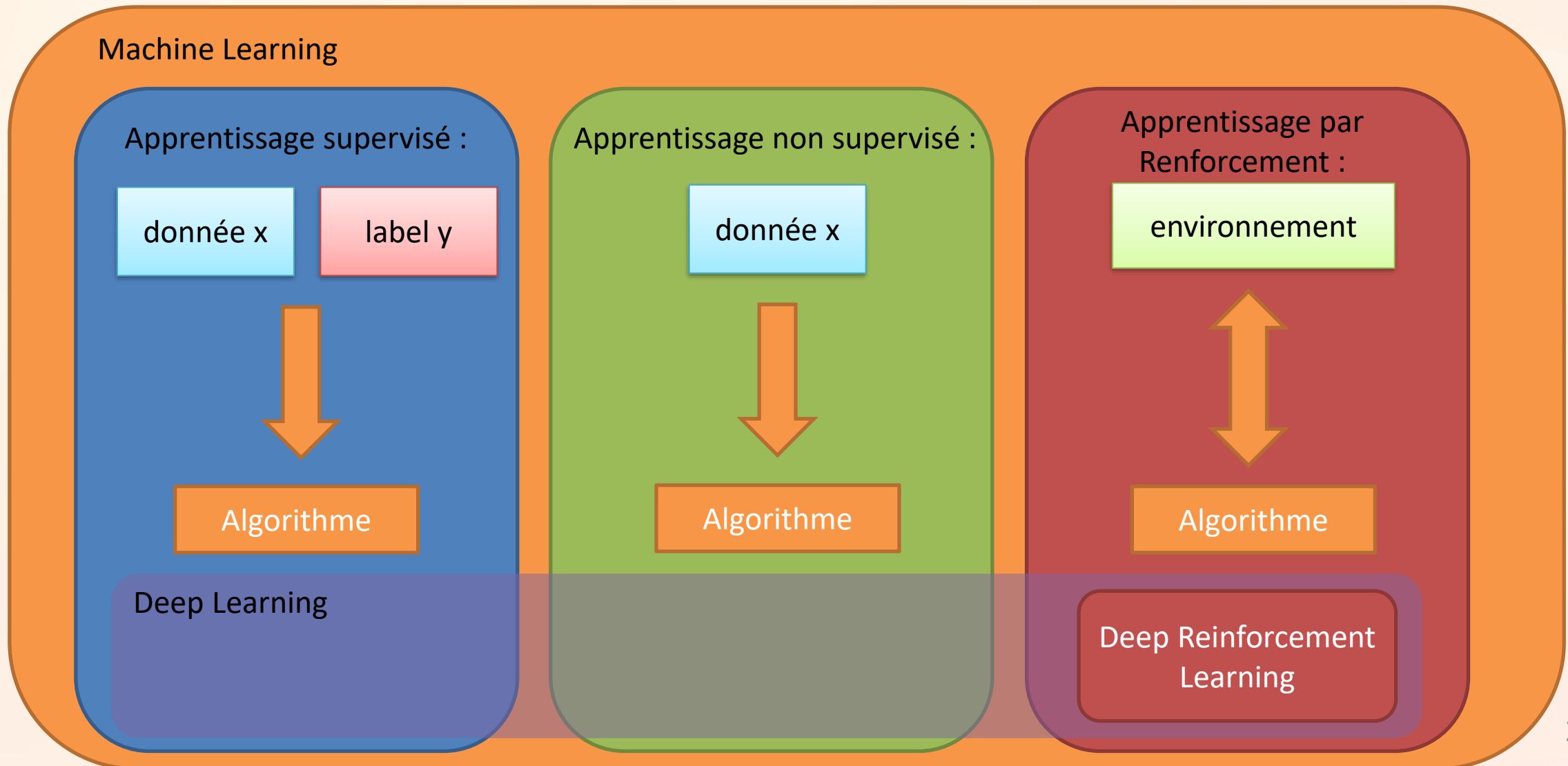




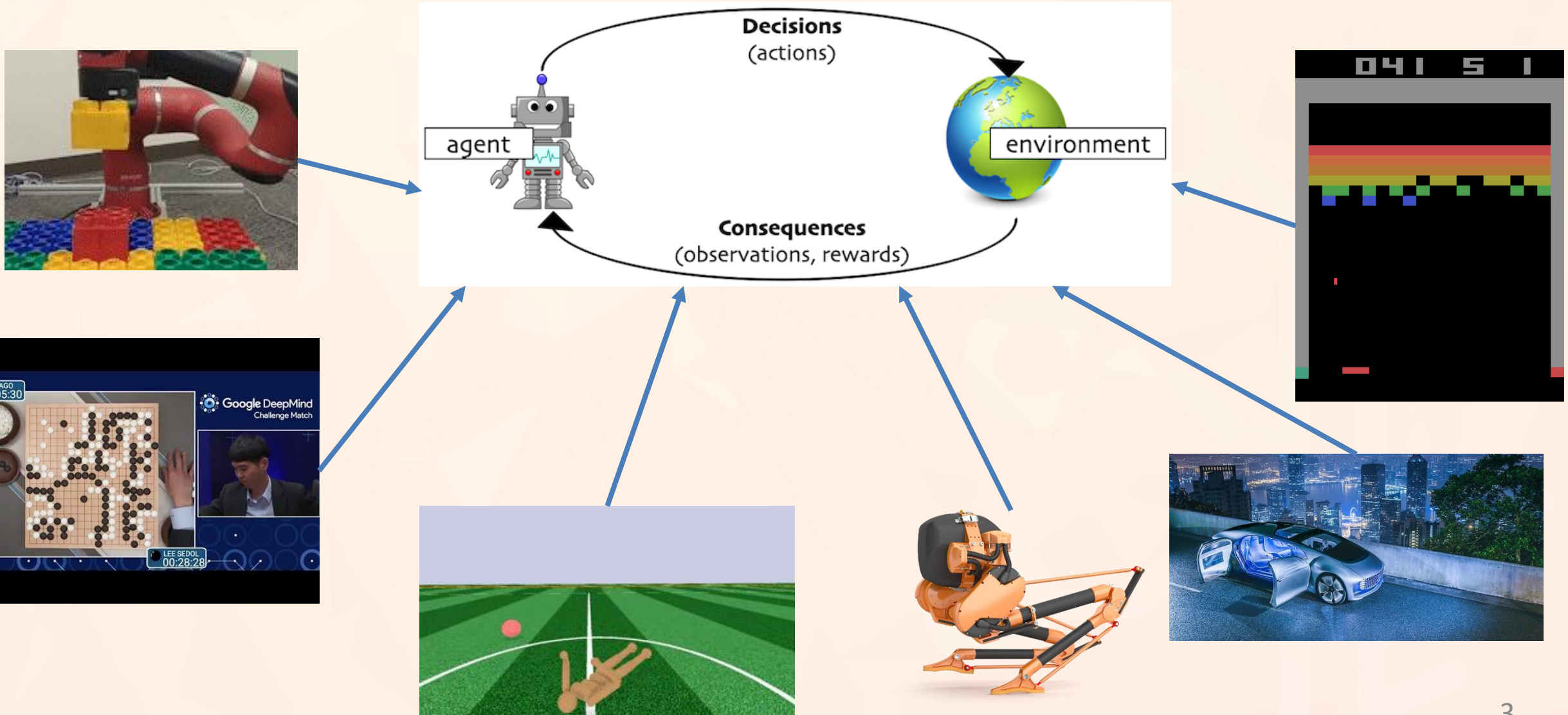
# Apprentissage par Renforcement

## Partie I

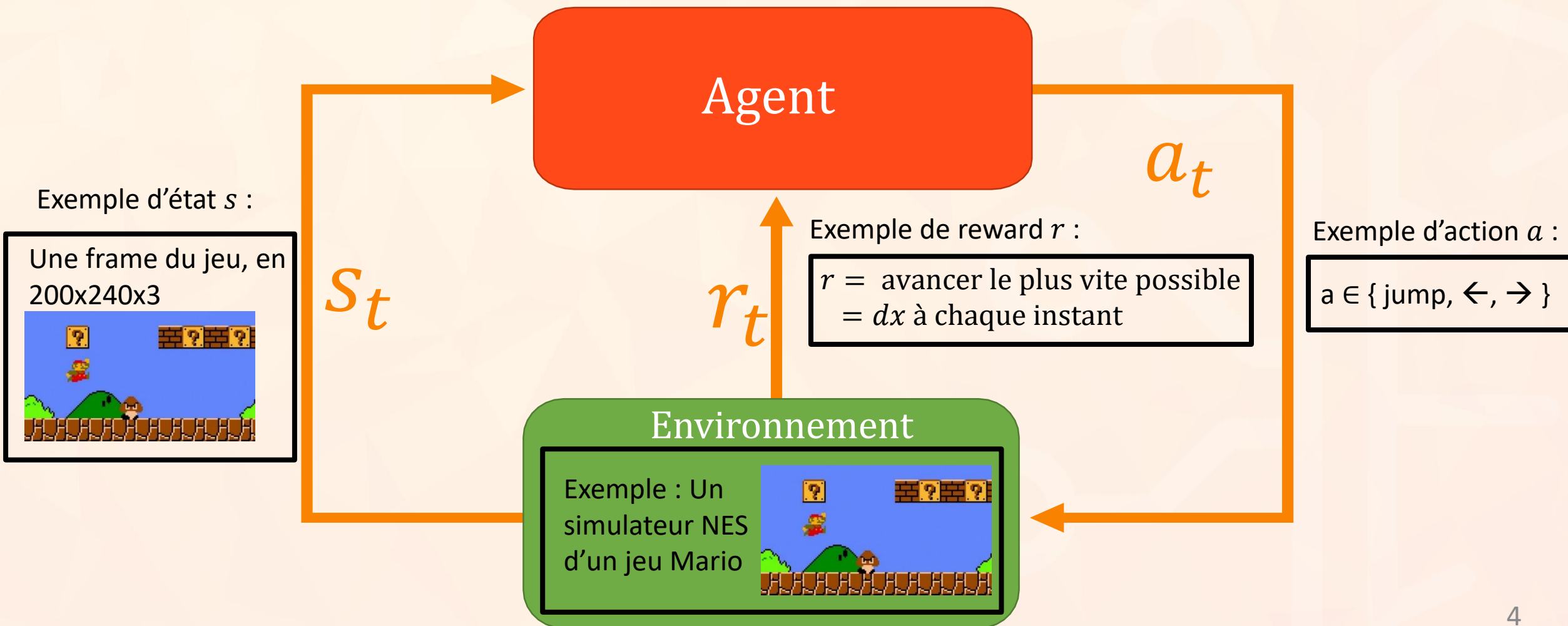
# Les 3 sous domaines du Machine Learning



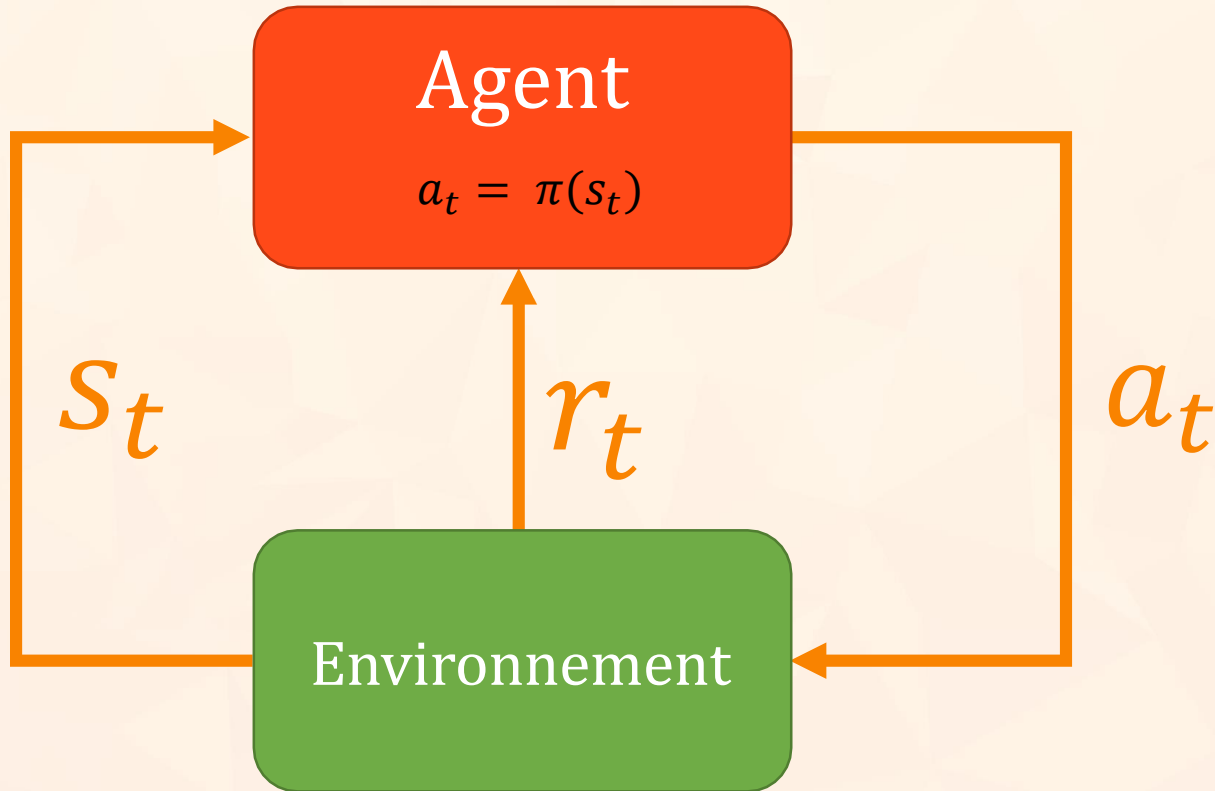
# Applications du RL



# Principe du RL : interaction env./agent



# Principe du RL : la politique



L'agent possède une politique  $\pi$ :

$$\pi(s) = a$$

ou

$$\pi(a|s) = P(A_t = a | S_t = s)$$

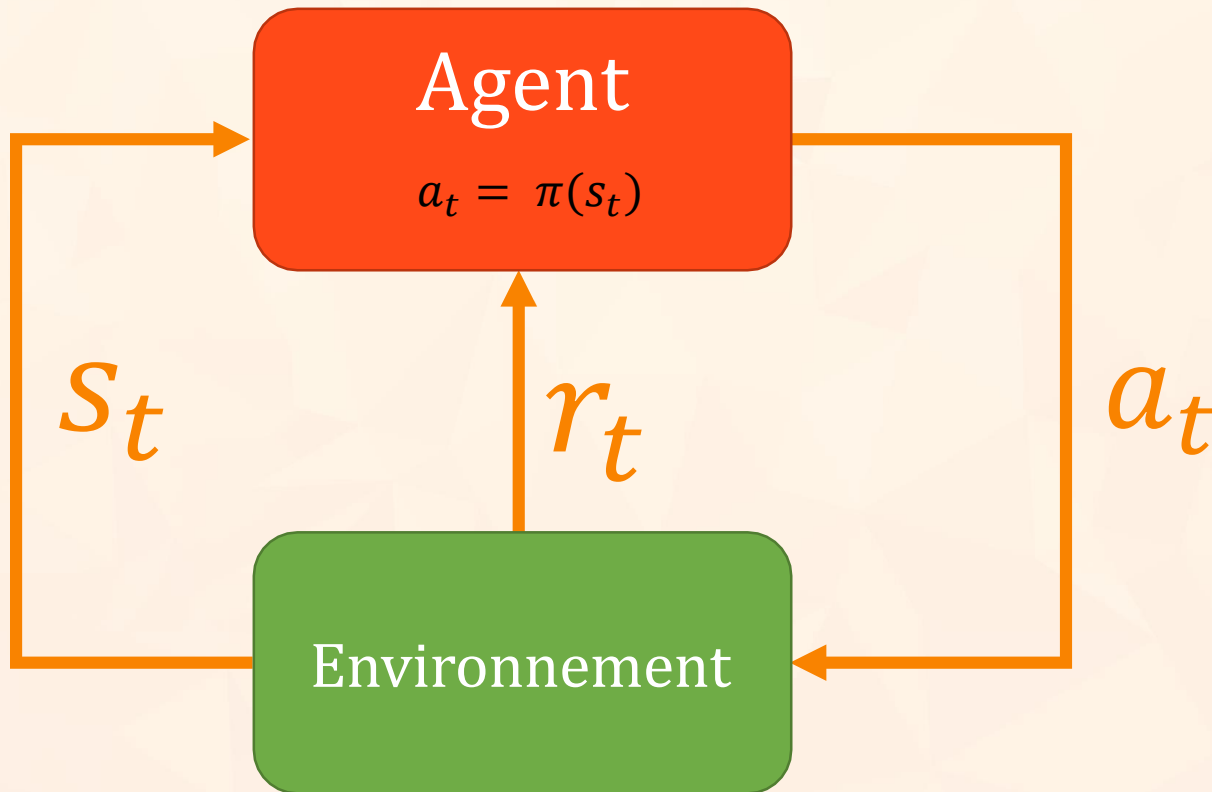
ou

$$\begin{cases} \pi(s) = \text{loi } L \\ a \sim L \end{cases}$$

$$\pi\left(\text{image de Mario dans un niveau Super Mario Bros}\right) = \begin{cases} \rightarrow & \text{à 90\%} \\ \leftarrow & \text{à 5\%} \\ \uparrow & \text{à 5\%} \end{cases}$$



# Principe du RL : transitions et épisodes



Un **épisode**  $\tau$  :

$$\tau = (s_0, a_0, r_0, \underbrace{s_1, a_1, r_1}_{\text{une transition}}, \dots, s_T, a_T, r_T)$$

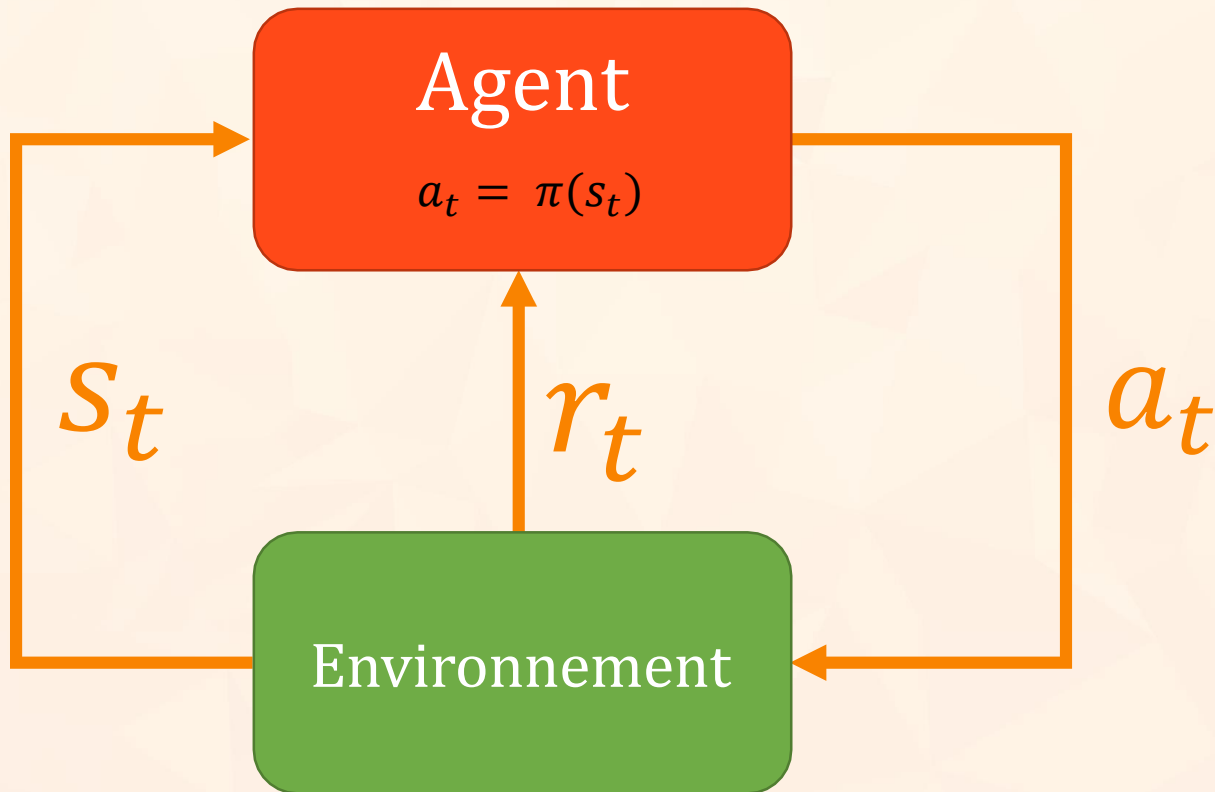
une **transition**

Pour Mario : un épisode = un niveau

L'env. commence dans  $s_0 \in S_{initiaux}$   
Il se termine à  $t = \mathbf{T}$  quand  $s_t \in S_{finaux}$

Remarque : il existe des environnements **non terminaux** !

