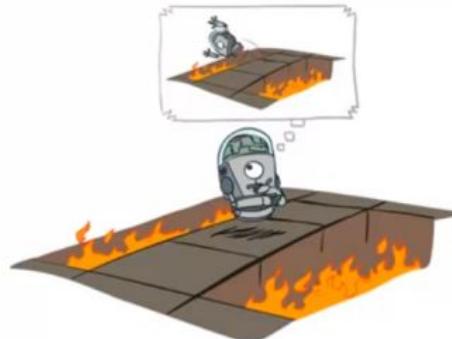
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that

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

Et si on ne connaît pas le MDP ... ?

- On ne peut plus utiliser les méthodes de Dynamic Programming...
- Il faut désormais que l'agent **découvre l'environnement en interagissant** avec
- C'est là que l'apprentissage par renforcement entre en jeu !



Dynamic Programming



Reinforcement Learning

Model-based / Model-free ?

- On peut commencer par explorer le monde et **apprendre un modèle du MDP**
- On utilise ensuite ce modèle pour **trouver la politique optimale** (e.g. Value Iteration)

=> **Model-based RL**

- On peut sinon **mélanger l'exploration et la recherche de la politique optimale** en utilisant pour cette dernière les informations collectées

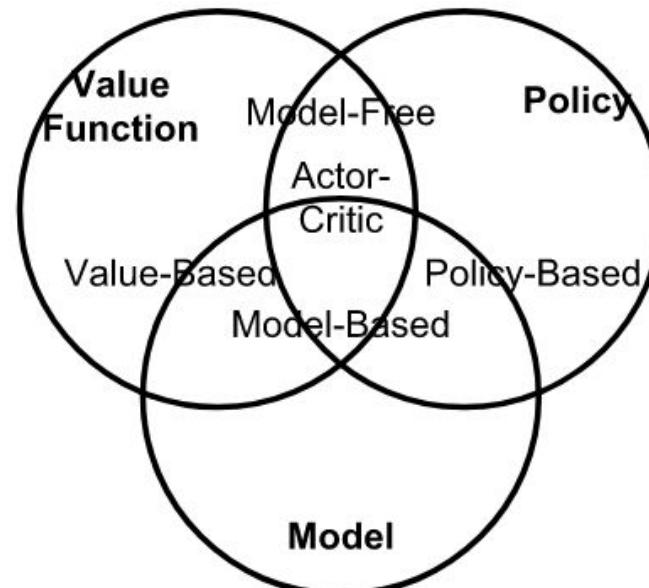
=> **Model-free RL**

Model-based / Model-free ?

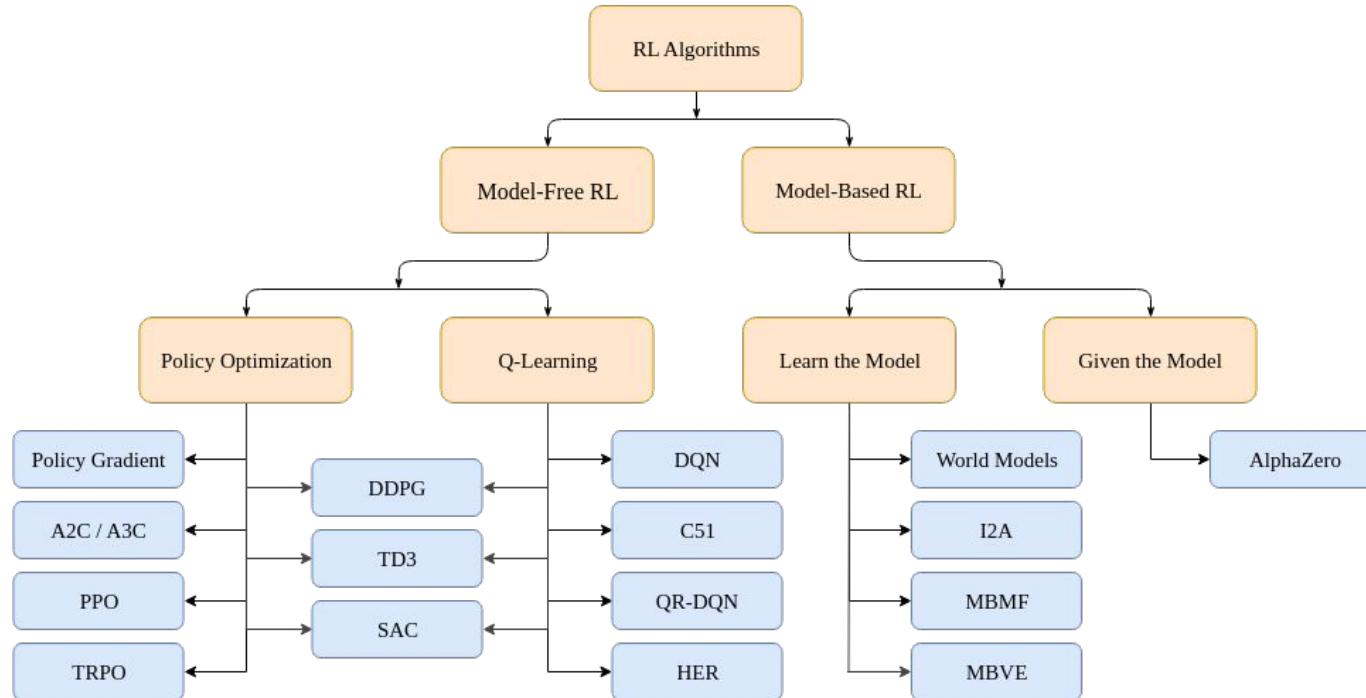
- En pratique, le **Model-free** est majoritaire
- Il est souvent **très (trop) difficile** d'apprendre un bon modèle du MDP
- Une des raisons principales: **comment bien explorer le MDP?**
- On cherche donc plutôt une une politique **sous-optimale**

Familles

- Value-Based:
 - On apprend la **valeur**
 - On s'en sert pour **jouer de façon optimale**
- Policy-Based:
 - On essaie de trouver directement une **fonction de politique**
 - On peut apprendre une valeur également (**Actor-Critic**)