# Principe du RL : gain futur



But : maximiser le gain futur  $G_t$  :

$$G_t = \sum_{t' \geq t}^T r_{t'}$$

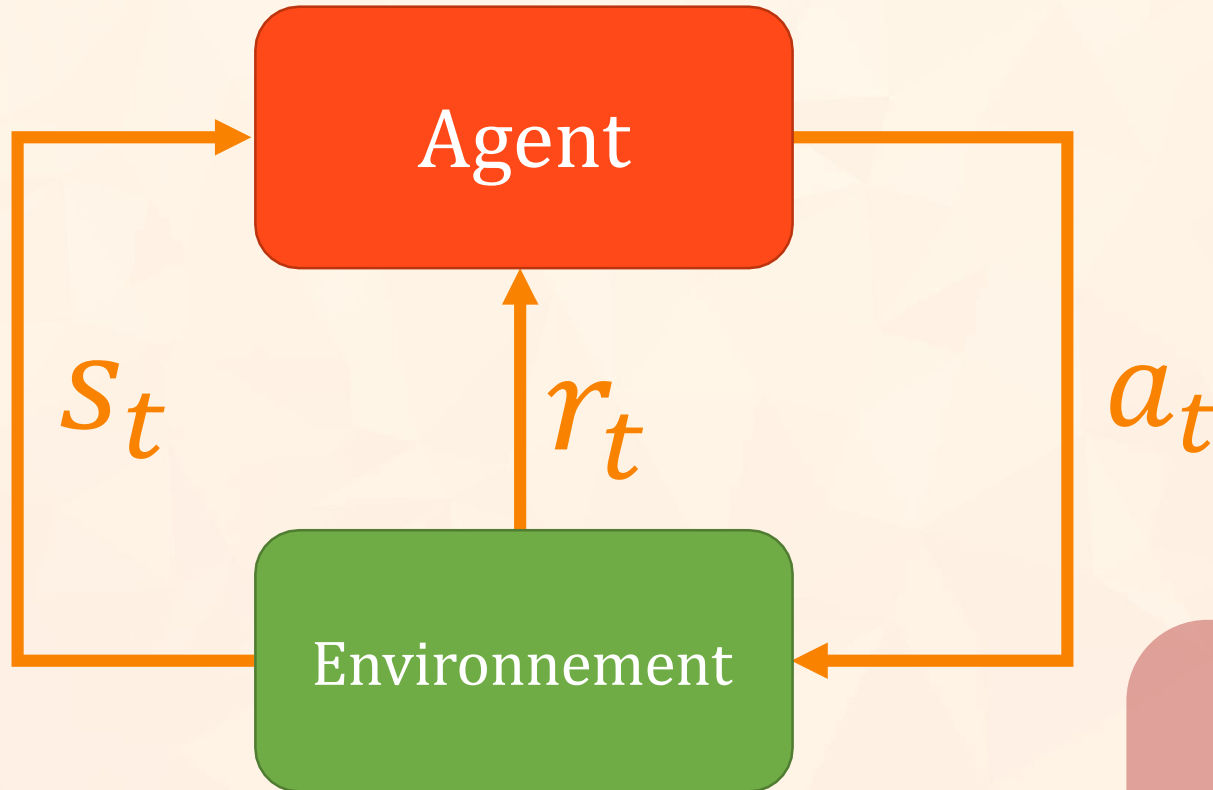
Pour Mario :

$r_t = dx = \text{avancement à } t$

$G_t = \text{avancement futur}$

Objectif : Trouver  $\pi^* = \underset{\pi}{\operatorname{argmax}} E[G_t | \pi]$

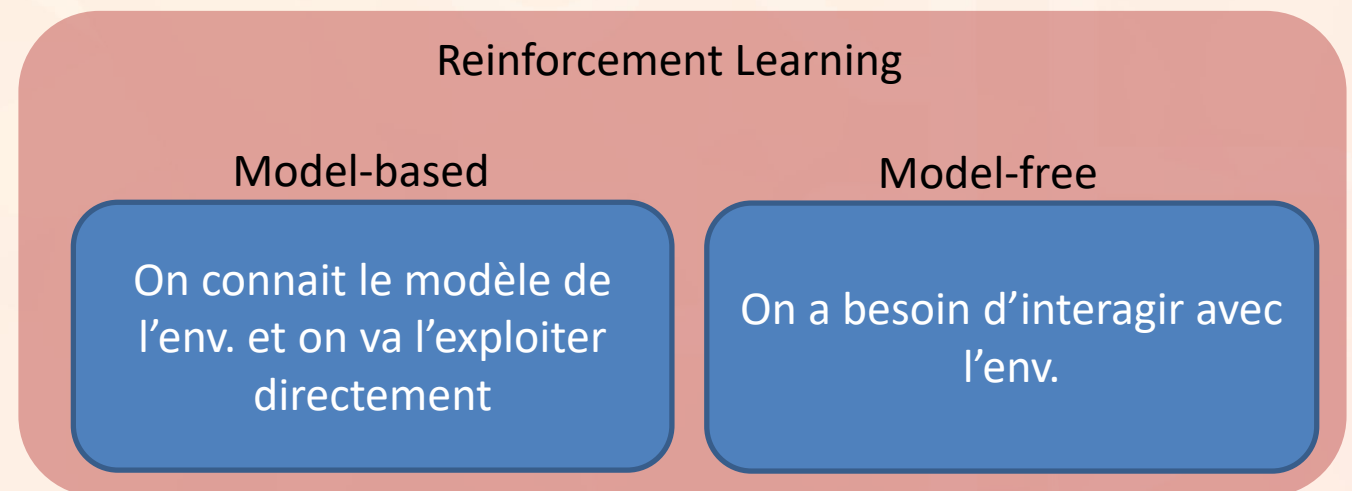
# Principe du RL : l'environnement



L'environnement est régi par des lois probabilistes qu'on appelle le modèle :

$P_{s \rightarrow s'}^a = P(S_{t+1} = s' | S_t = s, A_t = a)$   
(probabilité que l'env. arrive dans  $s'$  si on fait  $a$  dans  $s$ )

$R_s^a = E[R_t | S_t = s, A_t = a]$   
(récompense moyenne si on fait  $a$  dans  $s$ )

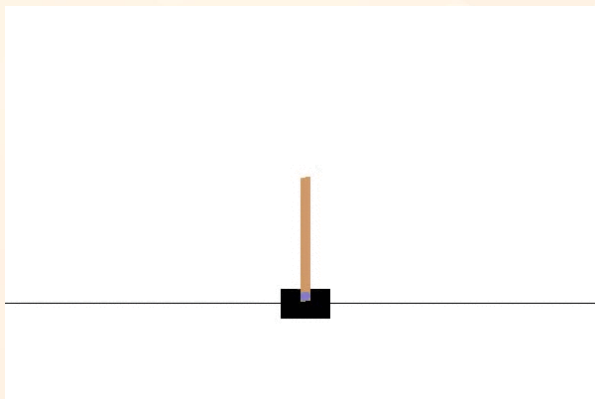




# Environnements de RL



## Exemple 1 : L'environnement CartPole




État :  $s = (\text{position et vitesse en } x \text{ et en } \theta)$

Action :  $a \in \{\leftarrow, \rightarrow\}$

Reward :  $r = +1$

## Exemple 2 : Jeu vidéo type Mario



État :  $s =$  

Action :  $a \in \{\text{jump}, \leftarrow, \rightarrow\}$

Reward :  $r = \frac{dx}{dt}$


# Environnements de RL



## Exemple 3 : Échecs (face à un adversaire donné)



État :

$s =$   ou (1.e4e5 2.Nc3Nf6 3.f4d5)

Action :

$a =$  prochain coup

Reward :

$r = +1$  si victoire,  $-1$  si défaite,  $0$  sinon

## Exemple 4 : Robotique



État :

$s =$  capteurs de pression/position du robot

Action :

$a =$  ordres pour chaque muscle robotique

Reward :

$r =$  récompense pour se tenir droit, tenir un objet, etc...

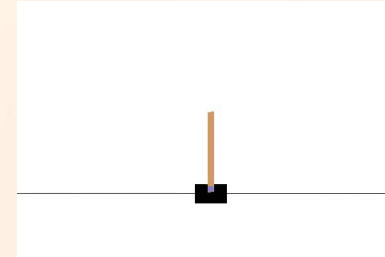
# Plusieurs types d'environnements



L'environnement peut être:

- **déterministe ou bien stochastique** (= inclure de l'aléatoire)

L'objectif est de trouver  $\pi$  qui maximise  $E[G_t | \pi]$



- **terminal ou non** :  $T$  peut valoir  $+\infty$

Pour éviter que  $G_t = \sum_{t' \geq t}^{+\infty} r_{t'}$  ne diverge, on introduit la notion de Discount Factor  $\gamma \in [0,1[$ .  $\gamma = 0,99$  par exemple.

$$G_t = \sum_{t' \geq t}^T \gamma^{t'-t} r_{t'} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$$

- **Markovien ou non** :

Propriété de Markov : l'état actuel contient toute l'information des états précédents



# Plusieurs types d'environnements



L'environnement peut être :

- **parfaitement observable ou non :**

On parlera alors d'observations plutôt que d'état :  $o_t = x(s_t)$



- **Model-based ou Model-free:**

Model-based = accès au modèle  $P_{s \rightarrow s'}^a$ , et  $R_s^a$  (cas des échecs et autres jeux adversariaux)

Model-free = modèle inaccessible/trop complexe, nécessité d'interagir avec l'environnement (Mario, CartPole, simulateurs physiques)

## Reinforcement Learning

### Model-based

On connaît le modèle de l'env. et on va l'exploiter directement

### Model-free

On a besoin d'interagir avec l'env.

# Plusieurs types d'environnements



Environnement	Déterministe Ou Stochastique	Terminal ?	Observabilité	Markovien ?	Model-based ou Model-free
Mario	Déterministe	Oui	Partielle	Non	Model-free
Échecs (en connaissant $\pi_{adversaire}$ )	Stochastique si $\pi_{adversaire}$ l'est	Non	Totale	Oui	Model-based
CartPole (pendule inversé)	Déterministe	Non (mais on l'arrête à 500 steps)	Totale	Oui	Model-free
Environnement robotique réel (non simulé)	Stochastique	Non	Partielle	Dépend de la qualité des capteurs	Model-free



Certains env. sont également **non stationnaires** (le modèle change au cours du temps) et nécessitent d'y appliquer des algorithmes qui s'adaptent en permanence.





# Environnement Shaping

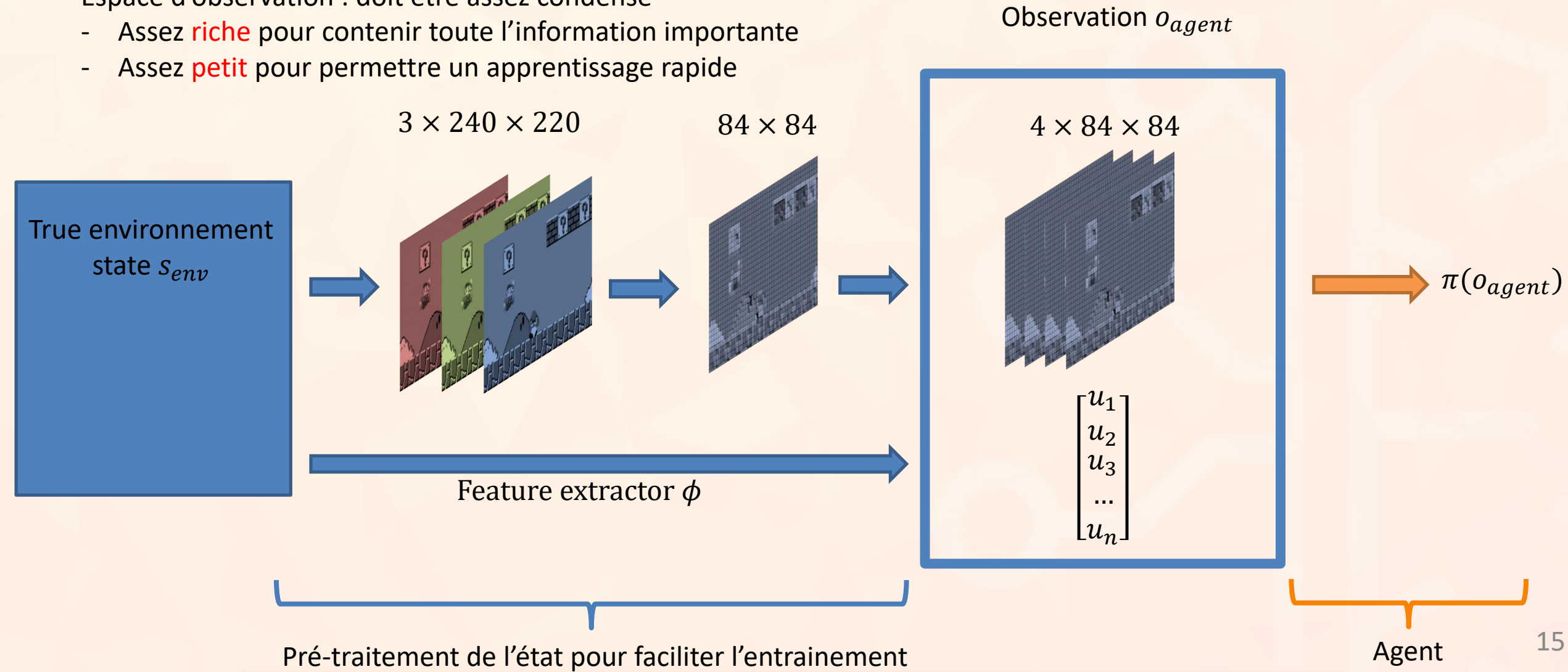


# Pré-traitement de l'environnement : états



Espace d'observation : doit être assez condensé

- Assez **riche** pour contenir toute l'information importante
- Assez **petit** pour permettre un apprentissage rapide



# Pré-traitement de l'env. : reward




La reward comme **but** : L'agent maximise la reward, il faut donc la design pour qu'elle corresponde à votre but

La reward comme **signal** : Les valeurs relatives de la reward aident l'agent à apprendre ce qu'il doit faire

Reward sparse : reward rarement non nulle, difficile pour l'agent d'apprendre

Reward dense : reward non uniforme, permet d'aider l'agent à accomplir des sous-objectifs

Environnement	Reward sparse	Reward dense
Mario	+1 quand niveau réussi	$\frac{dx}{dt}$
Échecs	+1/-1 en fin de partie	$n$ pour chaque pièce prise, +/-100 en fin de partie
Labyrinthe	+1 à la sortie	$\frac{d}{dt}$ (proximité avec le but)

 Reward shaping



Mal définir la reward peut amener à des comportements inattendus



# Pré-traitement de l'env. : action



Réduire l'espace d'action le plus possible :



Action :

$a \in \{ \text{jump}, \leftarrow, \rightarrow, \text{jump} + \rightarrow, \text{jump} + \leftarrow, \text{fireball}, \text{fireball} + \rightarrow, \dots \}$

Action « suffisantes » :

$a \in \{ \rightarrow, \text{jump} + \rightarrow \}$

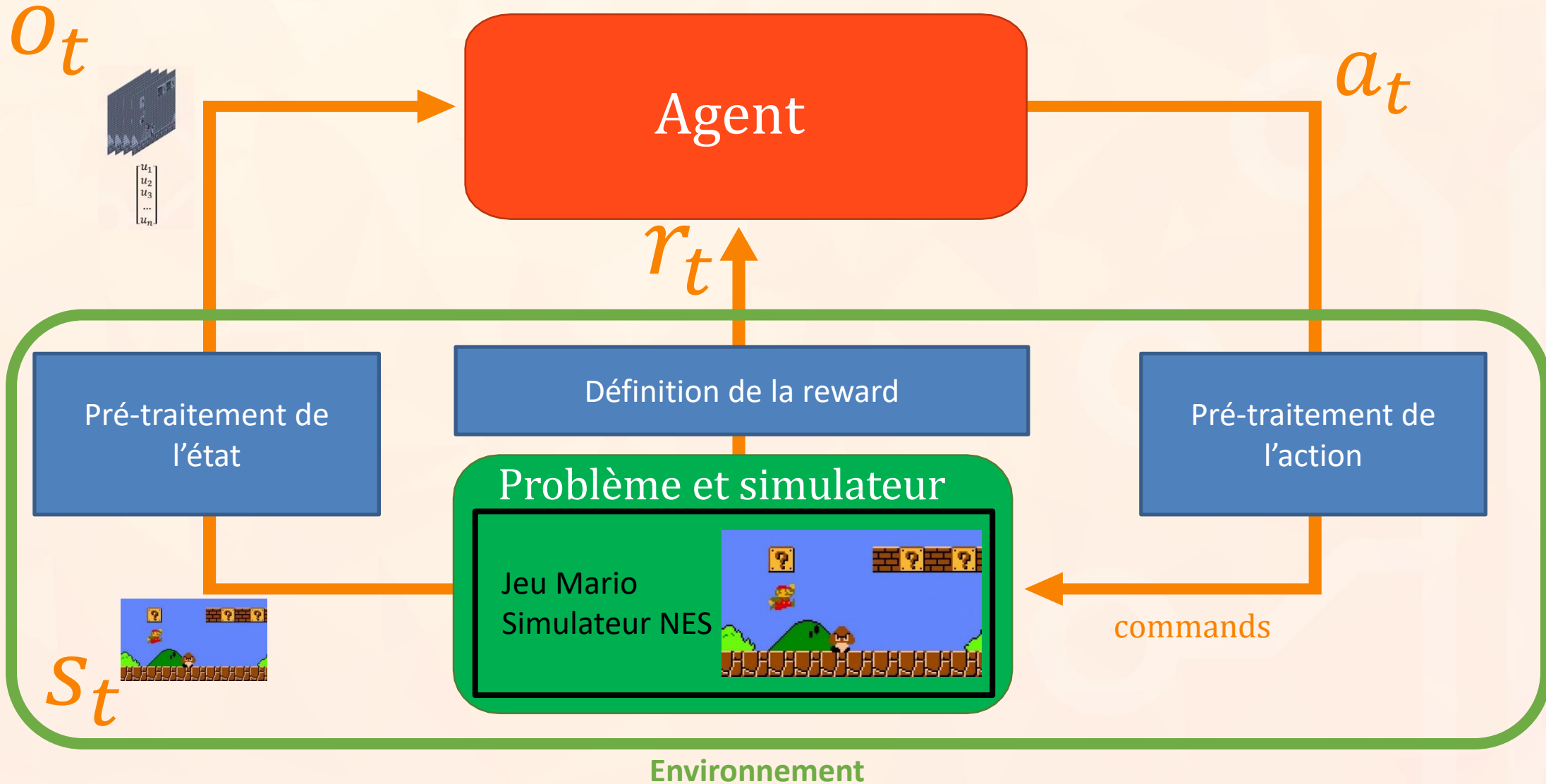
Créer des actions automatisées

Exemple :  $a = \text{« kill enemy »}$

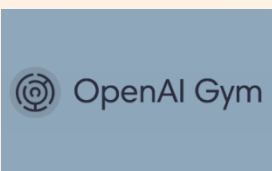
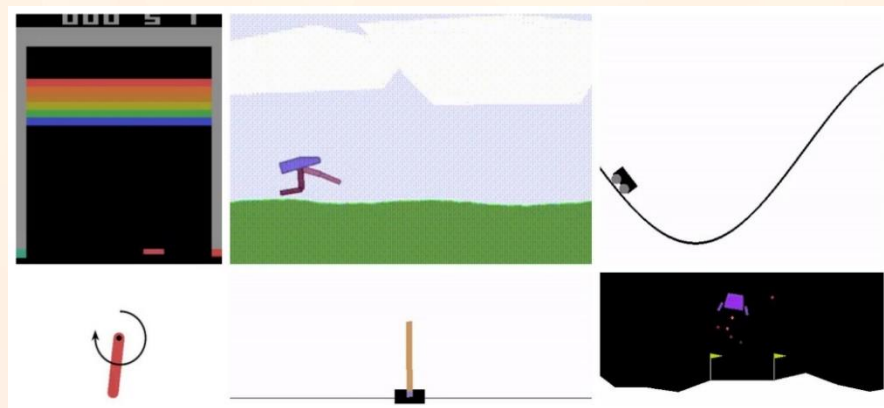
Prendre l'action  $a$  a pour effet de générer une suite de commandes censé tuer le prochain ennemi dans le jeu.