Liste (non-exhaustive) des méthodes

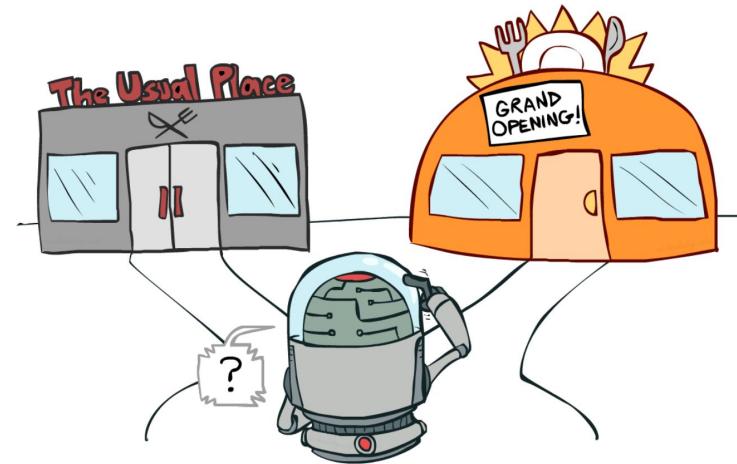


Exploration vs Exploitation

- Choisir un restaurant:
 - **Exploitation:** Aller dans votre restaurant favoris
 - **Exploration:** Essayer un nouveau restaurant
- Jeux:
 - **Exploitation:** Jouer le coup que vous pensez être le meilleur
 - **Exploration:** Tenter un autre coup
- Système de recommandation:
 - **Exploitation:** Proposer les produits ayant le plus de succès
 - **Exploration:** Proposer des nouveaux produits

Exploration vs Exploitation

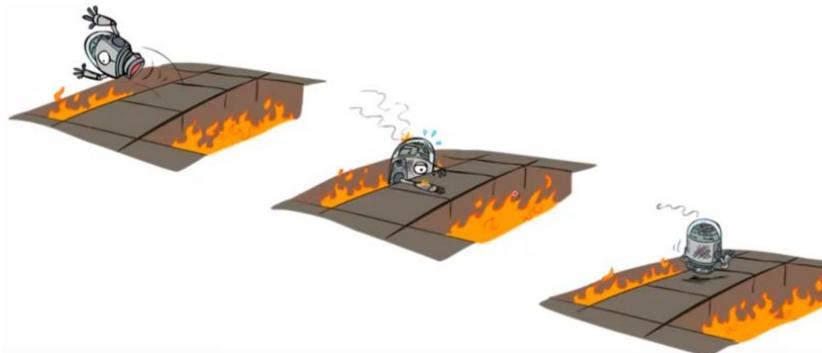
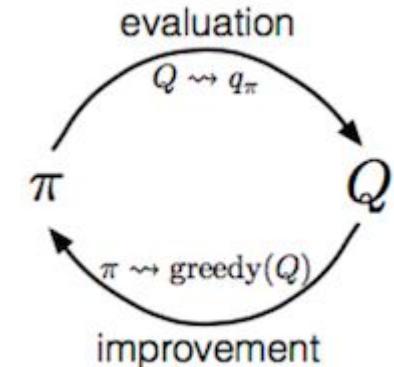
- Trop d'exploration:
 - On n'apprend jamais de bonne politique
- Trop d'exploitation:
 - On peut rater des informations pour trouver une bonne politique



Model-free Value-based RL

Objectif

- Utiliser une politique pour **collecter des transitions**
- Apprendre une estimation de la valeur de cette politique à partir des données collectées
- Jouer de manière optimale étant donné l'estimation actuelle



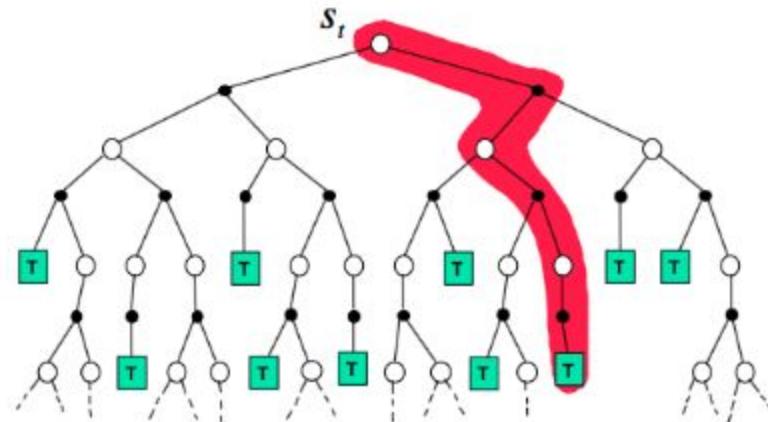
Estimations Monte-Carlo

- Comment estimer la valeur depuis un état ?
 - On définit la valeur d'un état comme la **moyenne des retours obtenus depuis cet état**
 - On utilise la politique **jusqu'à la fin de l'épisode**
 - On **met à jour nos estimations** de valeur pour chaque état visité étant donné le retour collecté (i.e. somme des récompenses) depuis cet état:

$$V(s) = \frac{\sum_{t=1}^T \mathbf{1}[S_t=s] G_t}{\sum_{t=1}^T \mathbf{1}[S_t=s]}$$

Monte-Carlo

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



Estimations Monte-Carlo

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):

Generate an episode following π : $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \dots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless S_t appears in S_0, S_1, \dots, S_{t-1} :

Append G to $Returns(S_t)$

$V(S_t) \leftarrow \text{average}(Returns(S_t))$

Estimations Monte-Carlo

- Défauts:
 - Lent car nécessite de finir un épisode avant de mettre à jour l'estimation de la valeur
 - Forte variance
- Avantages:
 - Utilise des “vrais” retour => stable

Temporal Difference

- Comment estimer la valeur depuis un état ?
 - On s'appuie sur les équations récursives de Bellman:

$$V(s) = \mathbb{E}[R_{t+1} + \gamma V(S_{t+1})|S_t = s]$$

- On calcule une erreur (**TD error**) à partir de notre propre estimation de la prochaine valeur:

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

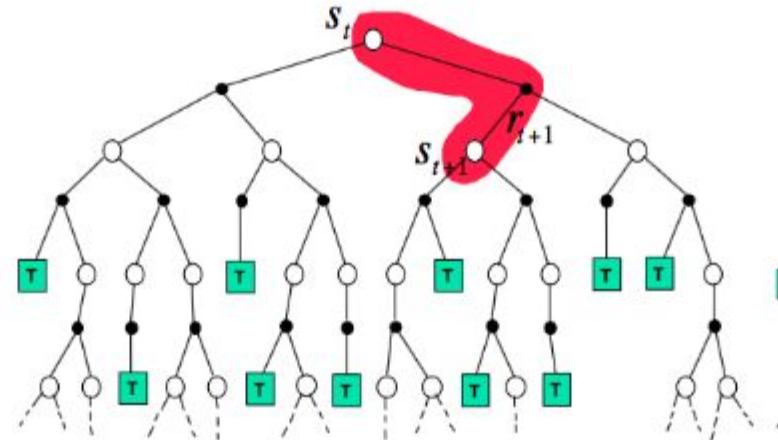
Observation

Ancienne estimations

- On met à jour notre estimation

Temporal-Difference

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

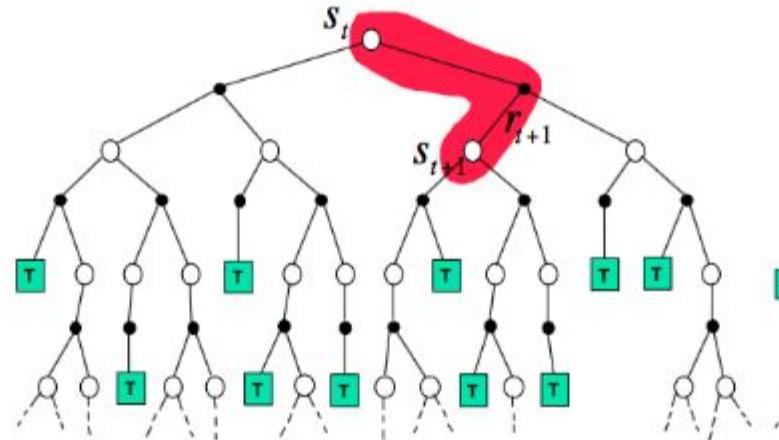


Temporal Difference

- On met à jour notre estimation à partir d'une estimation
 - **Bootstrapping**
- On peut mettre à jour la valeur d'un état **seulement à partir du prochain état**
 - Converge plus rapidement que MC
 - Moins stable

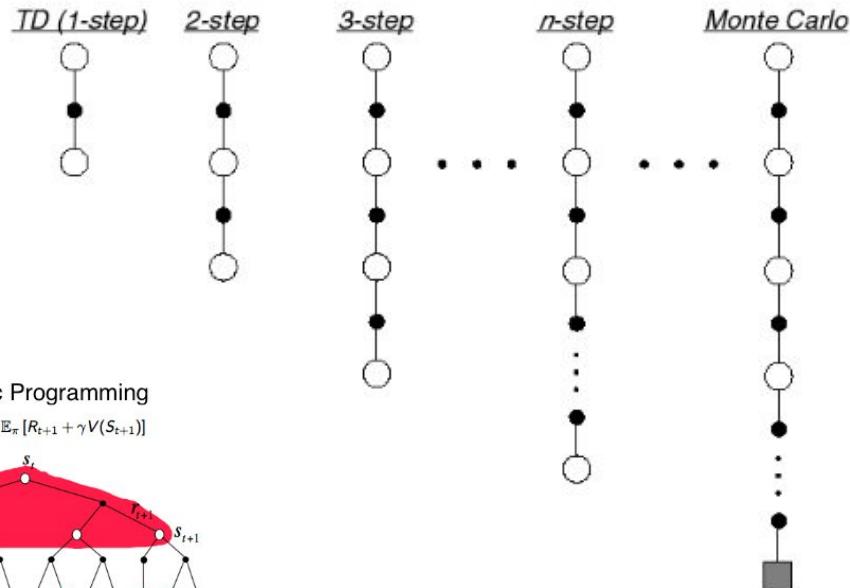
Temporal-Difference

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

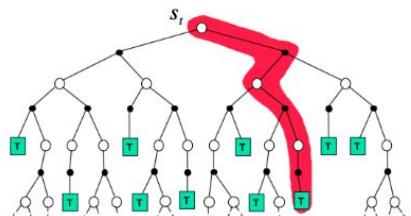


TD(λ)

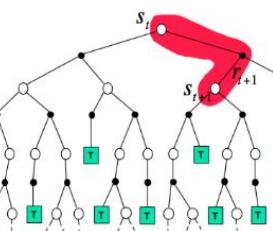
- On peut en réalité faire un mix entre MC et TD
=> **TD(λ)** (aussi appelé n-step return)



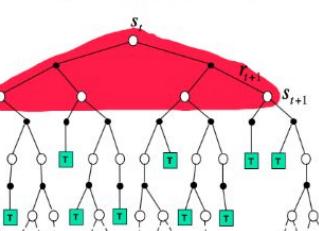
Monte-Carlo
 $V(S_t) \leftarrow V(S_t) + \alpha (G_t - V(S_t))$



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



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



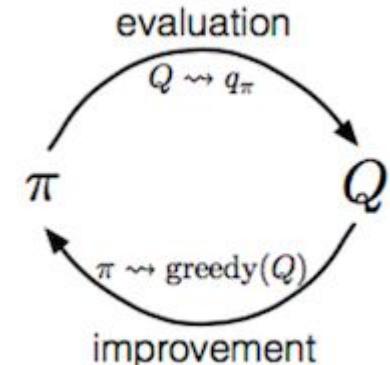
On ajoute la politique maintenant ?

- Comme dans la méthode de Policy Iteration:
 - On **collecte des transitions** en jouant de façon optimale étant donné notre **estimation actuelle de la Q-valeur**:

$$A_t = \arg \max_{a \in \mathcal{A}} Q(S_t, a)$$

$$A_{t+1} = \arg \max_{a \in \mathcal{A}} Q(S_{t+1}, a)$$
 - On met à jour la **Q-valeur** avec la **TD-error**:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$$



On ajoute la politique maintenant ?