Remarque : on peut même faire apprendre un « sous-agent » à apprendre à bien réaliser la macro-action  $a$ , et dans ce cas, on parle *d'apprentissage par renforcement hiérarchique*.

# Pré-traitement de l'env.



# Definissez vos propres environnements



**Gym** : Bibliothèque pour **définir** vos environnements ou **en utiliser** des déjà implémentés.

**OceanEnv** : exemple d'environnement custom avec Gym :

**État** :  $s \in \{0, 1, \dots, 10\}$ , représente la distance à la rive

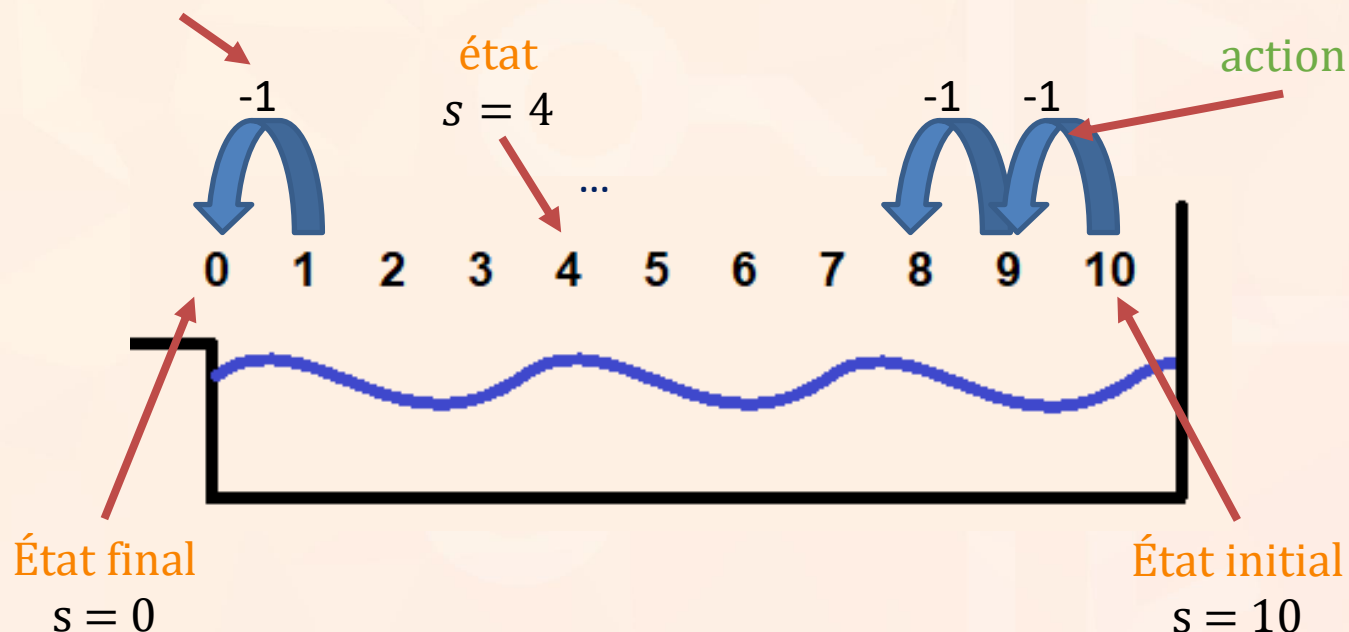
**Action** :  $a \in \{\text{s'éloigner, se rapprocher}\}$

**Reward** :  $r = -1$ , à chaque  $t$  (punition tant que la rive n'est pas atteinte)

État initial :  $s_0 = 10$

État terminal :  $s_T = 0$

reward  $r = -1$







Comment apprendre ?



# Les valeurs d'état et d'action



Valeur d'état :

$$v_{\pi}(s) = E[G_t | S_t = s, \pi]$$

à quel point mon état  $s$  est bon avec la politique  $\pi$

Valeur d'action :

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

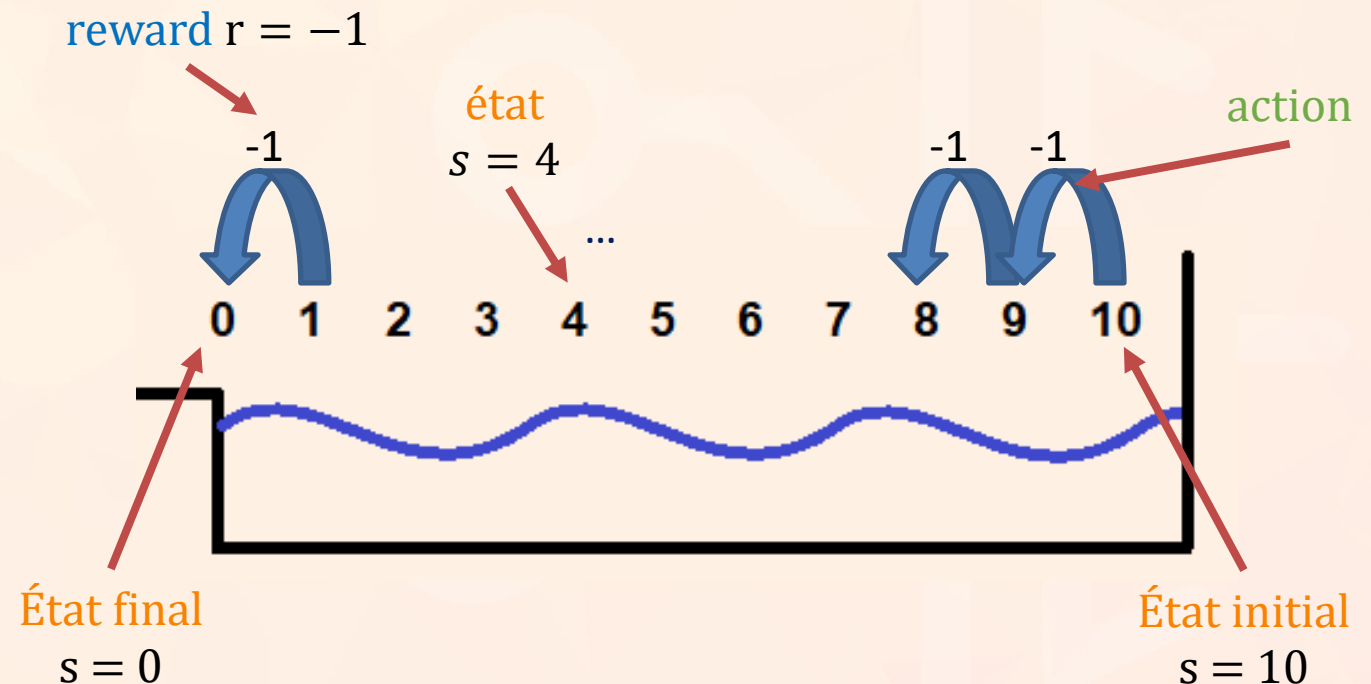
à quel point mon action  $a$  est bonne dans l'état  $s$  et avec la politique  $\pi$

$$v_{\pi_{se \text{ rapprocher de la rive}}}(s) = ?$$

$$v_{\pi_{s' \text{ éloigner de la rive}}}(s) = ?$$

$$q_{\pi_{se \text{ rapprocher de la rive}}}(s, s' \text{ approcher}) = ?$$

$$q_{\pi_{se \text{ rapprocher de la rive}}}(s, s' \text{ éloigner}) = ?$$



# Les valeurs d'état et d'action



Valeur d'état :

$$v_{\pi}(s) = E[G_t | S_t = s, \pi]$$

à quel point mon état  $s$  est bon avec la politique  $\pi$

Valeur d'action :

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

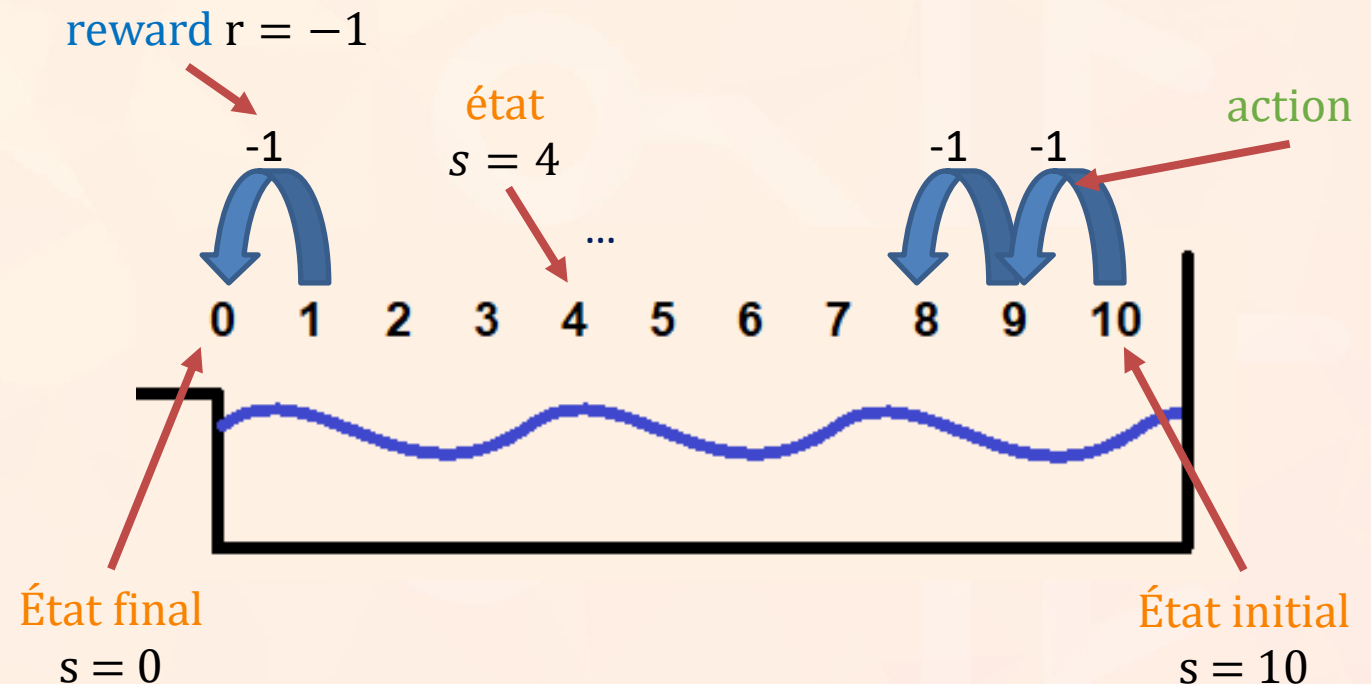
à quel point mon action  $a$  est bonne dans l'état  $s$  et avec la politique  $\pi$

$$v_{\pi_{se \text{ rapprocher de la rive}}}(s) = -s$$

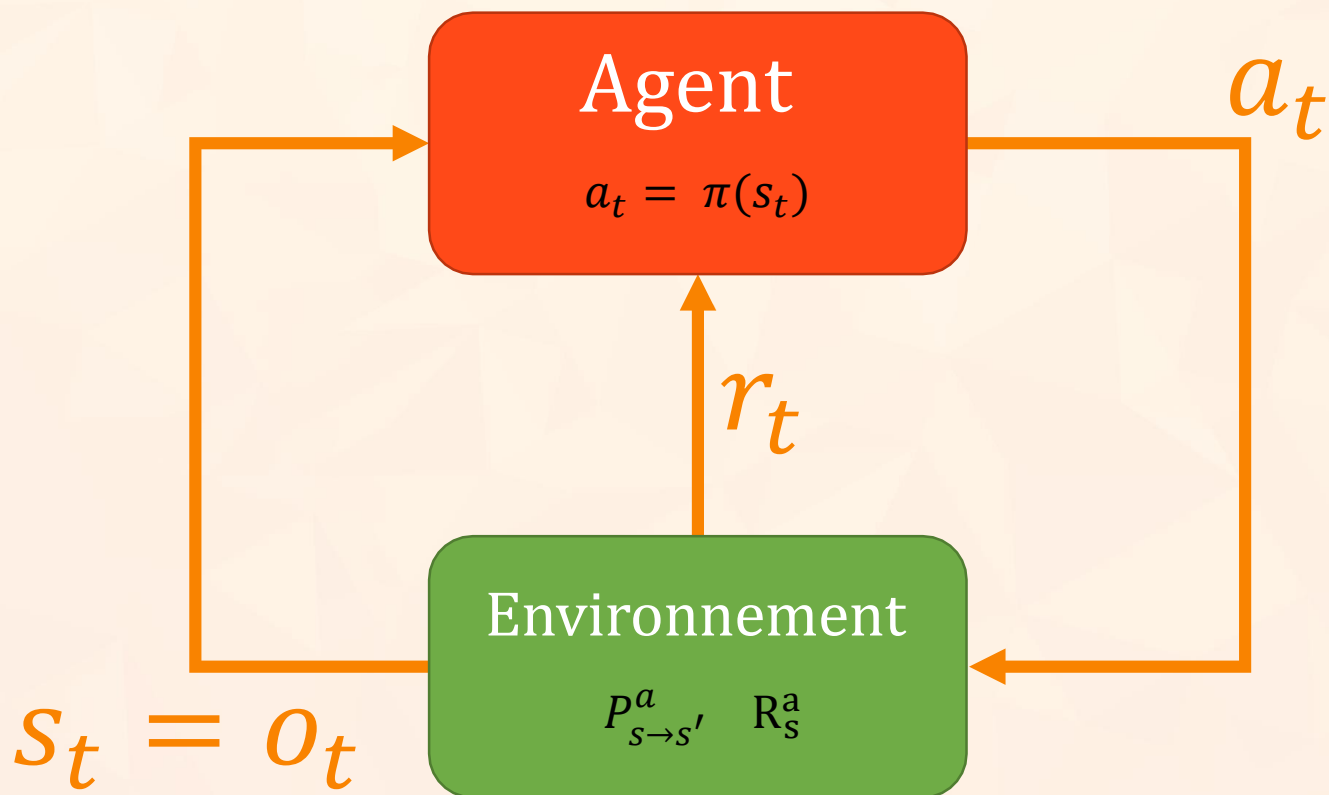
$$v_{\pi_{s' \text{ éloigner de la rive}}}(s) = -\infty$$

$$q_{\pi_{se \text{ rapprocher de la rive}}}(s, s' \text{ approcher}) = -s$$

$$q_{\pi_{se \text{ rapprocher de la rive}}}(s, s' \text{ éloigner}) = -s - 2$$



# Cadre du RL : Markovian Decision Process



On se place dans le cadre d'un **MDP**:

- Environnement Markovien
- Parfaitement observable

$$MDP = (S, A, P_{s \rightarrow s'}^a, R_s^a)$$

Politique :

$$\pi(a|s) = P(A_t = a | S_t = s)$$

Valeur d'état :

$$v_\pi(s) = E[G_t | S_t = s, \pi]$$

Valeur d'action :

$$q_\pi(s, a) = E[G_t | S_t = s, A_t = a, \pi]$$

But : Trouver  $\pi$  qui maximise le gain futur  $G_t$  :

$$G_t = \sum_{t'=t}^T \gamma^{t'-t} r_{t'}$$

# Division du problème du RL en 2 sous problèmes



## Prediction Problem :

**Estimer** les valeurs de  $v_\pi$  et  $q_\pi$

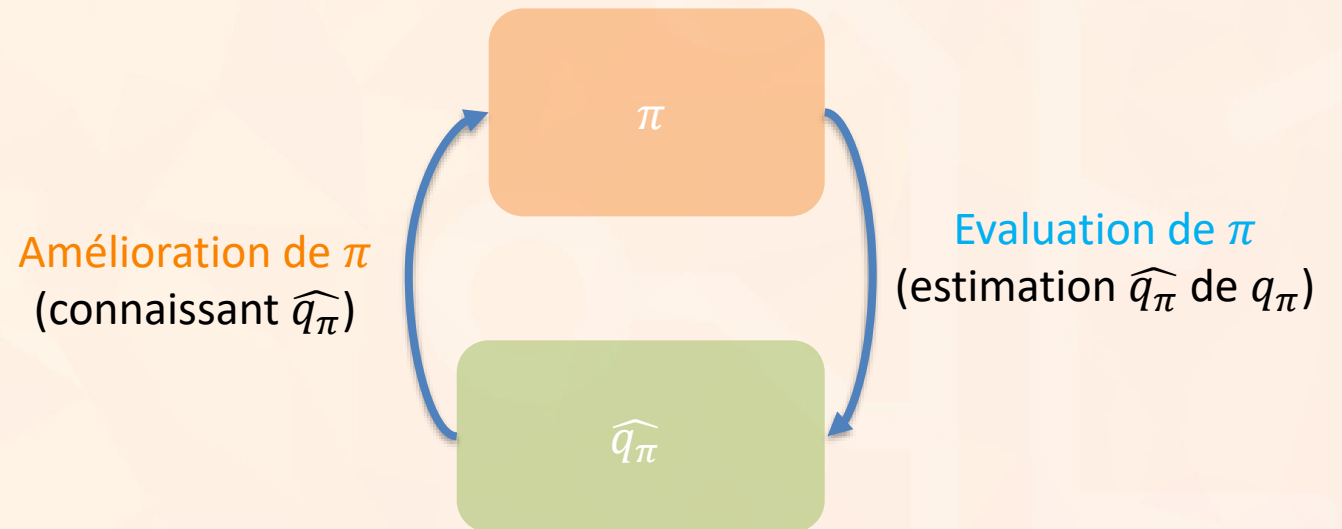
Estimateurs :  $\hat{v}_\pi$  et  $\hat{q}_\pi$

## Control Problem

**Améliorer**  $\pi$

Connaître  $\hat{q}_\pi(s, a)$  permet de choisir les meilleures actions

Generalized Policy Iteration :



# Sous catégories du RL



## Reinforcement Learning

### Model-based

Dynammic  
Programming

### Model-free

Monte  
Carlo  
methods

TD-Learning  
methods



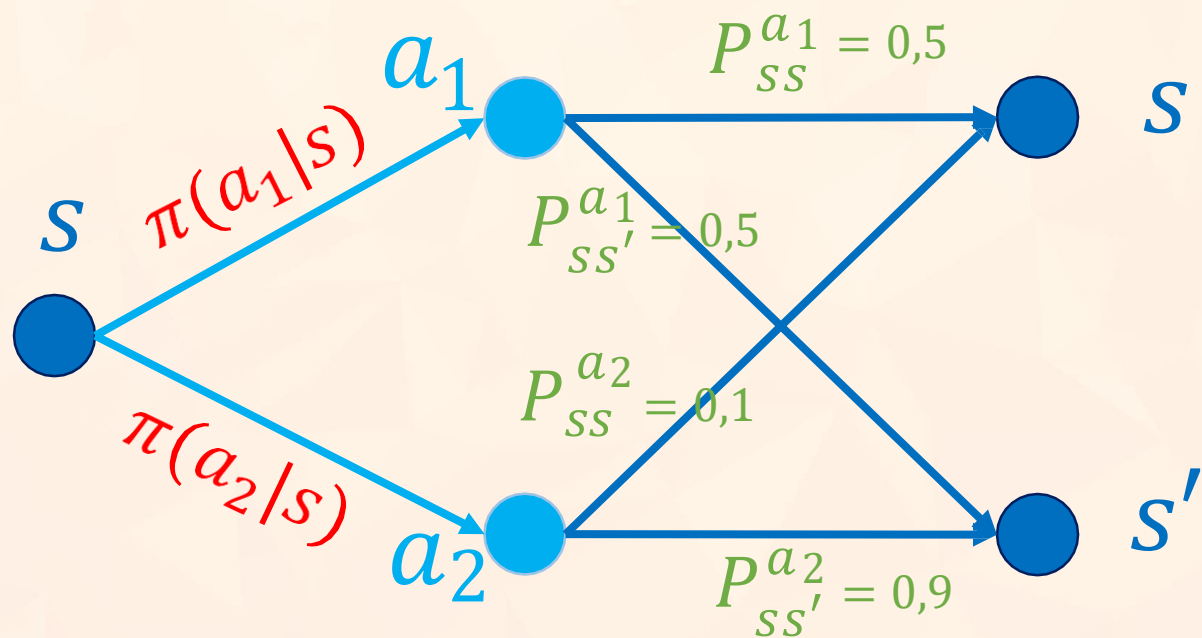
# Model-Based Reinforcement Learning



# Dynamic Programming : Prediction Problem



On connait le modèle  $(P_{s \rightarrow s'}^a, R_s^a)$  et la politique  $\pi$ . On cherche tout d'abord à estimer  $v_\pi(s)$  et  $q_\pi(s, a)$  (**Prediction Problem**).



$$v_\pi(s) = \sum_{a \in A} \pi(a|s) q_\pi(s, a)$$

$$q_\pi(s, a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s')$$

**Bootstrapping** : utiliser des valeurs d'états/d'actions pour calculer d'autres valeurs d'états/d'actions

Remarque : ici les actions et les états sont **discrets** et **peu nombreux**.