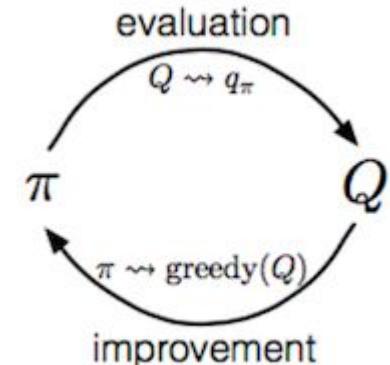
- Comme dans la méthode de Policy Iteration:
 - On **collecte des transitions** en jouant de façon optimale étant donné notre **estimation actuelle de la Q-valeur**:

$$A_t = \arg \max_{a \in \mathcal{A}} Q(S_t, a)$$

$$A_{t+1} = \arg \max_{a \in \mathcal{A}} Q(S_{t+1}, a)$$

- On met à jour la **Q-valeur** avec la **TD-error**:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$$



=> **SARSA**

On ajoute la politique maintenant ?

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

 Initialize S

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

 Loop for each step of episode:

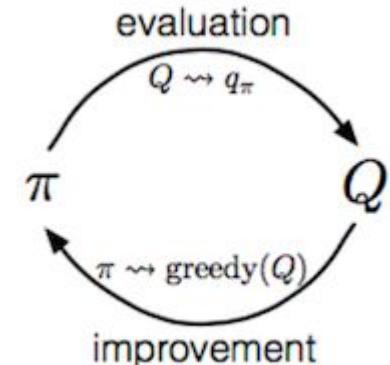
 Take action A , observe R, S'

 Choose A' from S' using policy derived from Q (e.g., ε -greedy)

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

$S \leftarrow S'; A \leftarrow A'$;

 until S is terminal



Off-Policy? Q-Learning



- Comme dans la méthode de Policy Iteration:
 - On **collecte des transitions** en jouant de façon optimale étant donné notre **estimation actuelle de la Q-valeur**:

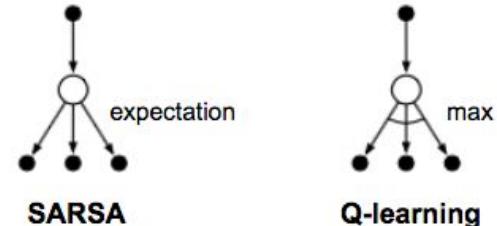
$$A_t = \arg \max_{a \in \mathcal{A}} Q(S_t, a)$$

$$A_{t+1} = \arg \max_{a \in \mathcal{A}} Q(S_{t+1}, a)$$
 - On met à jour la **Q-valeur** avec la **TD-error** en assumant une politique **greedy**:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \boxed{\max_{a \in \mathcal{A}} Q(S_{t+1}, a)} - Q(S_t, A_t))$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)) \Rightarrow \text{SARSA}$$

Off-Policy? Q-Learning



- Comme dans la méthode de Policy Iteration:
 - On **collecte des transitions** en jouant de façon optimale étant donné notre **estimation actuelle de la Q-valeur**:

$$A_t = \arg \max_{a \in \mathcal{A}} Q(S_t, a)$$

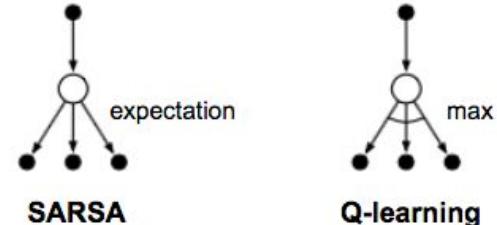
$$A_{t+1} = \arg \max_{a \in \mathcal{A}} Q(S_{t+1}, a)$$
 - On met à jour la **Q-valeur** avec la **TD-error** en assumant une politique **greedy**:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \boxed{\max_{a \in \mathcal{A}} Q(S_{t+1}, a)} - Q(S_t, A_t))$$

=> Pas besoin de collecter A_{t+1}

=> Preuve de convergence plus facile

Off-Policy? Q-Learning



Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

 Initialize S

 Loop for each step of episode:

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

 Take action A , observe R, S'

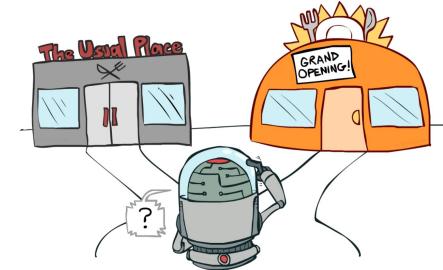
$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$$

$$S \leftarrow S'$$

 until S is terminal

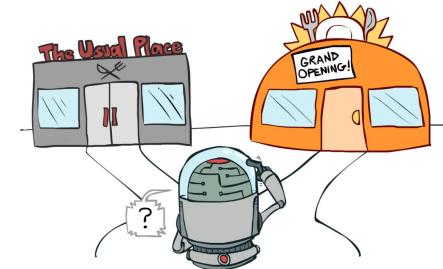
Et l'exploration dans tout ça ?

- SARSA et Q-Learning nécessite d'**explorer**
 - Rappel: on joue greedy à partir des estimations de la Q-valeur
 - => **On ne fait qu'exploiter !**



Et l'exploration dans tout ça ?

- SARSA et Q-Learning nécessite d'**explorer**
 - Rappel: on joue **greedy** à partir des estimations de la Q-valeur
 - => **On ne fait qu'exploiter !**
- Une méthode simple mais toujours utilisée: **ϵ -greedy**
 - Avec une probabilité $1 - \epsilon$: on joue $A_t = \arg \max_{a \in \mathcal{A}} Q(S_t, a)$
 - Sinon: on joue **aléatoirement** $A_t \sim \mathcal{U}(\mathcal{A})$
 - **On commence avec un ϵ haut et on le diminue petit à petit**

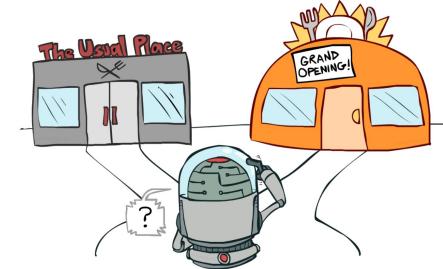


Et l'exploration dans tout ça ?

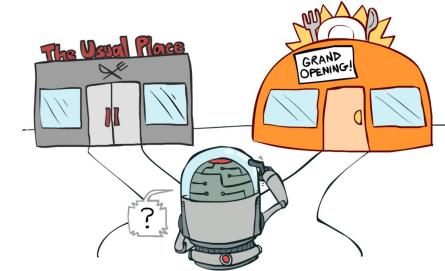
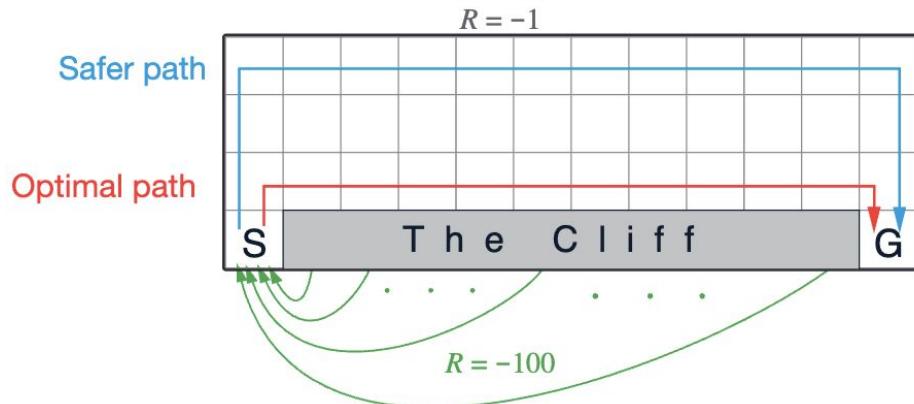
- SARSA et Q-Learning nécessite d'**explorer**
 - Rappel: on joue **greedy** à partir des estimations de la Q-valeur
 - => **On ne fait qu'exploiter !**
- Une méthode simple mais toujours utilisée: **ϵ -greedy**
 - Avec une probabilité $1 - \epsilon$: on joue $A_t = \arg \max_{a \in \mathcal{A}} Q(S_t, a)$
 - Sinon: on joue **aléatoirement** $A_t \sim \mathcal{U}(\mathcal{A})$
 - **On commence avec un ϵ haut et on le diminue petit à petit**

=> **On commence par beaucoup explorer**

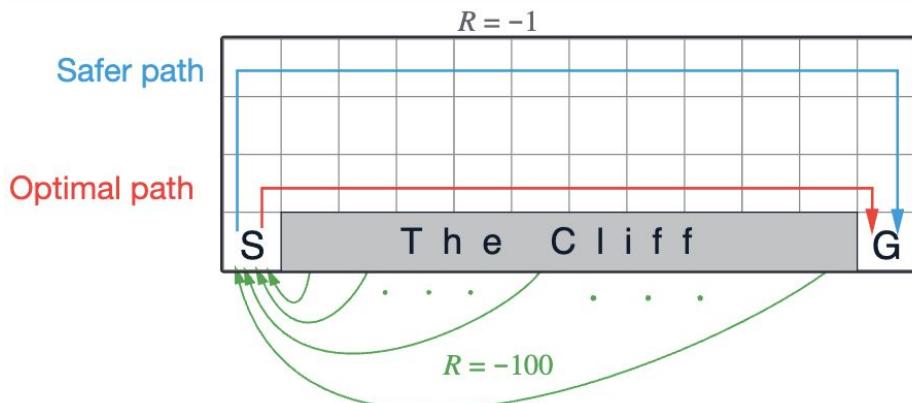
=> **Puis on exploite de plus en plus**



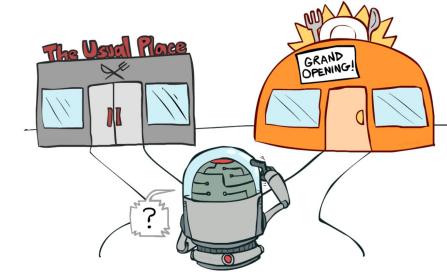
SARSA vs Q-Learning avec ϵ -greedy



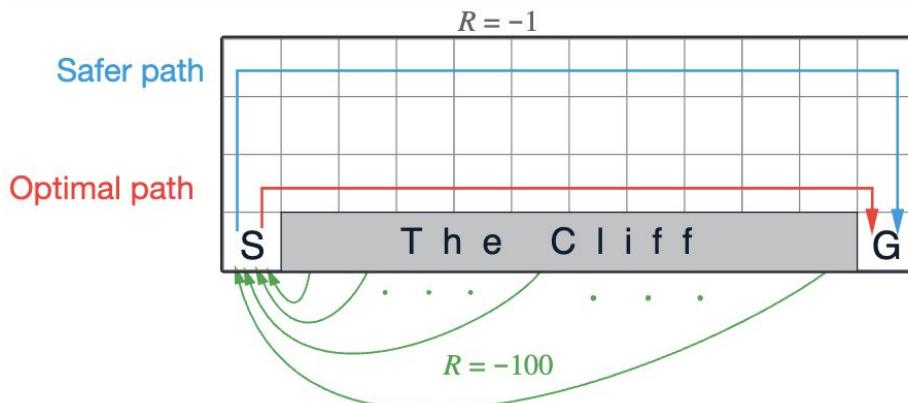
SARSA vs Q-Learning avec ϵ -greedy



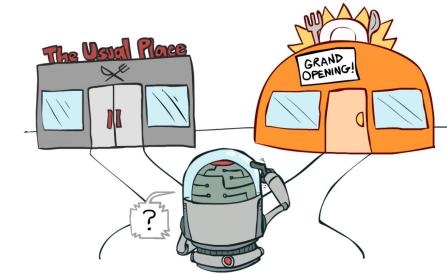
- SARSA prend en compte sa stratégie ϵ -greedy dans l'estimation des Q-valeurs et converge vers une stratégie sûre