# Équations de Ford-Bellman dynamiques



$$v_{\pi}(s) = \sum_{a \in A} \pi(a|s) (R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_{\pi}(s'))$$

$$q_{\pi}(s, a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \left( \sum_{a' \in A} \pi(a'|s') q_{\pi}(s', a') \right)$$

En notant  $V$  le vecteur des valeurs d'états  $V = (v_{\pi}(s))_{s \in S}$  on obtient une équation de la forme  $V = f(V)$ .  
Méthode de convergence itérative :

$$\begin{cases} V_0 \text{ arbitraire} \\ V_{k+1} = f(V_k) \end{cases}$$

# Iterative Policy Evaluation



Algorithme : **Iterative Policy Evaluation**

Algorithme utilisé en Dynamic Programming répondant au **Prediction Problem**

```
Input  $\pi$ , the policy to be evaluated
Initialize an array  $V(s) = 0$ , for all  $s \in \mathcal{S}^+$ 
Repeat
   $\Delta \leftarrow 0$ 
  For each  $s \in \mathcal{S}$ :
     $v \leftarrow V(s)$ 
     $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$ 
     $\Delta \leftarrow \max(\Delta, |v - V(s)|)$ 
until  $\Delta < \theta$  (a small positive number)
Output  $V \approx v_\pi$ 
```

$$v_\pi(s) = \sum_{a \in A} \pi(a|s) (R_s^a + \gamma \sum_{a \in A} P_{ss'}^a v_\pi(s))$$

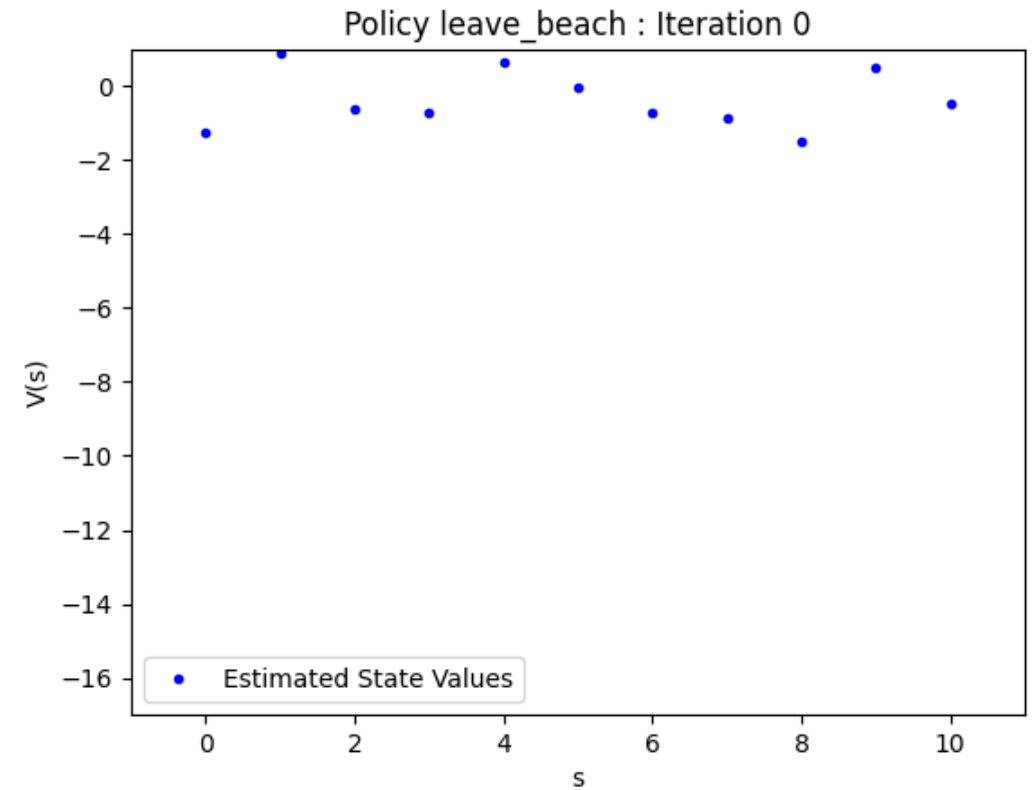
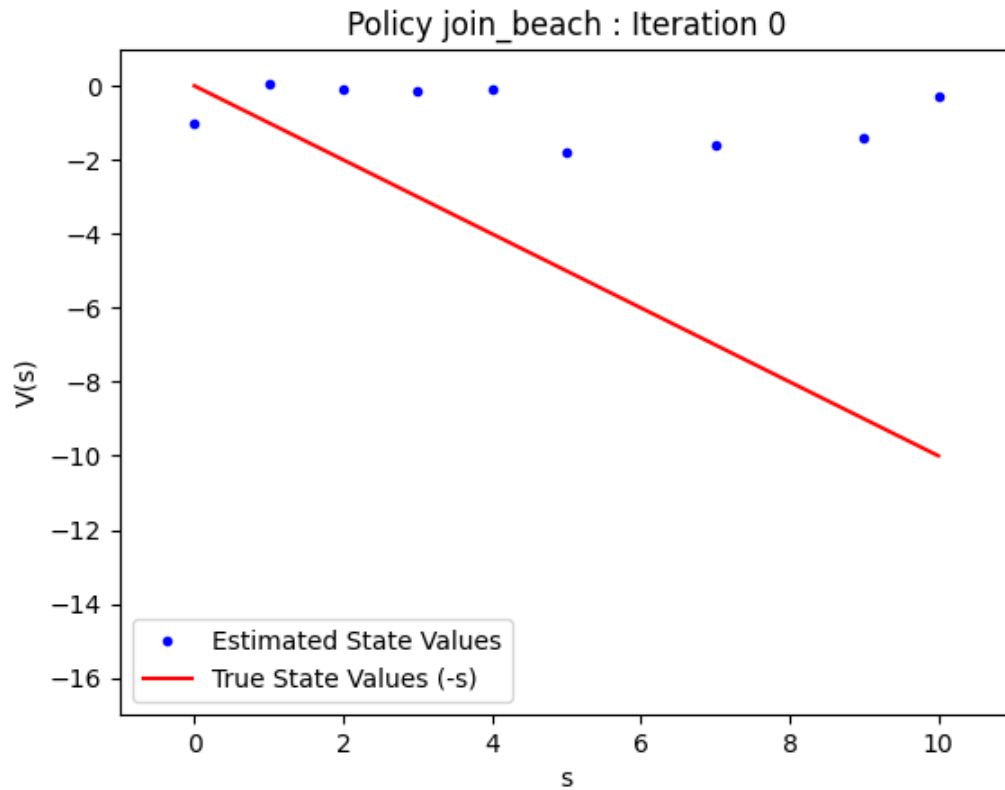
On obtient  $\hat{v}_\pi(s) \approx v_\pi(s)$ .

# Iterative Policy Evaluation : Résultats



$\pi = \text{se rapprocher}$   
( $\gamma = 0,98$ )

$\pi = \text{s'éloigner}$   
( $\gamma = 0,8$ )

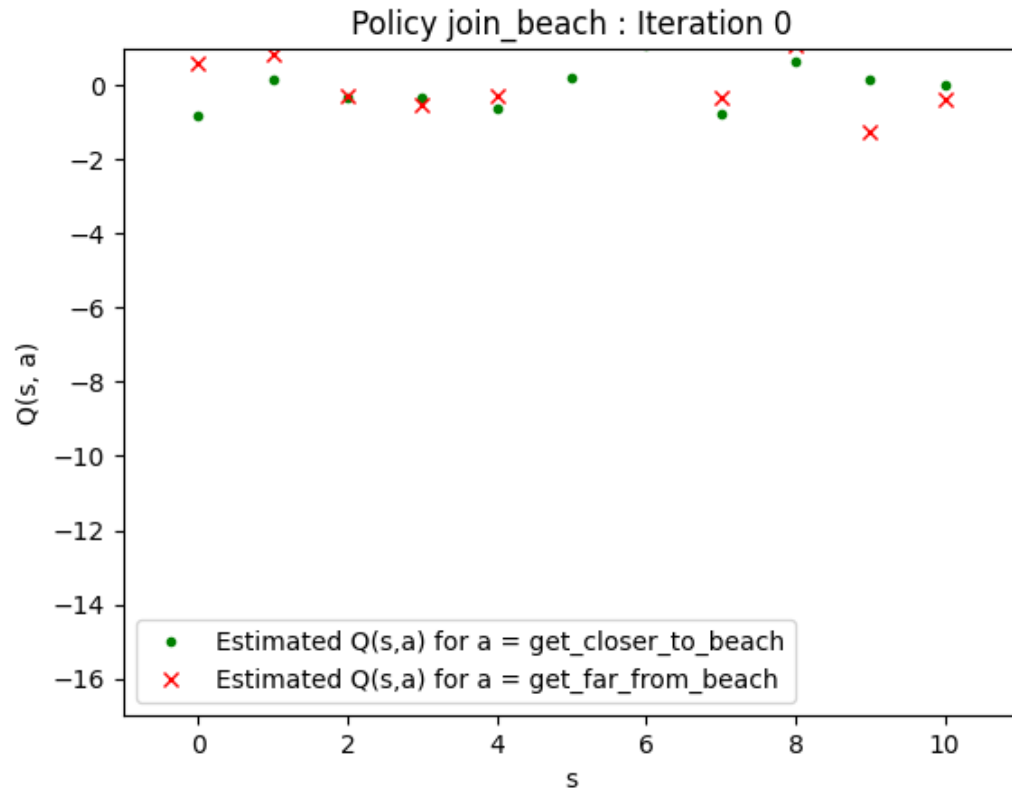


L'ordre avec lequel on parcourt les états est important

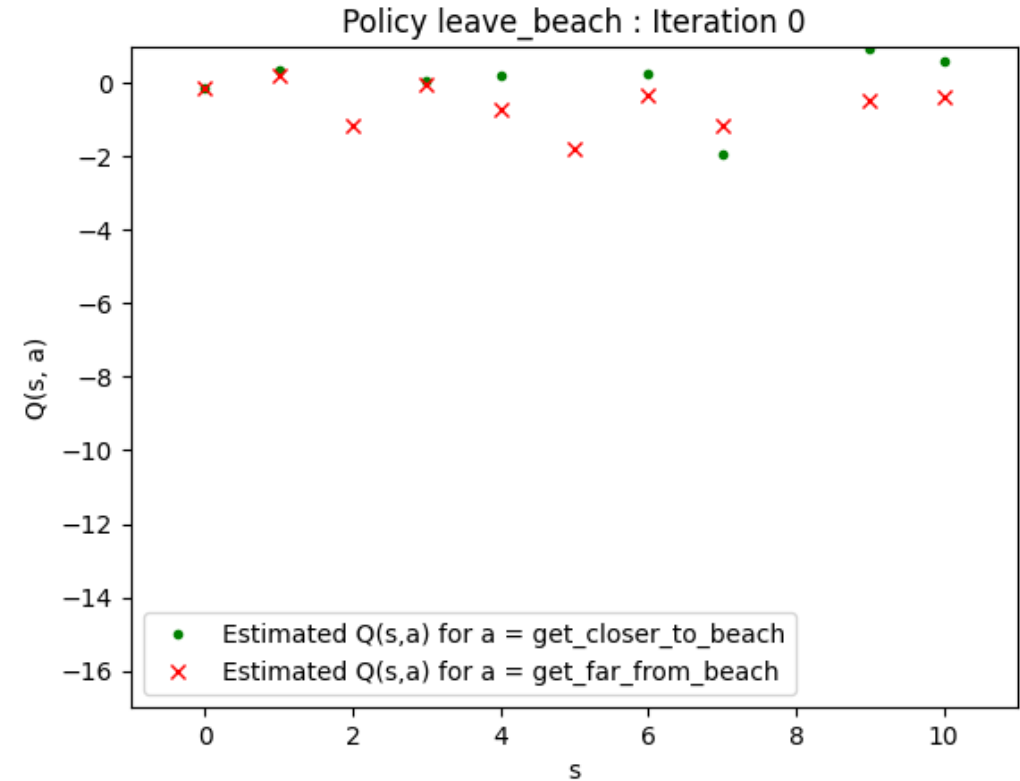
# Iterative Policy Evaluation : Résultats



$\pi = \text{se rapprocher}$   
( $\gamma = 0,98$ )



$\pi = \text{s'éloigner}$   
( $\gamma = 0,8$ )



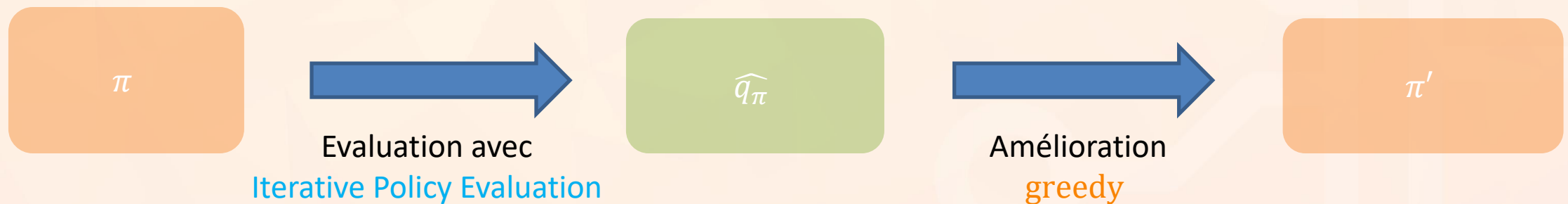
L'ordre avec lequel on parcourt les états est important

# De l'évaluation au Control Problem



Amélioration de la politique :

$$\pi(s) := \operatorname{argmax}_a \hat{q}_{\pi}(s, a) \approx \operatorname{argmax}_a q_{\pi}(s, a)$$



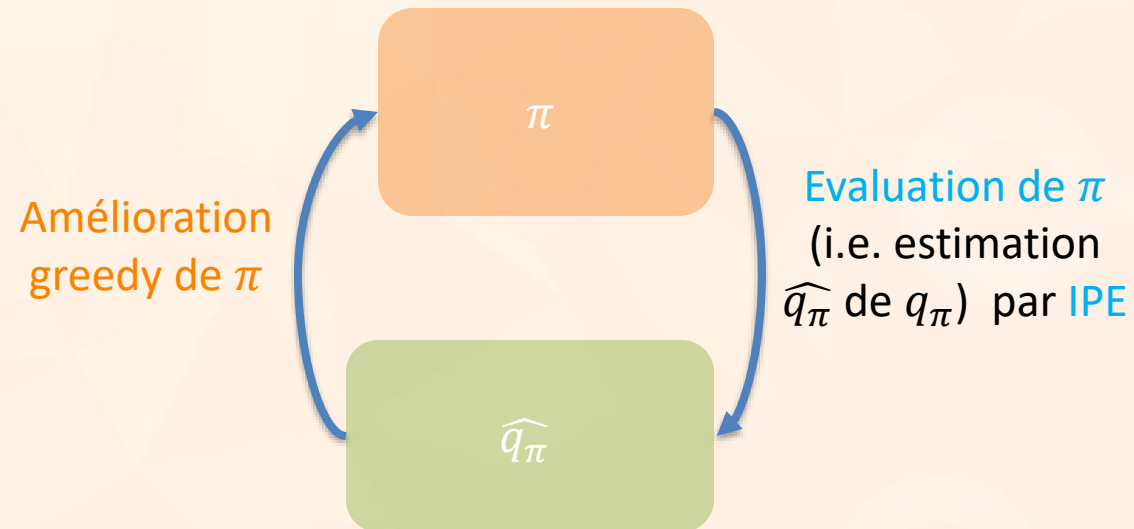


# Policy Iteration



Algorithme : **Policy Iteration**

Algorithme utilisé en Dynamic Programming pour résoudre le **Control Problem**.

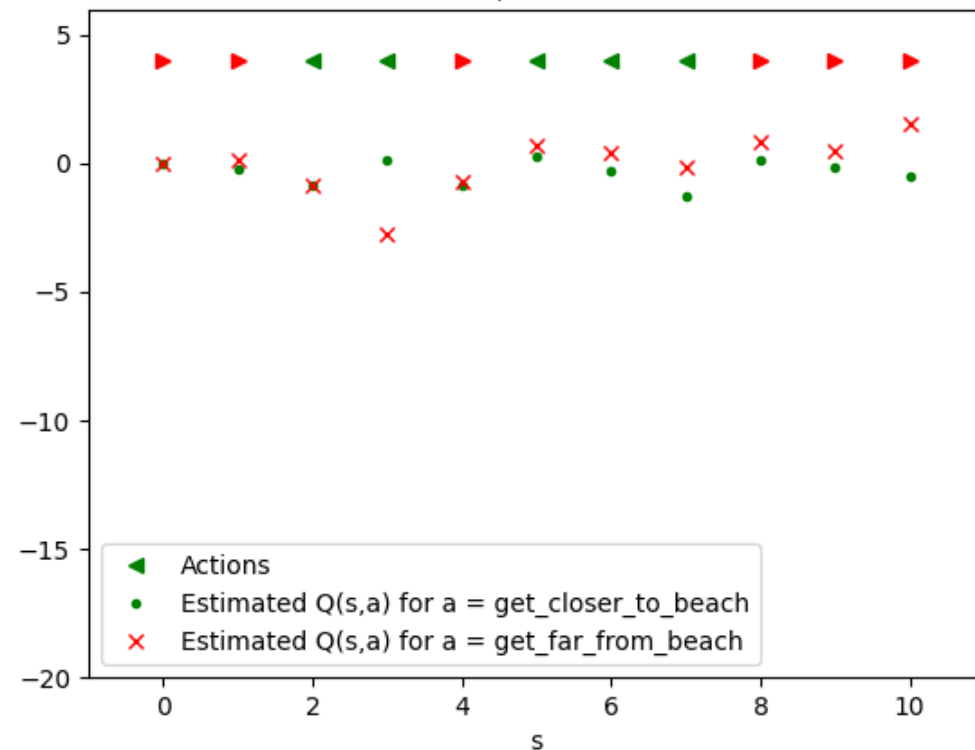


# Policy Iteration : Résultats

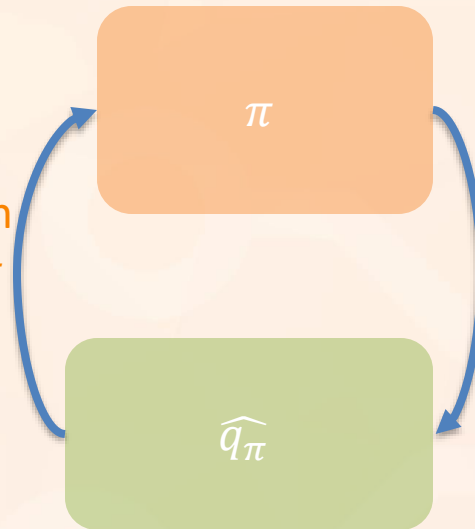


Policy Iteration ( $n_{\text{iter}} = 5$  itérations d'évaluation)

DP Control (PI or VI) - Iteration 0 | DP Prediction of Q (IPE) - Iteration 0 :