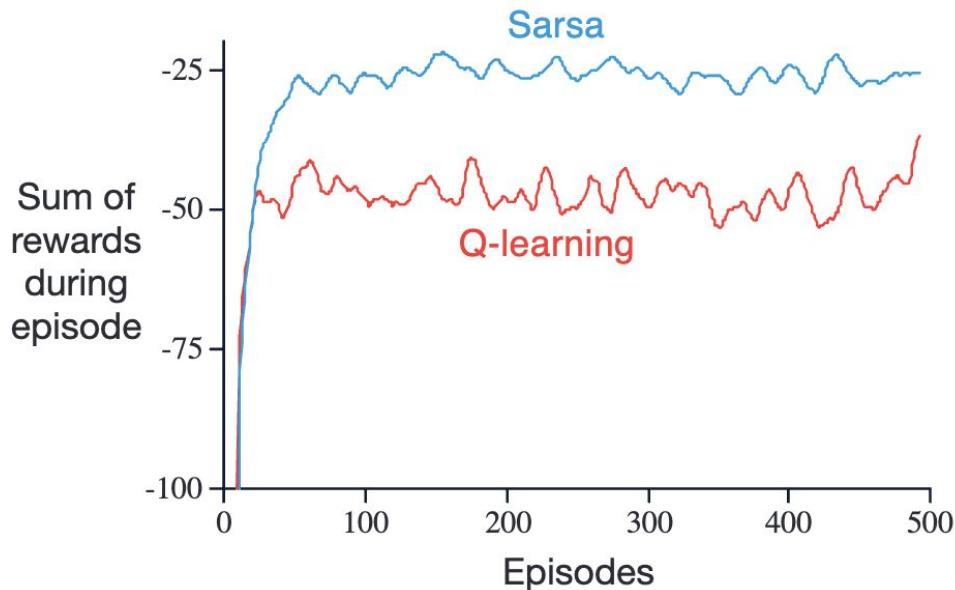
SARSA vs Q-Learning avec ϵ -greedy



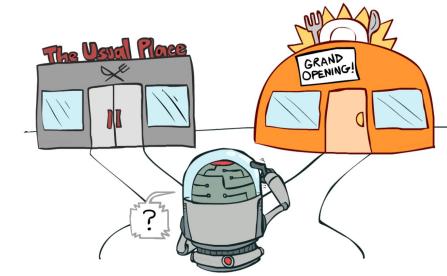
- SARSA prend en compte sa stratégie ϵ -greedy dans l'estimation des Q-valeurs et converge vers une stratégie sûre
- Q-learning ne prend pas en compte la stratégie ϵ -greedy, converge vers les valeurs optimales mais tombe par moment en jouant aléatoirement



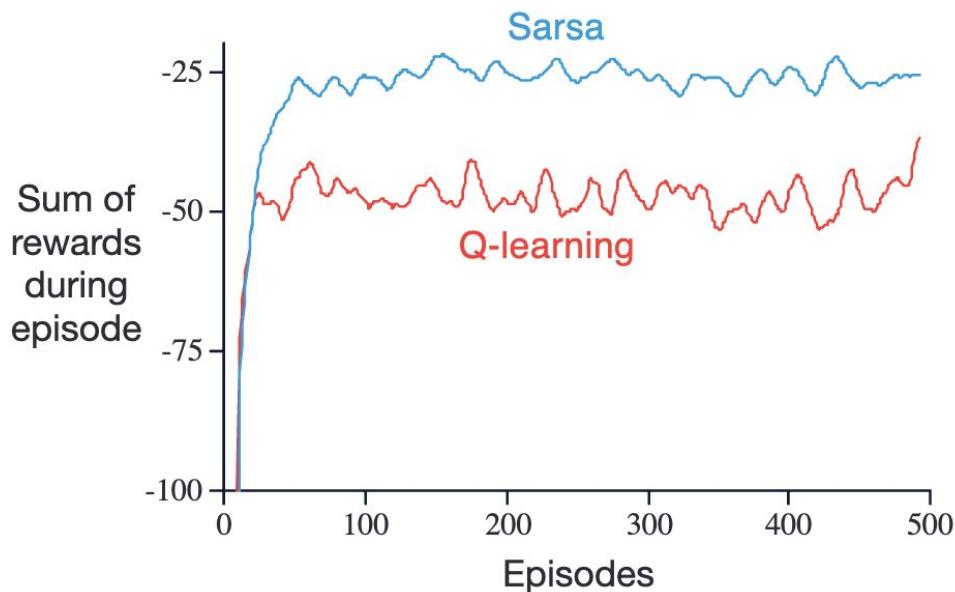
SARSA vs Q-Learning avec ϵ -greedy



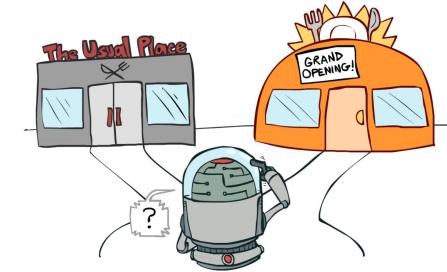
- SARSA prend en compte sa stratégie ϵ -greedy dans l'estimation des Q-valeurs et converge vers une stratégie sûre
- Q-learning ne prend pas en compte la stratégie ϵ -greedy, converge vers les valeurs optimales mais tombe par moment en jouant aléatoirement



SARSA vs Q-Learning avec ϵ -greedy



- SARSA prend en compte sa stratégie ϵ -greedy dans l'estimation des Q-valeurs et converge vers une stratégie sûre
- Q-learning ne prend pas en compte la stratégie ϵ -greedy, converge vers les valeurs optimales mais tombe par moment en jouant aléatoirement



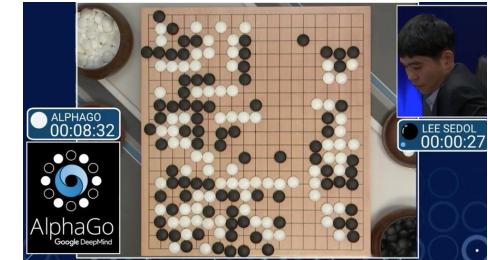
TP: Partie 3

Que fait-on si l'espace des états est trop grand ?