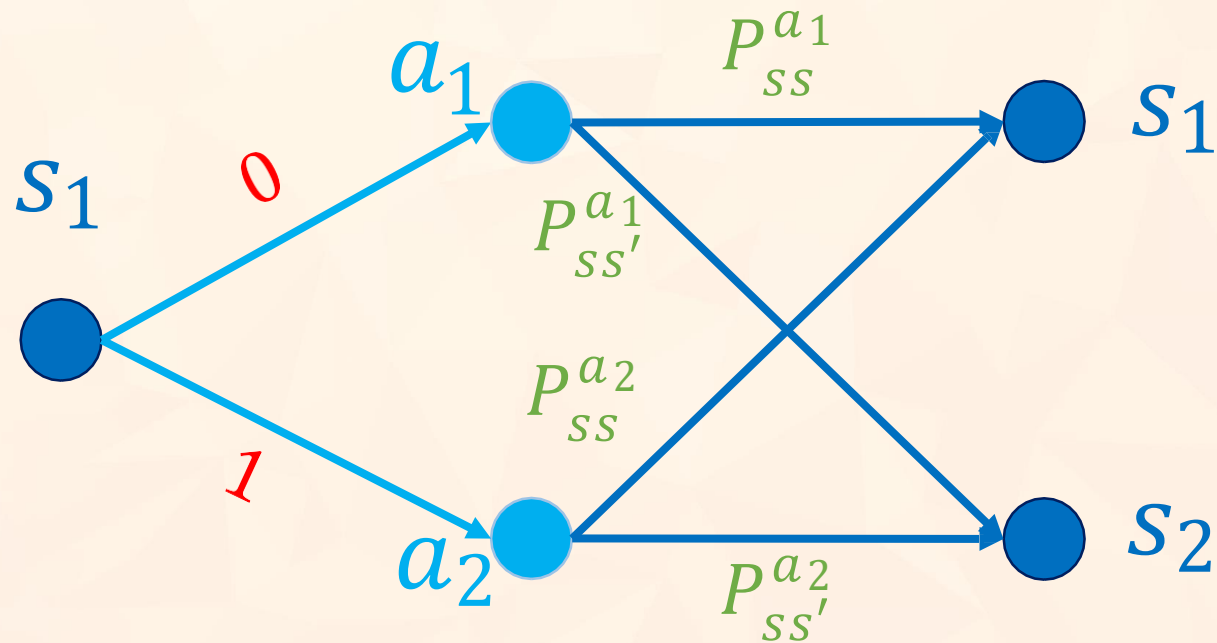


Amélioration  
greedy de  $\pi$



Evaluation de  $\pi$  par IPE  
en  $n_{\text{iter}}$  itérations

# Équations de Ford-Bellman optimales



On évalue non pas  $\pi$  quelconque  
mais directement  $\pi^*$  la politique  
optimale.

$$v_{\pi}(s) = \max_{a \in A} (R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_{\pi}(s'))$$

$$q_{\pi}(s, a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \max_{a' \in A} (q_{\pi}(s', a'))$$

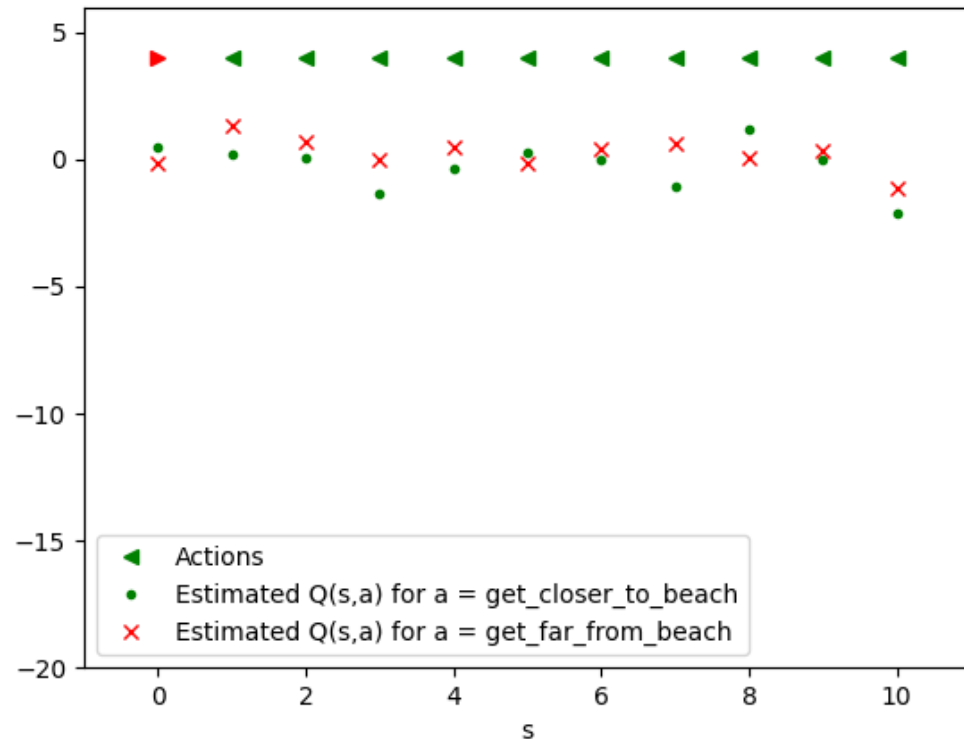
# Value Iteration



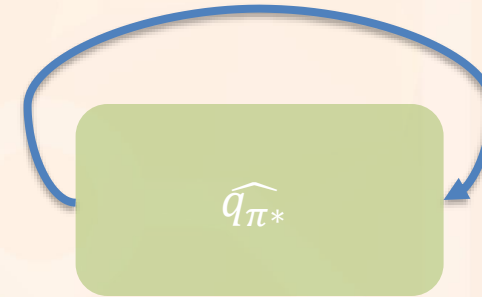
Algorithme : Value Iteration

Algorithme utilisé en Dynamic Programming pour résoudre le Control Problem.

DP Control (PI or VI) - Iteration 0 | DP Prediction of Q (IPE) - Iteration 0



Évaluation de  $\pi^*$  la politique optimale



$$q_{\pi}(s, a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \max_{a' \in A} (q_{\pi}(s', a'))$$

# Dynamic Programming : Conclusion

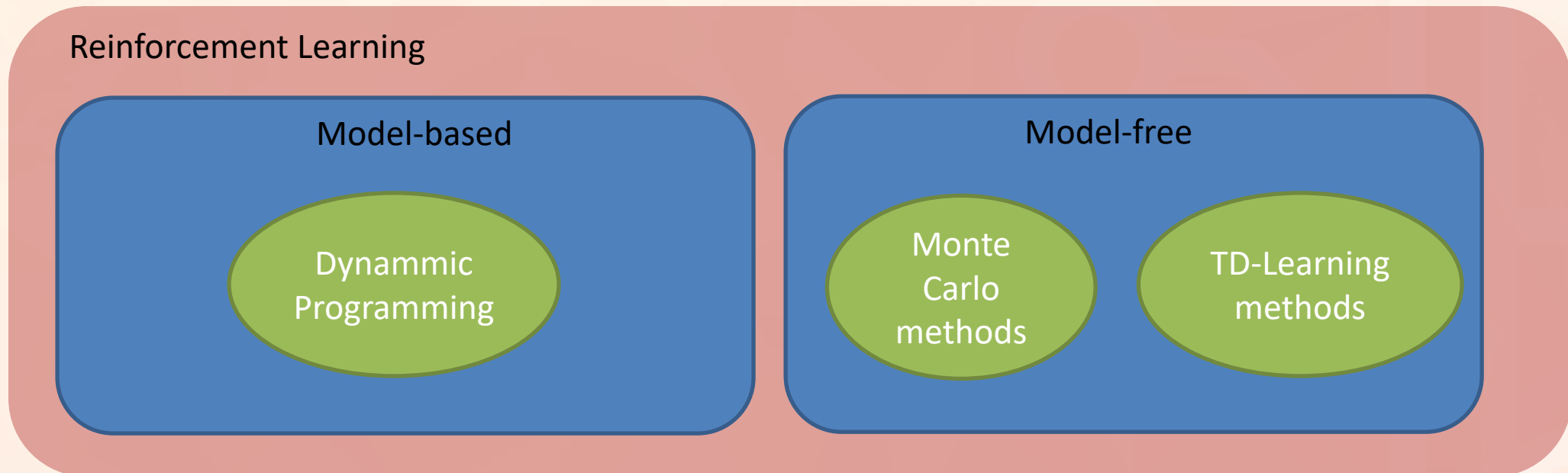


## Avantages :

- Converge rapidement vers la solution optimale
- Fortes fondations mathématiques

## Inconvénients :

- Adaptés à des petits espaces d'observations/actions discrets (finis) et non continus
- Model-Based : nécessite d'avoir accès au modèle





Fin de la 1<sup>ère</sup> formation  
Des questions ?



# Sources et ressources



Reinforcement Learning : an introduction, Sutton & Barto

Cours de DeepMind 2021 sur le RL : <https://dpmd.ai/DeepMindxUCL21>

Blog de Lilian Weng : <https://lilianweng.github.io>

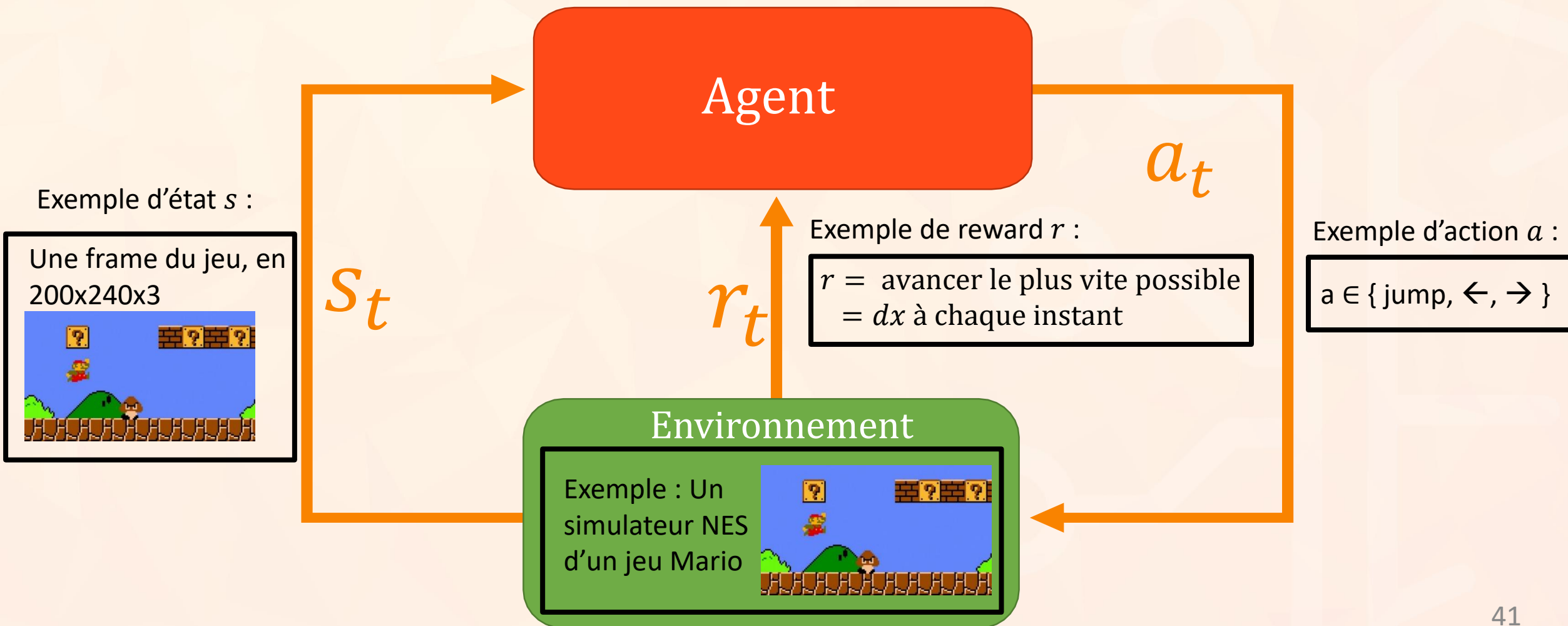
GitHub formation (slides & code) : [github.com/tboulet/Formation-Reinforcement-Learning](https://github.com/tboulet/Formation-Reinforcement-Learning)



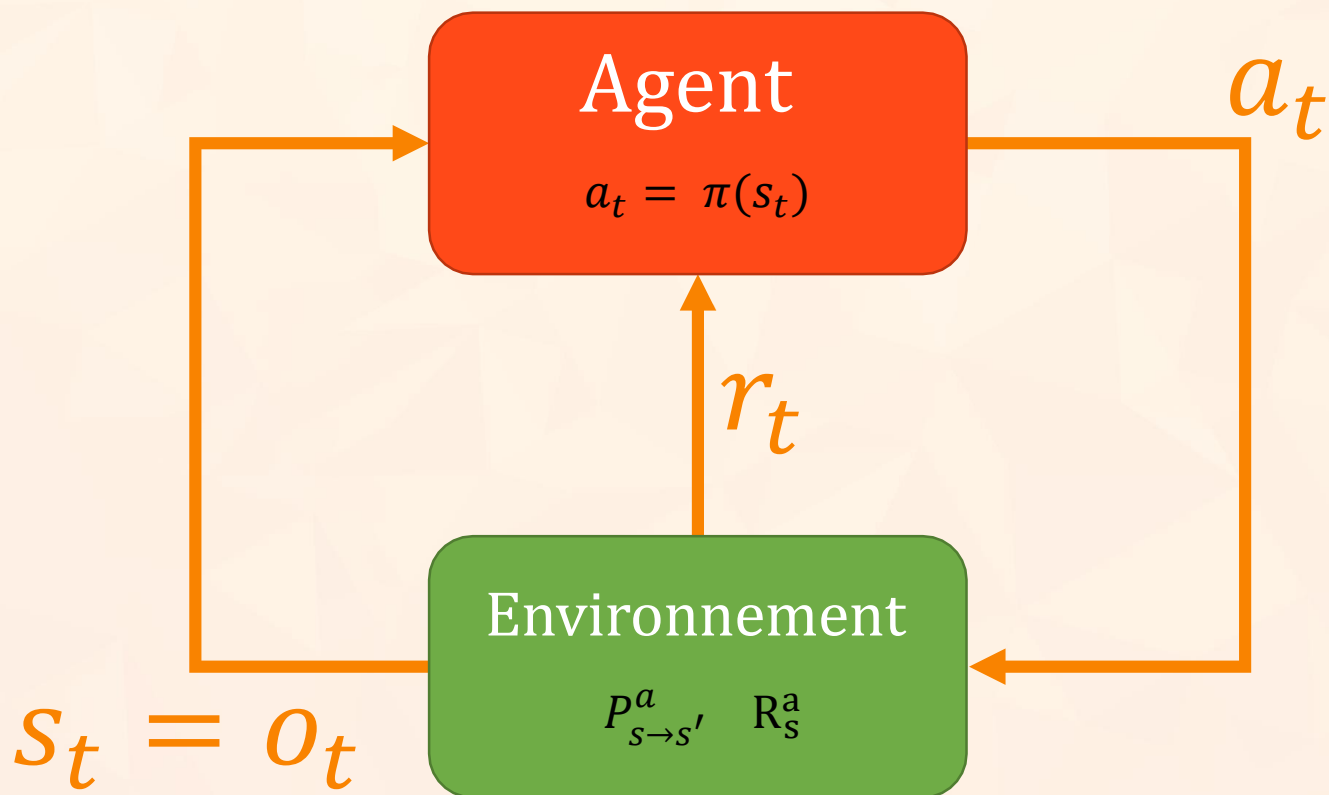
# Apprentissage par Renforcement

## Partie II

# Résumé de la formation précédente



# Résumé de la formation précédente



On se place dans le cadre d'un **MDP**:

- Environnement Markovien
- Parfaitement observable

Politique :

$$\pi(a|s) = P(A_t = a | S_t = s)$$

Valeur d'état :

$$v_\pi(s) = E[G_t | S_t = s, \pi]$$

Valeur d'action :

$$q_\pi(s, a) = E[G_t | S_t = s, A_t = a, \pi]$$

But : maximiser le gain futur  $G_t$  :

$$G_t = \sum_{t'=t}^T \gamma^{t'-t} r_{t'}$$

# Résumé de la formation précédente



**OceanEnv** : exemple d'environnement custom avec Gym :

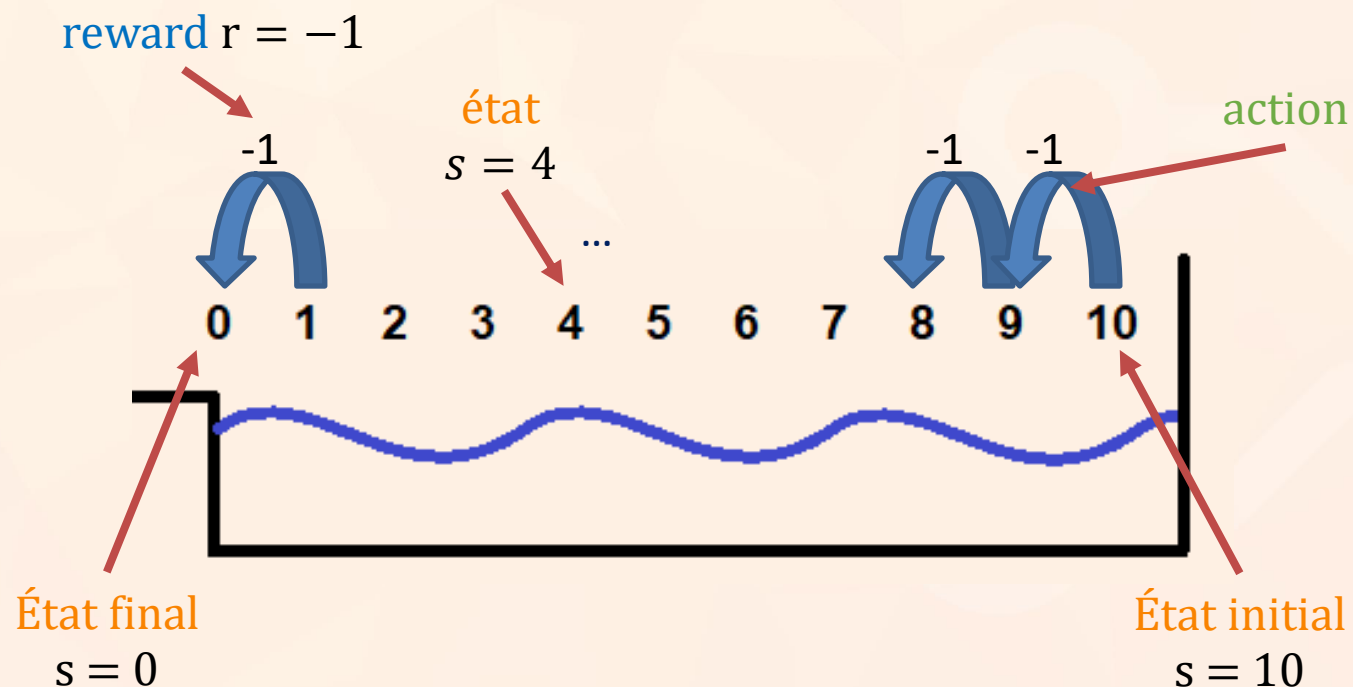
État :  $s \in \{0, 1, \dots, 10\}$ , représente la distance à la rive

Action :  $a \in \{\text{s'éloigner}, \text{se rapprocher}\}$

Reward :  $r = -1$ , à chaque  $t$  (punition tant que la rive n'est pas atteinte)

État initial :  $s_0 = 10$

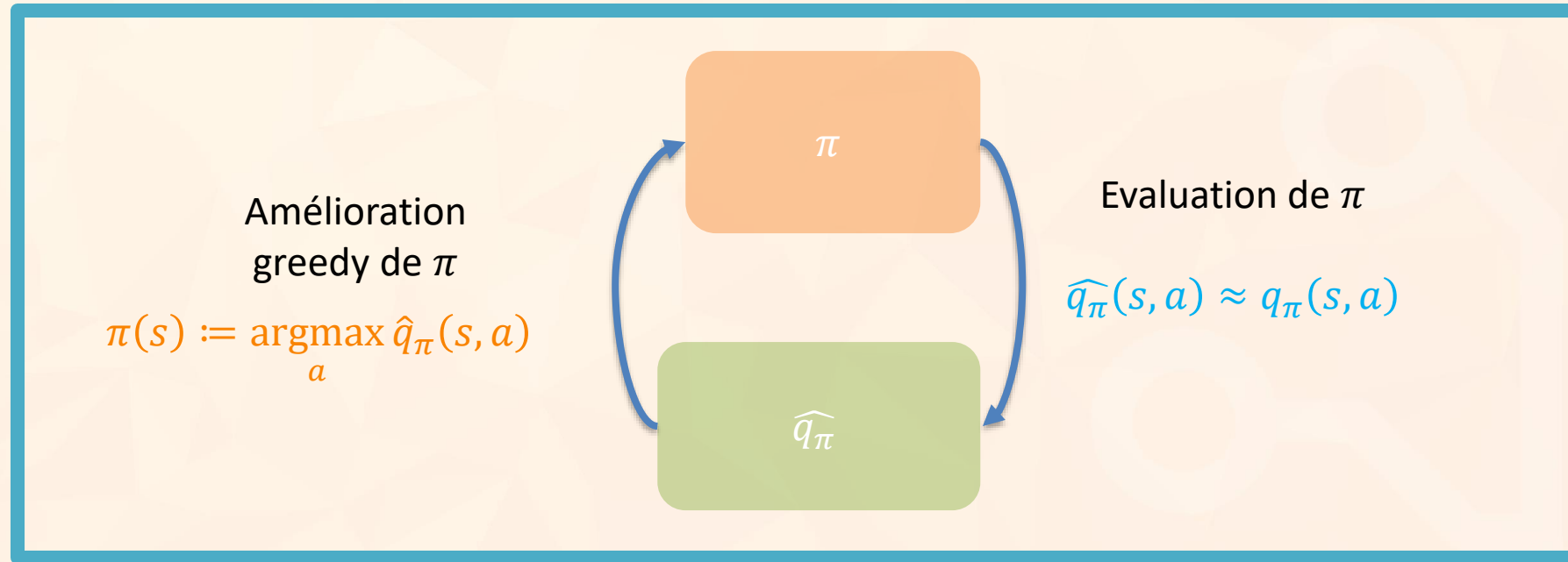
État terminal :  $s_T = 0$



# Résumé de la formation précédente



## Algorithme de RL







# Monte Carlo Methods

# MonteCarlo : Apprendre par l'interaction



But : apprendre  $\widehat{v}_\pi(s)$  à partir des  $G_t$  observés.