- Dans des **scénarios plus complexes** où:
 - l'espace des **actions** reste discret
 - l'espace des **états est discret mais très grand** ou **continu**
- Stocker toutes les Q-valeurs devient impossible

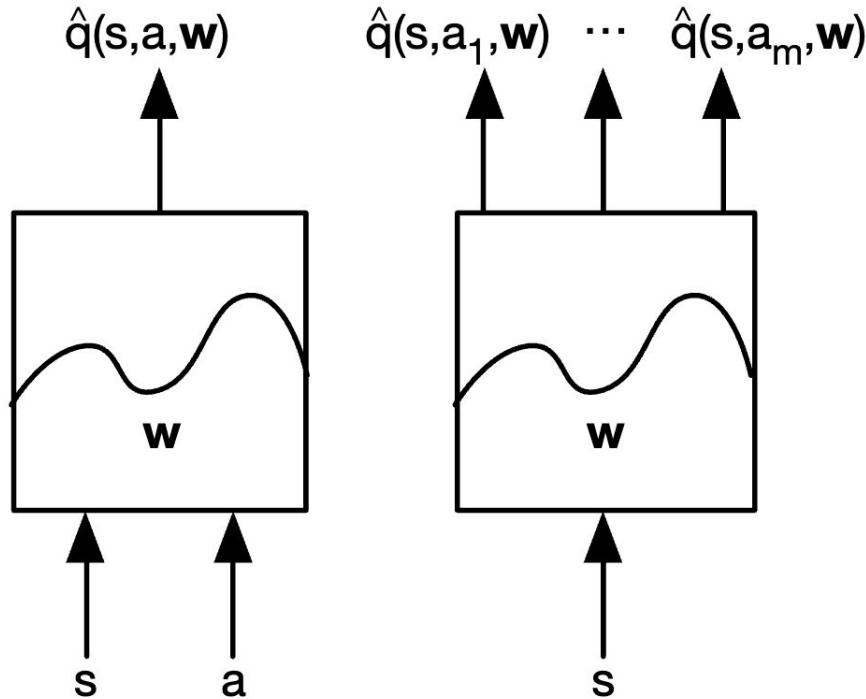
=> Il nous faut une **fonction qui approime** les Q-valeurs
=> Un réseau de neurones par exemple...
=> Généralisation à des états jamais vus

Go: 10^{170} états



Approximer la Q-value: Deep RL

Deux architectures possibles



- On prend s et a en entrée
 - Autant d'appel au réseau que d'actions

- On ne prend que s en entrée
 - Moins de partage d'informations

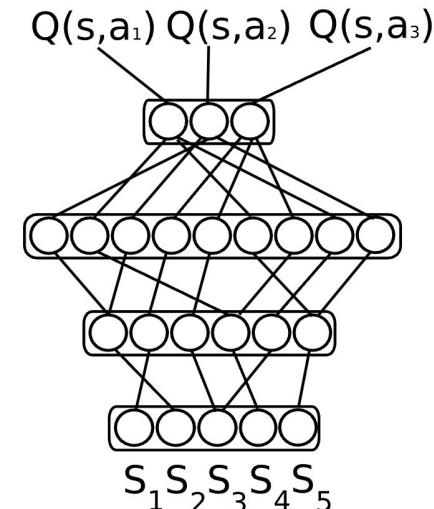
Deep Q-network (*Mnih et al., 2015*)

- On prend s en entrée
- On entraîne notre réseau avec une erreur **MSE sur la TD-error:**

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s') \sim U(D)} \left[(r + \gamma \max_{a'} Q(s', a'; \theta) - Q(s, a; \theta))^2 \right]$$

Observation

Ancienne estimations



=> On applique la méthode de **Q-Learning** pour collecter les transitions

Deep Q-network (*Mnih et al., 2015*)

- L'utilisation de réseaux de neurones avec le Q-Learning est réputé très instable (*Deadly Triad Issue*)
- Le DQN introduit deux tricks pour améliorer la stabilité:

Deep Q-network (*Mnih et al., 2015*)

- L'utilisation de réseaux de neurones avec le Q-Learning est réputé très instable (*Deadly Triad Issue*)
- Le DQN introduit deux tricks pour améliorer la stabilité:
 - Un **replay buffer** contenant les transitions collectées depuis lequel des batchs sont samplés

Deep Q-network (*Mnih et al., 2015*)

- L'utilisation de réseaux de neurones avec le Q-Learning est réputé très instable (*Deadly Triad Issue*)
- Le DQN introduit deux tricks pour améliorer la stabilité:
 - Un **replay buffer** contenant les transitions collectées depuis lequel des batchs sont samplés
 - Un second DQN (**target network**) mis à jour moins fréquemment et utilisé pour stabiliser la TD-error

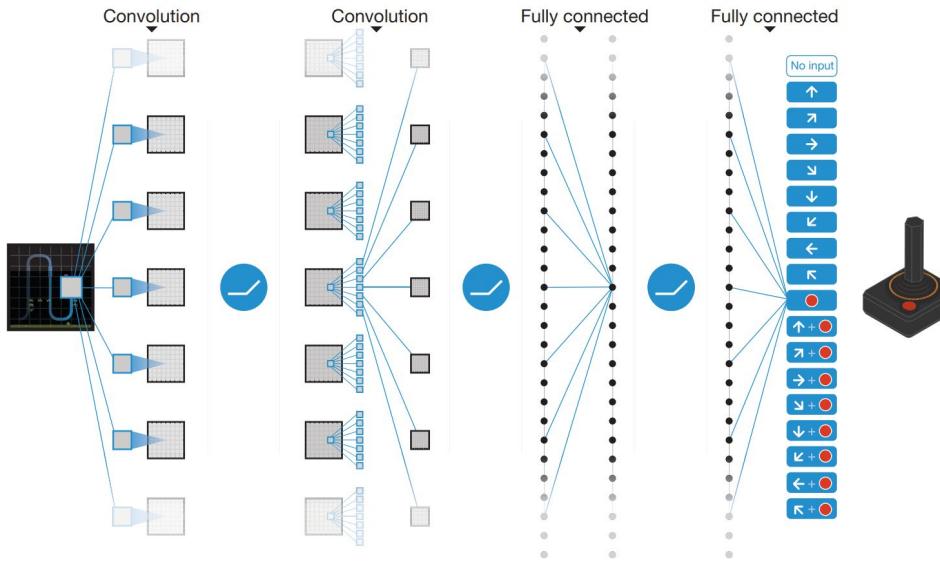
$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s') \sim U(D)} \left[(r + \gamma \max_{a'} Q(s', a'; \phi) - Q(s, a; \theta))^2 \right]$$

Deep Q-network (*Mnih et al., 2015*)

Algorithm 1: Deep Q-Network

- 1: target critic = critic
- 2: **for** n_epochs epochs **do**
- 3: Collect N samples using ϵ -greedy exploration
- 4: Put transitions in replay buffer of size S
- 5: **for** W iterations **do**
- 6: Sample a minibatch from replay buffer
- 7: Update critic
- 8: Every K samples, update target critic
- 9: **end for**
- 10: **end for**