MonteCarlo (sur 1 épisode) :

- On joue un épisode  $\tau$  où on observe des états  $S_\tau$
- $\forall s_t \in S_\tau, \widehat{v}_\pi(s_t) \leftarrow G_t$

MonteCarlo (sur  $N$  épisodes) :

- On joue  $N$  épisode où on observe des états  $S$
- $\forall s \in S, \widehat{v}_\pi(s) \leftarrow \text{moyenne}(\{G_t | S_t = s\})$

$N$  = tradeoff temps/variance

MonteCarlo pour  $q$ :

- On joue  $N$  épisode où on observe des couples  $(s, a)$
- $\forall (s, a)$  observés,  $\widehat{q}_\pi(s, a) \leftarrow \text{moyenne}(\{G_t | S_t = s, A_t = a\})$

**Avantages** : intuitif, mathématiquement vrai :  $E[\underbrace{G_t}_{v_{MC}} | S_t = s] = v_\pi(s)$  , l'estimateur de MonteCarlo  $v_{MC}$  est dit non-biaisé

**Inconvénients** : terminal, haute variance

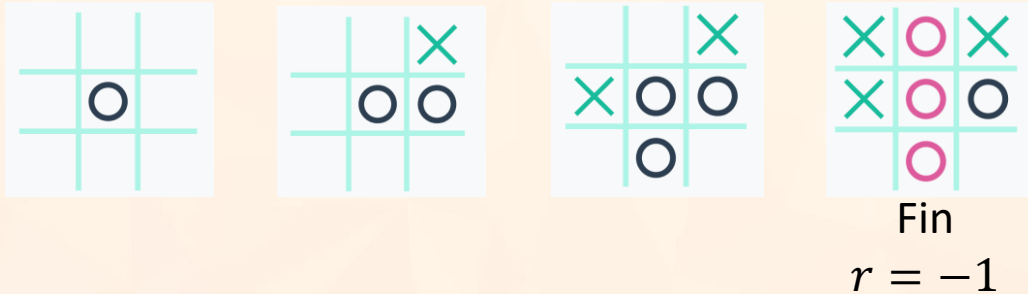
$v_{MC}$

# MonteCarlo : Apprendre par l'interaction

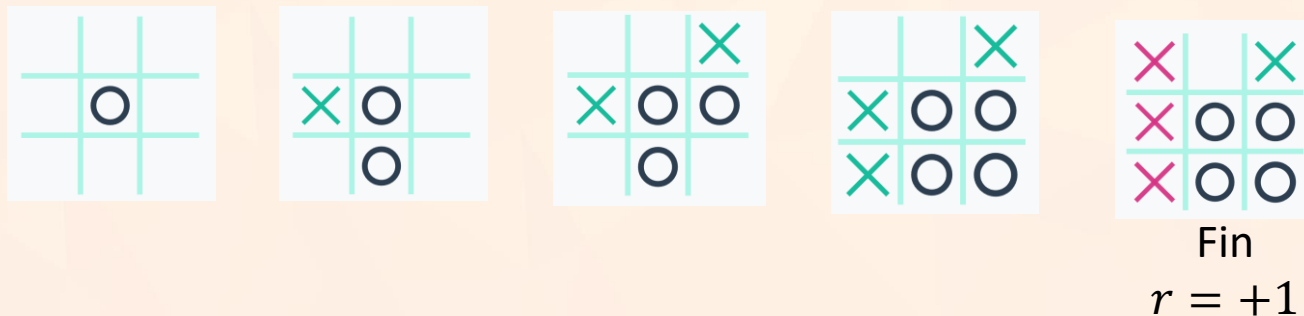


Environnement : morpion face à un adversaire jouant aléatoirement  
 Reward  $r = \pm 1$  quand on gagne/perd, 0 sinon

Episode 1 :



Episode 2 :



L'apprentissage est **terminal** : on est obligé  
 d'attendre la fin d'un épisode pour mettre à jour  $v$

$s$	$\widehat{v}_{\pi}(s)$
	$\frac{-1 + 1}{2}$
	$-1$
	$+1$
	$\frac{-1 + 1}{2}$
	$+1$

# Remarque : cumulative vs. moving average



## Cumulative average

$$\hat{X}_N = \frac{x_1 + x_2 + \dots + x_N}{N}$$

Formule incrémentale :

$$\hat{X}_{N+1} = \frac{N}{N+1} \hat{X}_N + \frac{1}{N+1} x_{N+1}$$

- $\hat{X}_N$  **tend** vers  $E[X]$  à l'infini.
- Adapté aux env. et politiques stationnaires
- Tous les  $x_i$  pesent autant

Exemple pour Monte Carlo :

A chaque fin d'épisode, pour chaque  $s$  vu à  $t_s$  :

$$\widehat{v}_\pi(s) = \frac{N(s)}{N(s)+1} \widehat{v}_\pi(s) + \frac{1}{N(s)+1} G_{t_s}$$

## Moving average

Formule incrémentale :

$$\hat{X}_{N+1} = (1 - \alpha) \hat{X}_N + \alpha x_{N+1}$$

On notera :  $\hat{X} \leftarrow x_i$

- $\hat{X}$  **se rapproche** de  $E[X]$  en permanence
- Adapté aux env. et politiques **non** stationnaires
- Les  $x_i$  récents pesent plus

Exemple pour Monte Carlo :

A chaque fin d'épisode, pour chaque  $s$  vu à  $t_s$  :

$$\widehat{v}_\pi(s) = 0,99 \widehat{v}_\pi(s) + 0,01 G_{t_s}$$

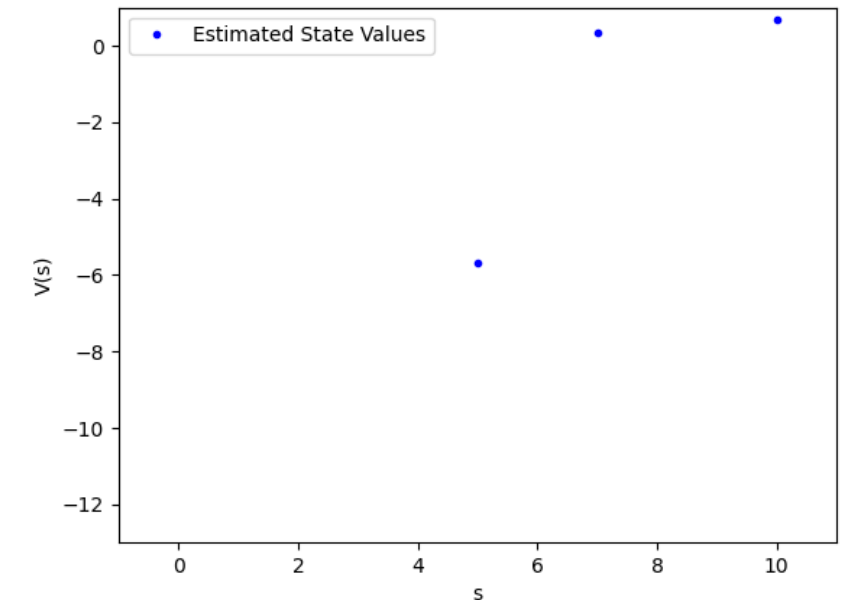
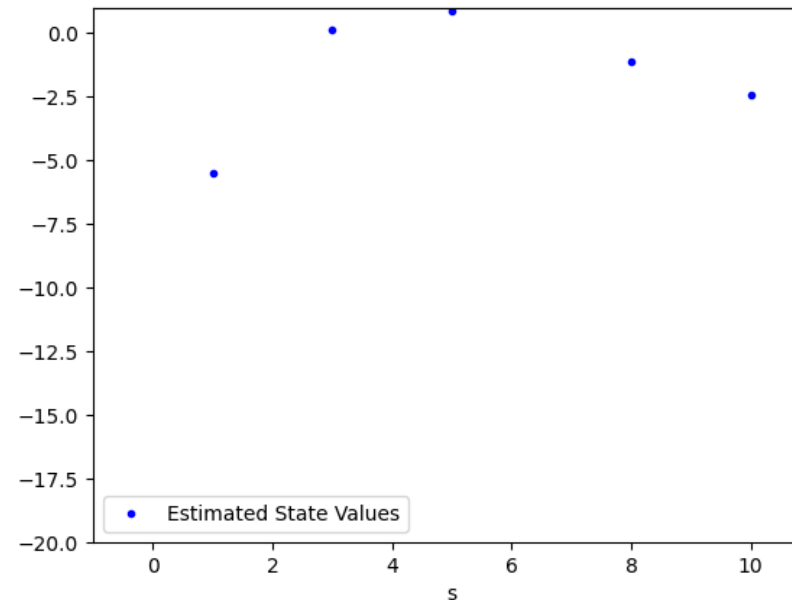
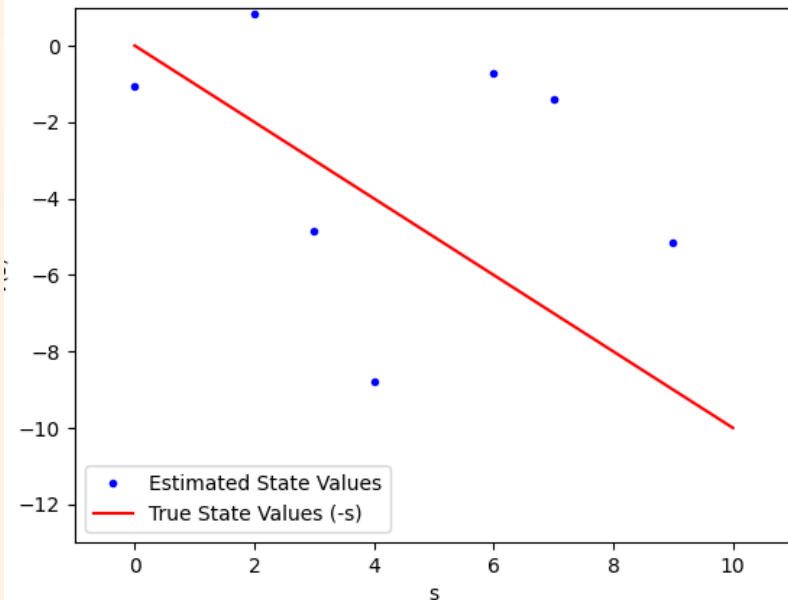
# Monte Carlo : Résultats pour $\hat{v}_\pi(s)$



$\pi = \text{se rapprocher}$

$\pi = \begin{cases} \text{se rapprocher,} & 80\% \\ \text{s'éloigner,} & 20\% \end{cases}$

$\pi = \text{s'éloigner}$



Remarques d'implémentation : les  $\hat{v}_\pi(s)$  sont initialisés aléatoirement et on utilise **moving average** pour apprendre  $v_\pi(s)$ .



Problème d'exploration : pour  $\pi_{s'eloigner}$ , on ne voit jamais les états proches de la rive, et donc on ne peut pas les évaluer

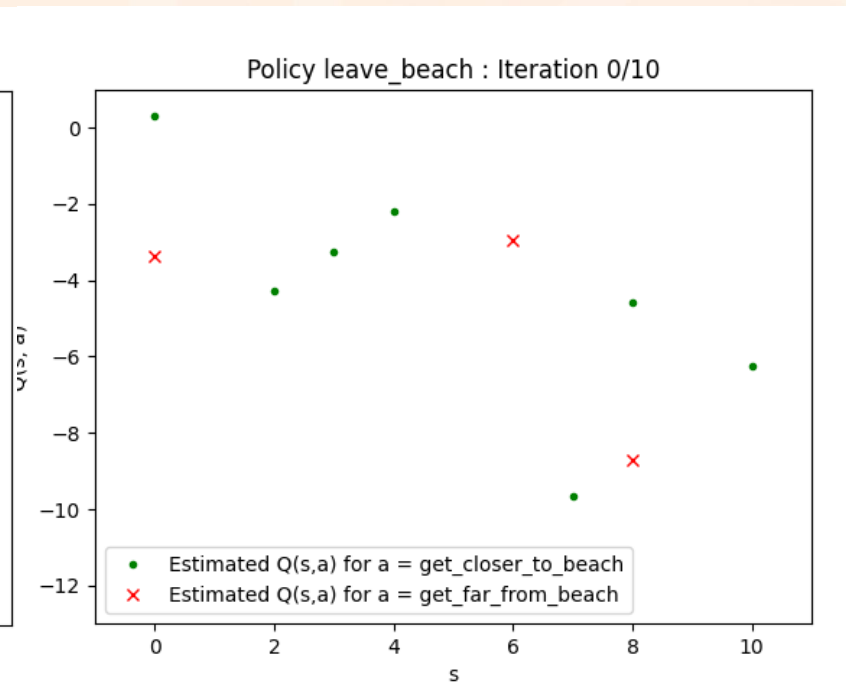
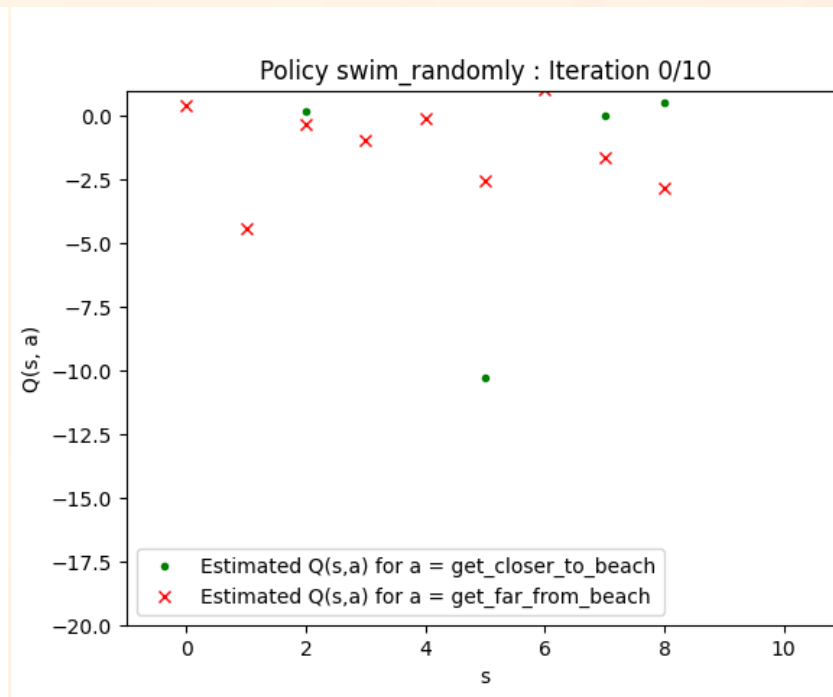
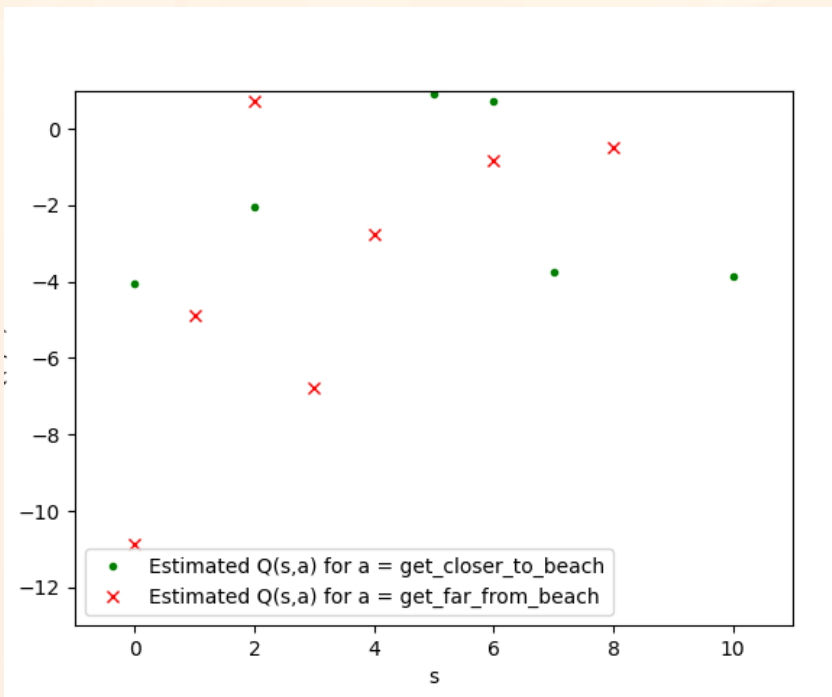
# Monte Carlo : Résultats pour $\hat{q}_\pi(s)$



$\pi = \text{se rapprocher}$

$\pi = \begin{cases} \text{se rapprocher,} & 80\% \\ \text{s'éloigner,} & 20\% \end{cases}$

$\pi = \text{s'éloigner}$



Problème d'exploration : les actions qui ne sont jamais prises, et les états qui ne sont jamais atteints en jouant  $\pi$  ne sont pas évalués



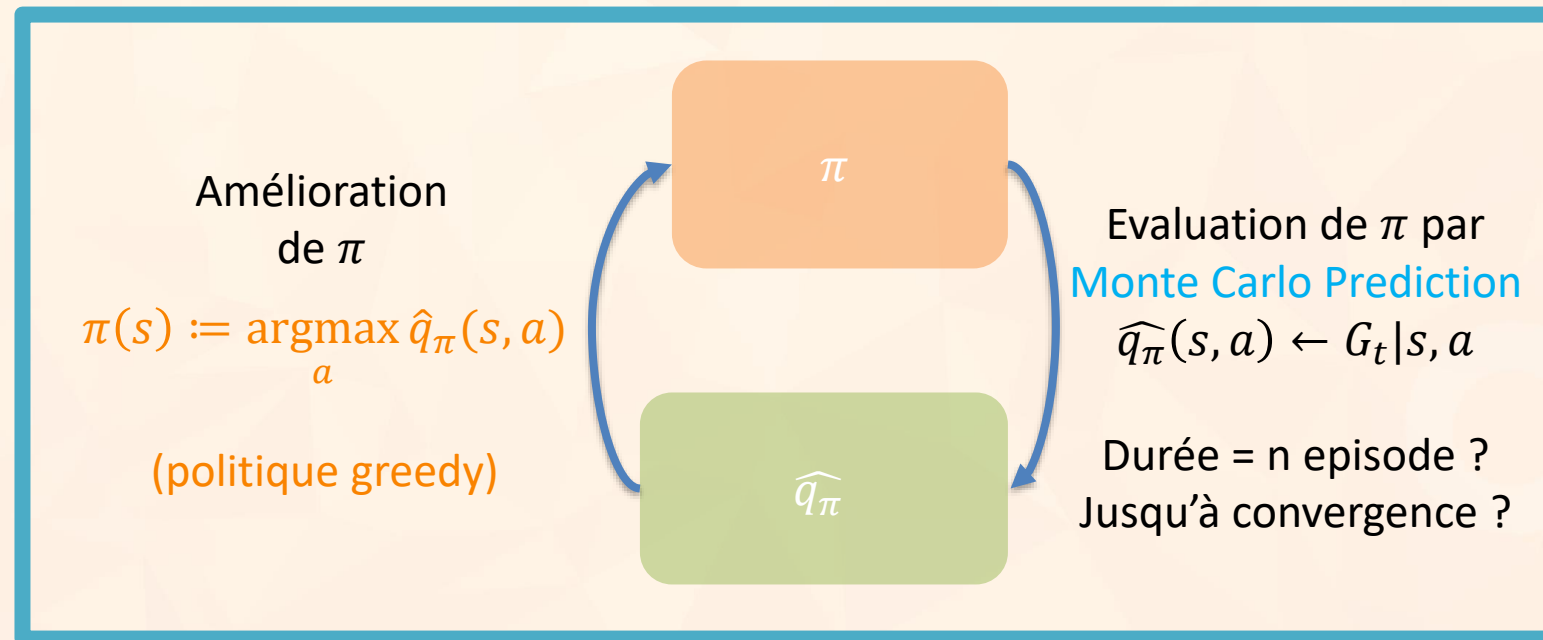
MonteCarlo ne sait **PAS** évaluer les politiques déterministes



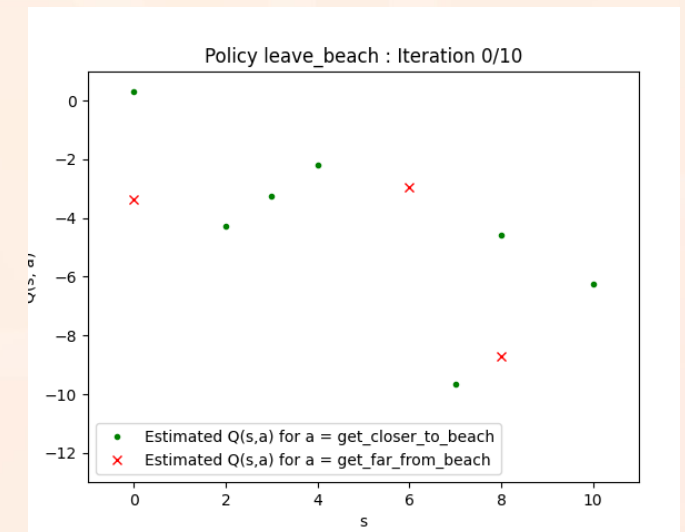
# Monte Carlo : Control Problem



Algorithme : Monte Carlo Control



Problème de la politique greedy : on a vu que Monte Carlo n'évalue pas bien les politiques déterministes pour ce qui est des actions non choisies, car l'algorithme a besoin d'expériences où ces actions ont lieu.



# Dilemme Exploration vs. Exploitation



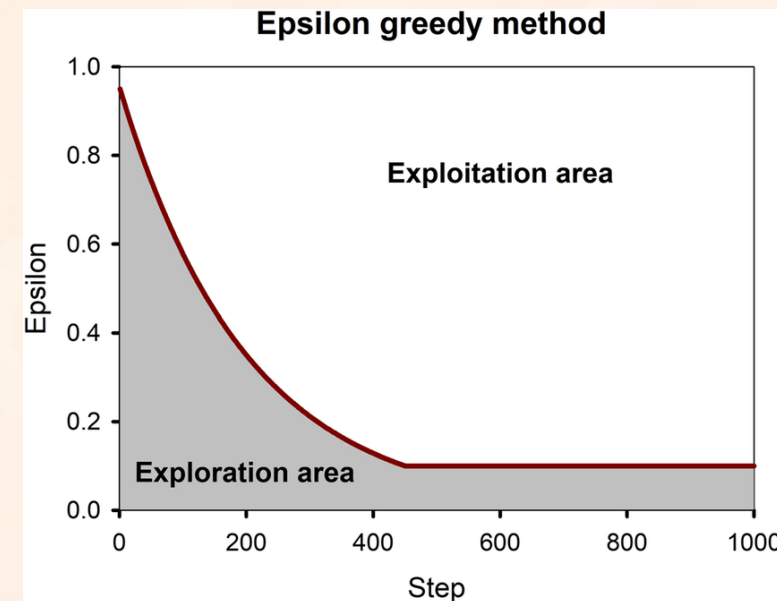
Problème d'**exploration** : certaines actions sont peu visitées donc moins bien estimées.

Solution : utiliser des politiques plus exploratives

Politique  $\varepsilon$ -greedy :  $\pi(s) := \begin{cases} \operatorname{argmax}_a \hat{q}_\pi(s, a) & \text{avec probabilité } 1 - \varepsilon \\ a \sim A & \text{avec probabilité } \varepsilon \end{cases}$

Politique de Boltzmann :  $\pi(a|s) := \frac{e^{\hat{q}_\pi(s, a)/T}}{\sum_{a'} e^{\hat{q}_\pi(s, a')/T}}$

Politique UCB :  $\pi(s) := \operatorname{argmax}_a \underbrace{\hat{q}_\pi(s, a)}_{\text{exploitation}} + c \underbrace{\sqrt{\frac{\log(t)}{N(s, a)}}}_{\text{exploration}}$

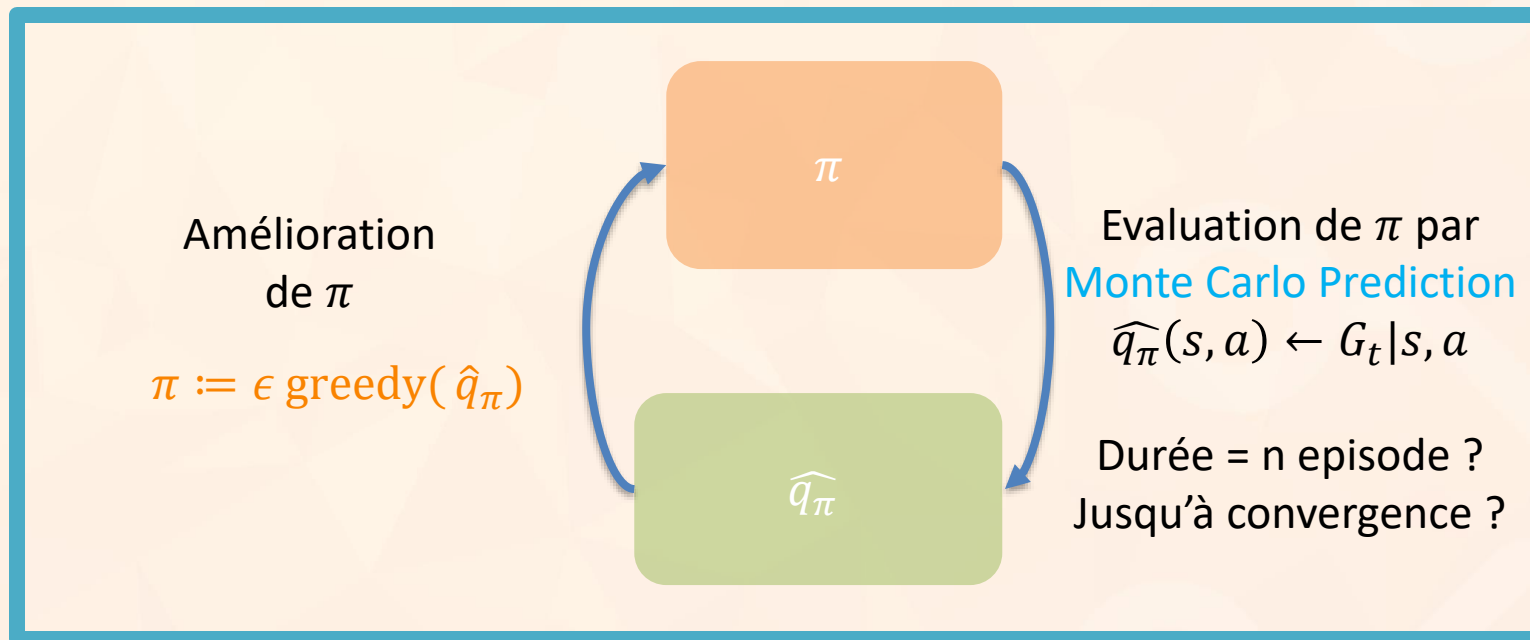


Remarque : le dilemme exploitation/exploration est un aspect essentiel du RL. Il est largement étudié dans un problème fondamental du RL, celui des K machines à sous (N-Bandit Problem).

# Monte Carlo : Control Problem



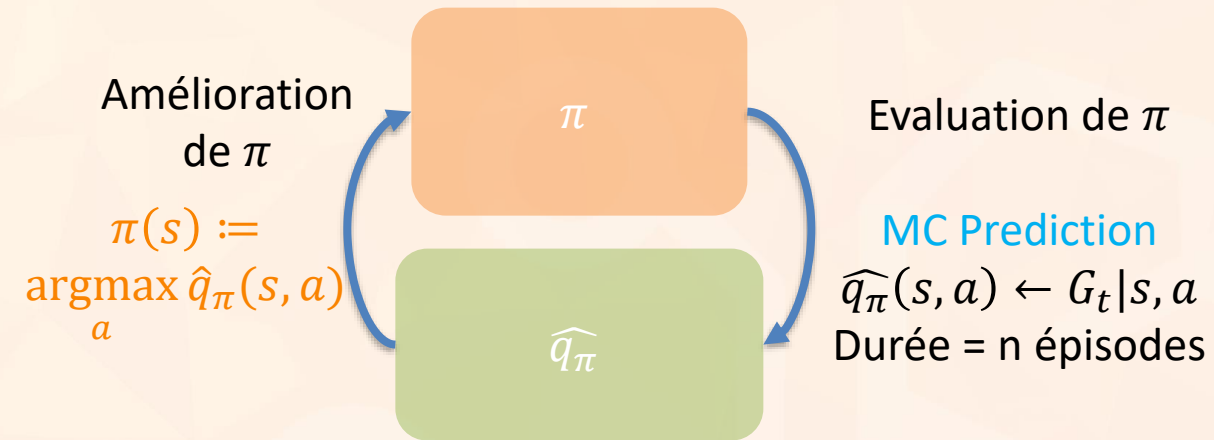
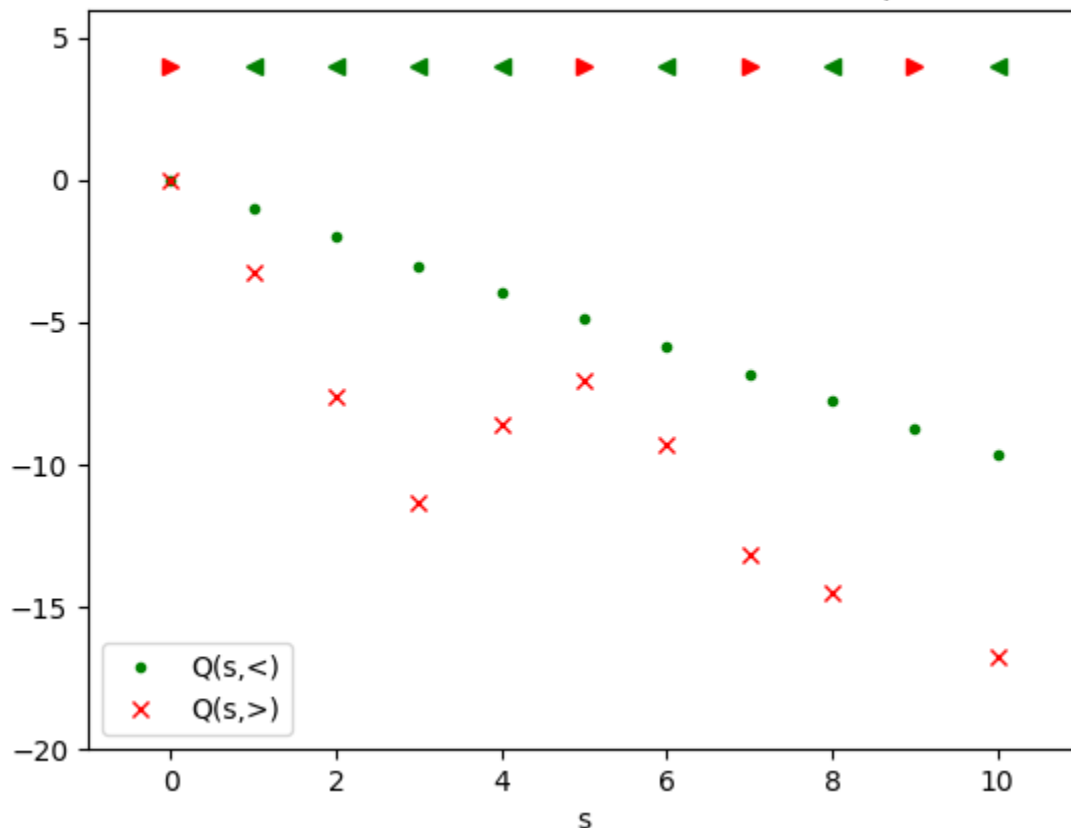
Algorithme : Monte Carlo Control



# Monte Carlo Control : Résultats



MC Control - Iteration 5/8 - MC Prediction of Q - Episode 38/40



Remarques d'implémentation :

Les  $q$  values sont initialisées aléatoirement au début, puis à chaque phase d'évaluation (Prediction) elles sont initialisés comme les précédentes  $Q$  values.

On explore avec une politique  $\epsilon$  **greedy** avec  $\epsilon$  constant à 0,1.

# Monte Carlo : Conclusion



## Avantages :

- Peut apprendre de vraies expériences donc adapté aux problèmes réels

## Inconvénients :

- Offline : On doit attendre la fin d'un episode pour estimer les valeurs
- Variance de l'estimateur élevé quand  $T$  devient grand

## Reinforcement Learning

### Model-based

Dynamic  
Programming

### Model-free

Monte  
Carlo  
methods

TD-Learning  
methods



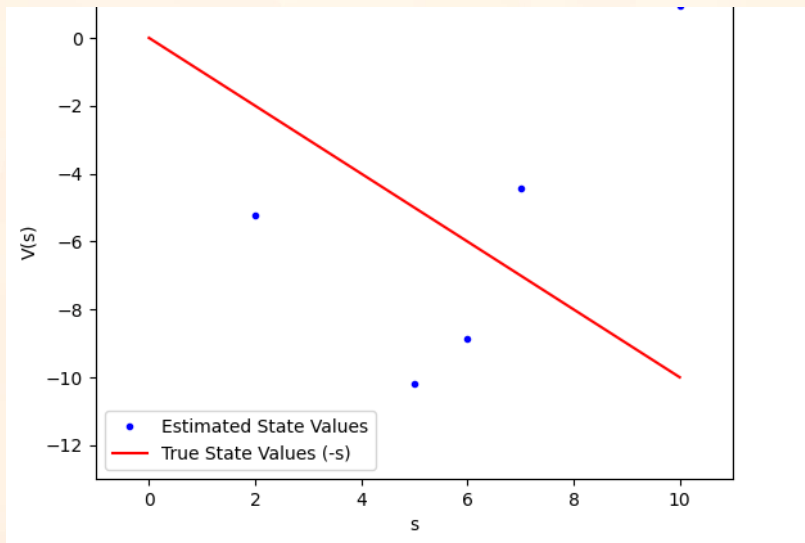
# TD Learning methods



# Vers TD Learning : le dilemme biais-variance



Exemple d'algo avec du biais :



Qu'est ce que le biais d'un estimateur  $\hat{v}$  ?

C'est l'erreur systématique  $|E[\hat{v}] - v_\pi|$ .

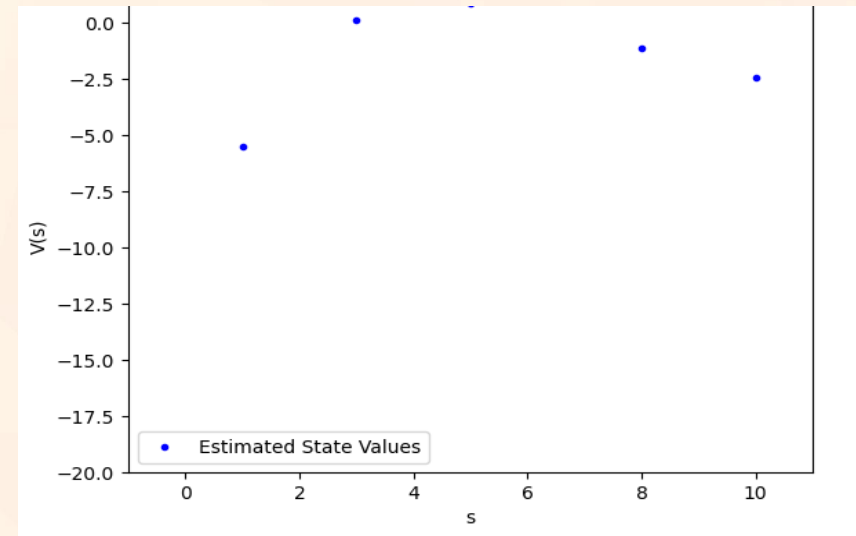
Si le biais est non nul, on apprend vers une mauvaise valeur.

$$v_{\text{MonteCarlo}} = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \dots + \gamma^{T-t} R_T$$

- Sans biais :  $E[v_{\text{MonteCarlo}}] = v_\pi(s_t)$

- Variance très importante car  $R_T$  et  $s_t$  très peu corellés

Exemple d'algo avec de la variance :



Qu'est ce que la variance ?

C'est l'erreur typique  $(\hat{v} - E[\hat{v}])^2$  obtenue pour 1 estimation.

Si la variance est élevée, il faudra beaucoup de sample avant d'avoir une bonne estimation.

Sources de variance : transitions/reward/politique de l'agent stochastiques

# TD Learning



Comme dans Monte Carlo on apprend par l'expérience, mais ici par **bootstrapping** on n'attend pas la fin de l'épisode :

$$\widehat{v}_{\pi}(s_t) \leftarrow r_t + \gamma \widehat{v}_{\pi}(s_{t+1}) \quad \text{TD(0)}$$

$$\widehat{v}_{\pi}(s_t) \leftarrow r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots + \gamma^{T-t} r_T = G_t \quad \text{MonteCarlo}$$

$R_t + \gamma v_{\pi}(S_{t+1})$  est un estimateur sans biais et à faible variance de  $v_{\pi}(s_t) = E_{\pi}[G_t | S_t = s_t]$ .

$R_t + \gamma \widehat{v}_{\pi}(S_{t+1})$  est un estimateur **avec biais** mais à faible variance.



terme non biaisé,  
permettant d'apprendre

terme biaisé, estime la  
suite des rewards

# TD Learning : Prediction Problem



Implémentation :  $\widehat{v}_{\pi}(s_t) := \widehat{v}_{\pi}(s_t) + \alpha(r_t + \gamma \widehat{v}_{\pi}(s_{t+1}) - \widehat{v}_{\pi}(s_t))$

$\delta_t$  = Différence Temporelle (TD)

avec  $\alpha = 0,01$  par exemple, le Learning Rate

Algorithme : **TD(0)** (pour le **Prediction Problem** d'estimer  $v$ )

```
Input: the policy  $\pi$  to be evaluated
Initialize  $V(s)$  arbitrarily (e.g.,  $V(s) = 0, \forall s \in \mathcal{S}^+$ )
Repeat (for each episode):
  Initialize  $S$ 
  Repeat (for each step of episode):
     $A \leftarrow$  action given by  $\pi$  for  $S$ 
    Take action  $A$ ; observe reward,  $R$ , and next state,  $S'$ 
     $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$ 
     $S \leftarrow S'$ 
  until  $S$  is terminal
```



L'apprentissage est **non nécessairement terminal** : on peut apprendre pendant qu'on joue !