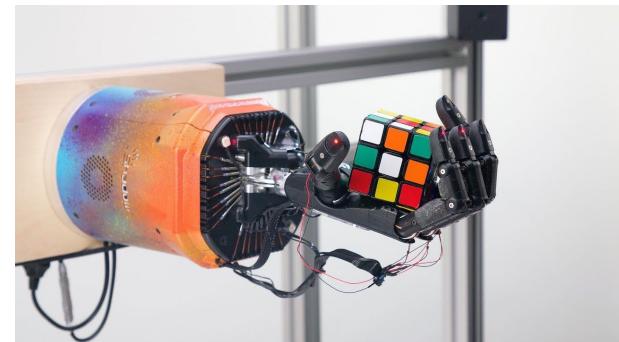
TP: DQN



Que fait-on si l'espace des actions est continu ?

- Dans des scénarios où l'espace des actions est continu:
 - e.g. robotique
- On ne peut plus évaluer la Q-valeur de toutes les actions

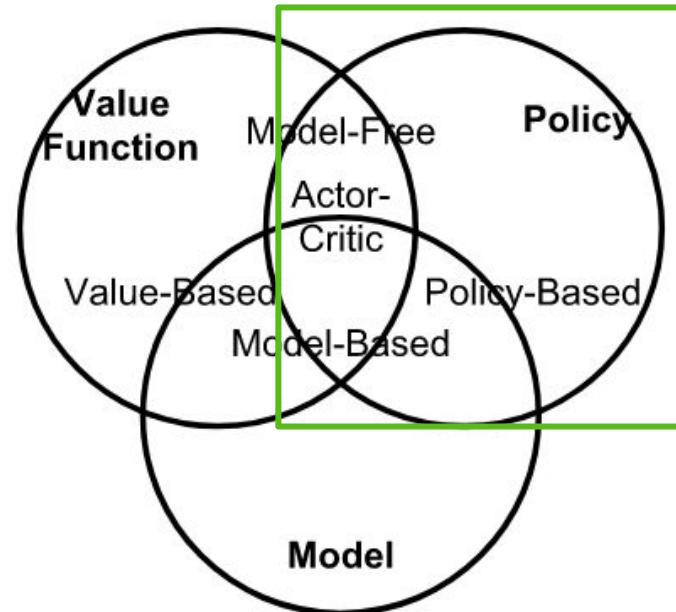
=> On ne peut donc plus avoir une politique qui sélectionne l'action maximisant la Q-valeur



Vers les actions continues: Policy Gradient

Policy-Based

- Policy-Based:
 - On essaie de trouver directement une **fonction de politique**
 - On peut apprendre une valeur également (**Actor-Critic**)



Policy Gradient

- Comment utiliser un réseau de neurones comme politique ?
 - => Policy Gradient Theorem
 - On va plutôt travailler avec des politiques stochastiques
 - On cherche à maximiser la probabilité des actions à fort:
 - Return
 - Q-valeur
 - ...

Policy Gradient

Policy gradient methods maximize the expected total reward by repeatedly estimating the gradient $g := \nabla_{\theta} \mathbb{E} [\sum_{t=0}^{\infty} r_t]$. There are several different related expressions for the policy gradient, which have the form

$$g = \mathbb{E} \left[\sum_{t=0}^{\infty} \Psi_t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right], \quad (1)$$

where Ψ_t may be one of the following:

- | | |
|--|---|
| 1. $\sum_{t=0}^{\infty} r_t$: total reward of the trajectory. | 4. $Q^{\pi}(s_t, a_t)$: state-action value function. |
| 2. $\sum_{t'=t}^{\infty} r_{t'}$: reward following action a_t . | 5. $A^{\pi}(s_t, a_t)$: advantage function. |
| 3. $\sum_{t'=t}^{\infty} r_{t'} - b(s_t)$: baselined version of previous formula. | 6. $r_t + V^{\pi}(s_{t+1}) - V^{\pi}(s_t)$: TD residual. |

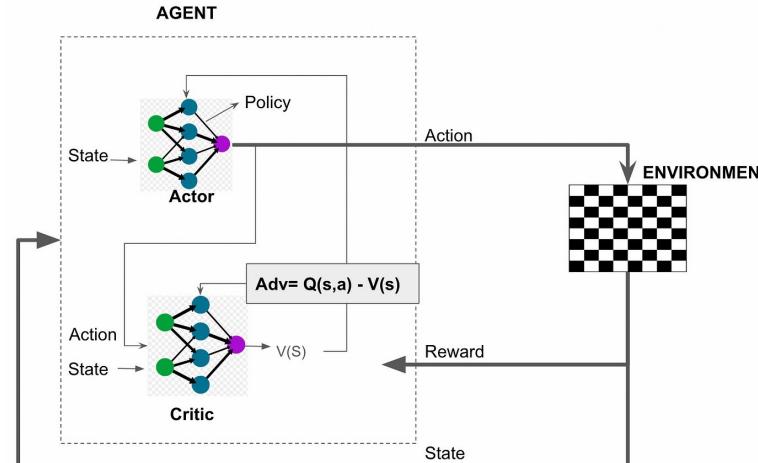
The latter formulas use the definitions

$$V^{\pi}(s_t) := \mathbb{E}_{\substack{s_{t+1:\infty}, \\ a_{t:\infty}}} \left[\sum_{l=0}^{\infty} r_{t+l} \right] \quad Q^{\pi}(s_t, a_t) := \mathbb{E}_{\substack{s_{t+1:\infty}, \\ a_{t+1:\infty}}} \left[\sum_{l=0}^{\infty} r_{t+l} \right] \quad (2)$$

$$A^{\pi}(s_t, a_t) := Q^{\pi}(s_t, a_t) - V^{\pi}(s_t), \quad (\text{Advantage function}). \quad (3)$$

Actor-Critic

- Si ce qu'on cherche à maximiser est une valeur ou Q-valeur:
 - On apprend la politique et cette valeur / Q-valeur (e.g. DQN)



C'est terminé !

<https://forms.gle/rHk1ZnuJZHToqs5R7>