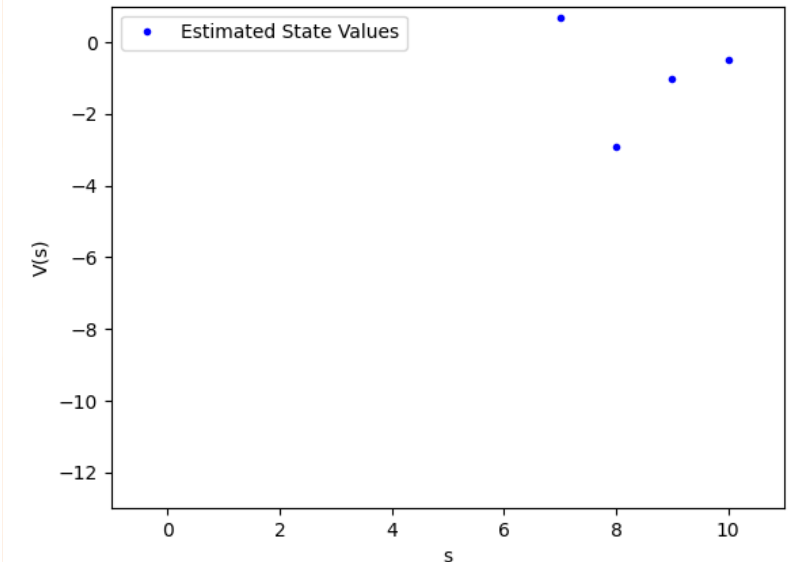
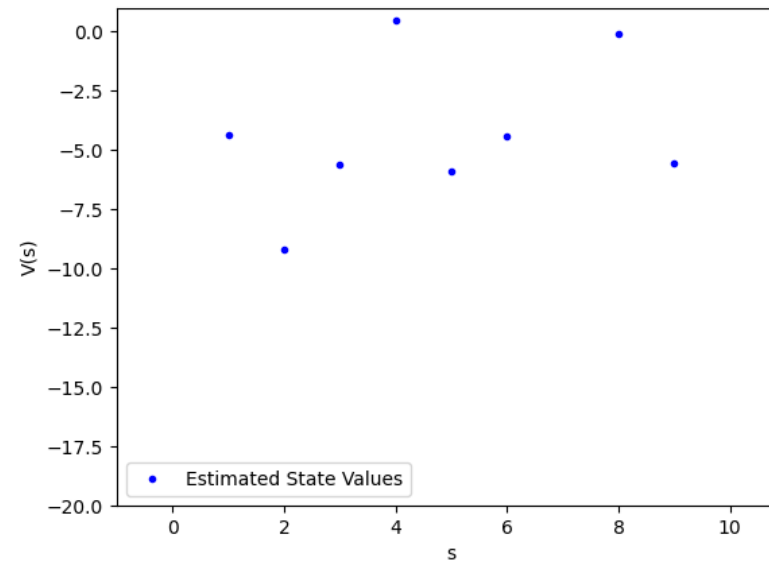
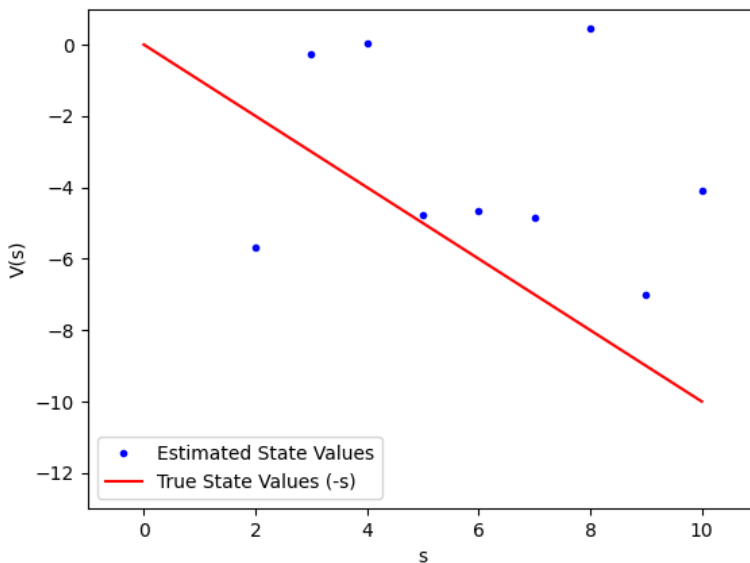
# TD(0) : Résultats pour $\hat{v}_\pi(s)$



$\pi = \text{se rapprocher}$

$\pi = \begin{cases} \text{se rapprocher,} & 80\% \\ \text{s'éloigner,} & 20\% \end{cases}$

$\pi = \text{s'éloigner}$



Problème d'exploration : pour  $\pi_{s'eloigner}$ , on ne voit jamais les états proches de la rive, et donc on ne peut pas les évaluer

# TD Learning : Prediction Problem



Et pour les  $q$  values ?

$$\widehat{v}_{\pi}(s_t) \leftarrow r_t + \gamma \widehat{v}_{\pi}(s_{t+1})$$

TD(0)

pour estimer  $E_{\pi}[G_t | S_t = s_t]$

$$\widehat{q}_{\pi}(s_t, a_t) \leftarrow r_t + \gamma \widehat{q}_{\pi}(s_{t+1}, a_{t+1})$$

SARSA

$E_{\pi}[G_t | S_t = s_t, A_t = a_t]$

$$\widehat{q}_{\pi}(s_t, a_t) \leftarrow r_t + \gamma \sum_{a'} \pi(a' | s_{t+1}) \widehat{q}_{\pi}(s_{t+1}, a') \quad \text{SARSA-Expected}$$

$E_{\pi}[G_t | S_t = s_t, A_t = a_t]$

← même biais mais moins de variance car pas de  $a_{t+1}$

Dans le cas de  $\pi = \text{greedy}(\widehat{q}_{\pi})$ , SARSA-Expected correspond à :

$$\widehat{q}_{\pi} \leftarrow r_t + \gamma \max_{a'} \widehat{q}_{\pi}(s_{t+1}, a')$$

Q Learning

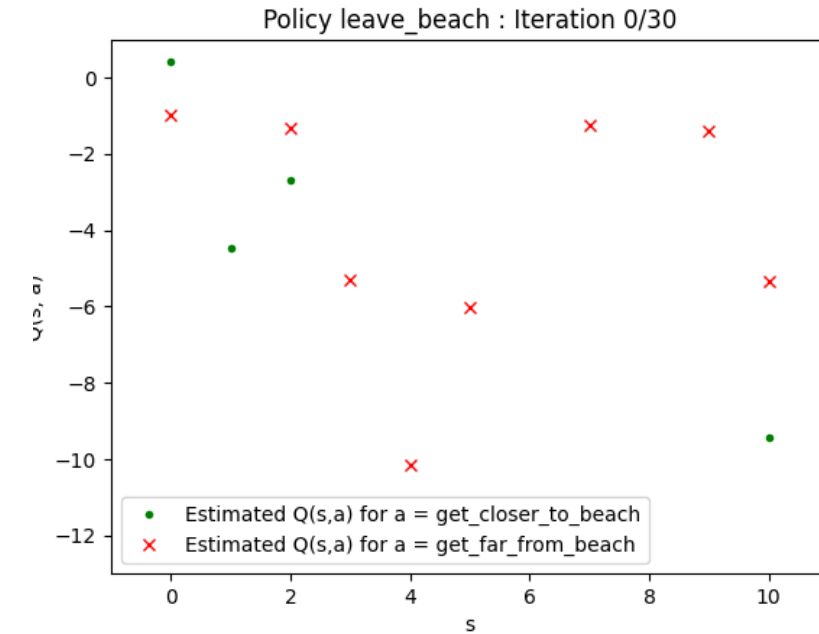
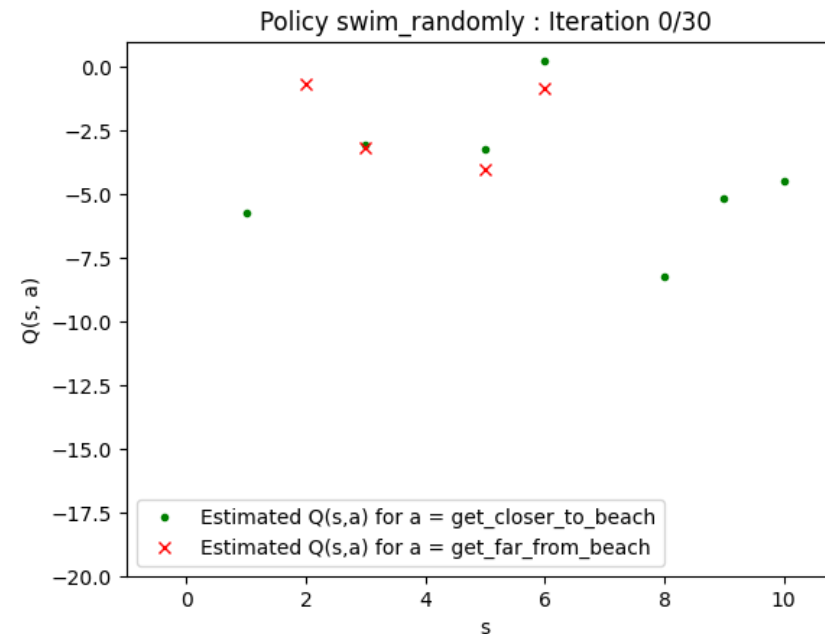
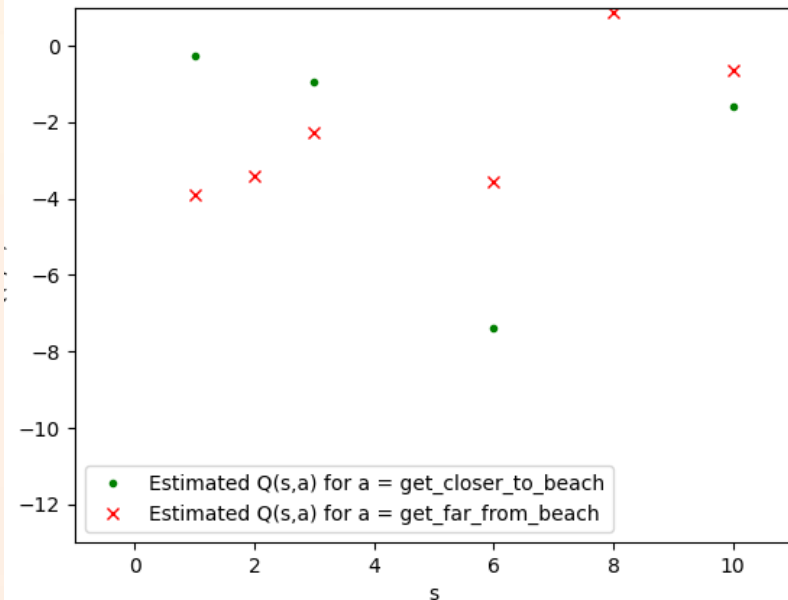
# SARSA: Résultats pour $\hat{q}_\pi(s)$



$\pi = \text{se rapprocher}$

$$\pi = \begin{cases} \text{se rapprocher,} & 80\% \\ \text{s'éloigner,} & 20\% \end{cases}$$

$\pi = \text{s'éloigner}$



Problème d'exploration : pour  $\pi_{s'eloigner}$ , on ne voit jamais les états proches de la rive, et donc on ne peut pas les évaluer



# TD Learning : Control Problem



Algorithme : **SARSA Control**

Algorithme de type TD pour le **Control Problem**

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  $S$

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

Repeat (for each step of episode):

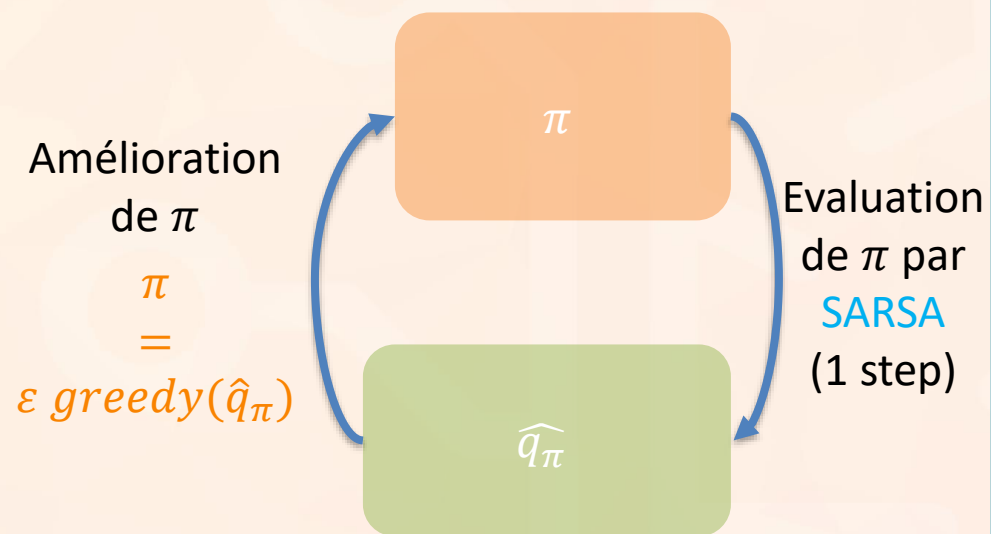
Take action  $A$ , observe  $R, S'$

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

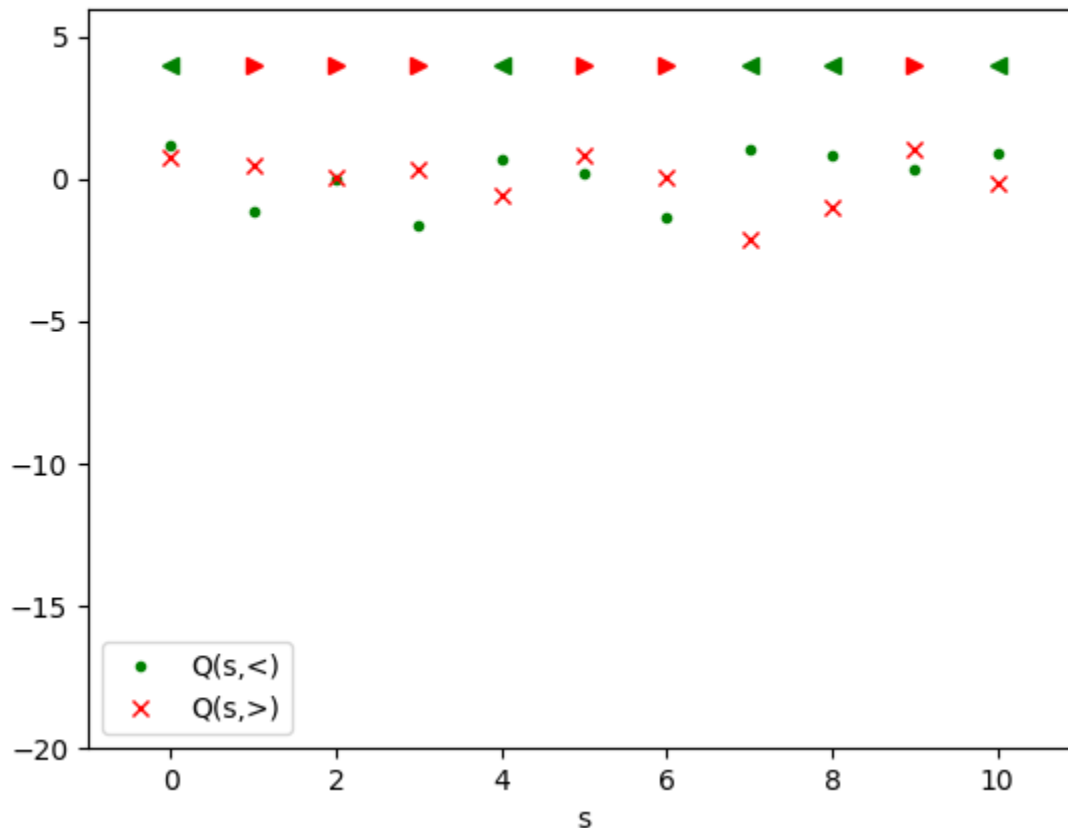
$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$

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

until  $S$  is terminal



# SARSA Control : Résultats



Amélioration  
de  $\pi$

$$\pi = \epsilon \text{ greedy}(\hat{q}_\pi)$$



Evaluation  
de  $\pi$  par  
SARSA  
(1 step)

Remarques d'implémentation :

Les  $q$  values sont initialisées aléatoirement au début.

On explore avec une politique  $\epsilon$  **greedy** avec  $\epsilon$  constant à 0,1.

# n-step TD Learning



Plutôt que de bootstrap au bout d'une étape, on va bootstrap après  $n$  étapes

$$\widehat{v}_{\pi}(s_t) \leftarrow r_t + \gamma r_{t+1} + \dots + \gamma^{n-1} r_{t+n-1} + \gamma^n \widehat{v}_{\pi}(s_{t+n})$$

n-step TD

$$\widehat{q}_{\pi}(s_t, a_t) \leftarrow r_t + \gamma r_{t+1} + \dots + \gamma^{n-1} r_{t+n-1} + \gamma^n \widehat{q}_{\pi}(s_{t+n}, a_{t+n})$$

n-step SARSA

$$\widehat{q}_{\pi}(s_t, a_t) \leftarrow r_t + \gamma r_{t+1} + \dots + \gamma^{T-t} r_T$$

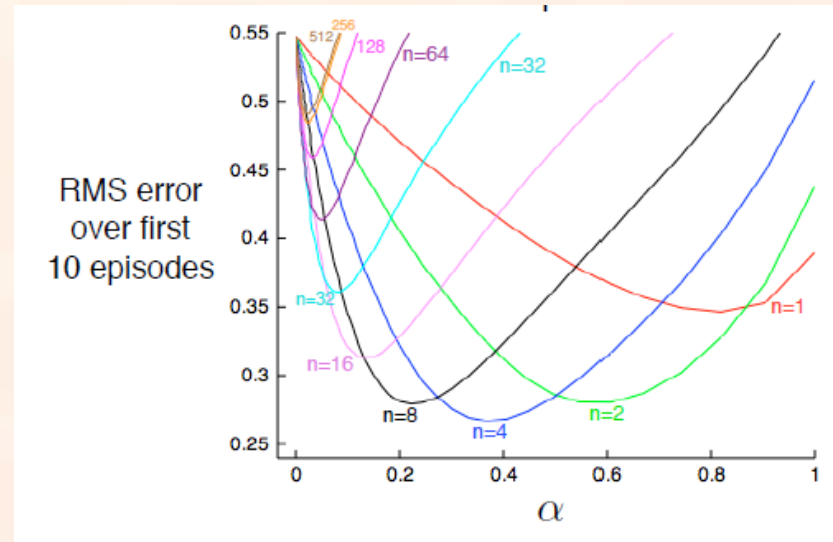
MonteCarlo

( $n \rightarrow +\infty$ )

Augmenter  $n$  a pour effet de :

- Augmenter la variance
- Imposer d'attendre plus longtemps ( $n$  étapes) avant d'apprendre
- Baisser le biais

L'hyperparamètre  $n$  optimal dépend de l'env. et d'autres hyperparamètres.



# TD Learning: Conclusion



## Avantages :

- Pas besoin d'attendre la fin de l'épisode pour apprendre
- Faible variance

## Inconvénients :

- Biaisé car  $\widehat{v}_{\pi}(s_t) \neq v_{\pi}(s_t)$

## Reinforcement Learning

### Model-based

Dynamic  
Programming

### Model-free

Monte  
Carlo  
methods

TD-Learning  
methods



# Off Policy et Q-Learning

# Les limites des politiques exploratives

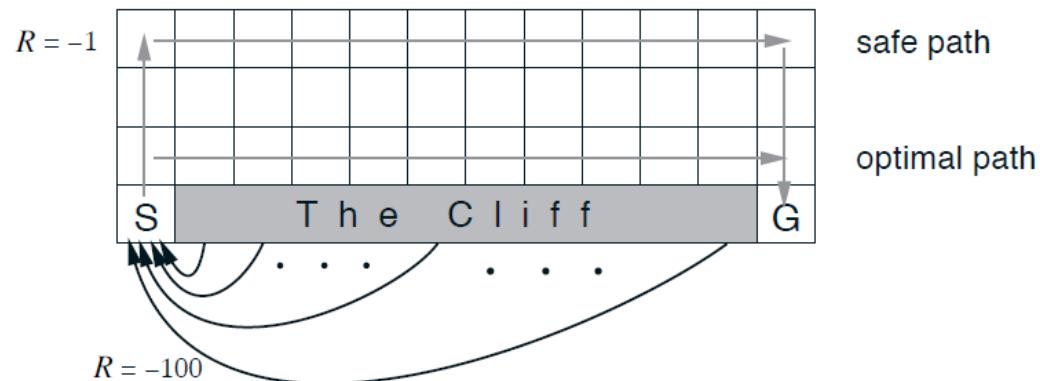


Besoin d'exploration :  $\pi = \varepsilon \text{ greedy}$

Implémenté ainsi, **Monte Carlo Control** et **SARSA Control** entraînent alors  $\pi_{\text{target}} = \varepsilon \text{ greedy}$  vers la meilleure politique explorative.

Or sur certains environnements, la meilleure politique **explorative** est trop prudente et est bien inférieure à la meilleure politique :

Environnement : The Cliff



Solution : dissocier politique **cible**  $\pi$  (à évaluer et améliorer) de politique **comportementale**  $\mu$  (avec qui jouer les épisodes) :

$$\begin{cases} \pi = \text{greedy} \\ \mu = \text{exploration } (\varepsilon \text{ greedy}, \text{UCB}, \dots) \end{cases}$$

Le fait d'utiliser une politique comportementale  $\mu$  différente de la politique qu'on entraîne  $\pi$  constitue l'apprentissage **Off Policy**.



# Off Policy



Déf. : **Off Policy** = utiliser une politique de **comportement**  $\mu$  différente de votre politique à évaluer et optimiser  $\pi$ .

Exemples :

$$\left\{ \begin{array}{l} \pi = \text{greedy}(\widehat{q}_{\pi}), \pi_{\text{quelconque}} \\ \mu = \underbrace{\epsilon \text{ greedy}(\widehat{q}_{\pi}), \text{random}}_{\text{Permet l'exploration}}, \underbrace{\pi_{\text{old}}, \pi_{\text{other agent}}}_{\text{Sample efficiency}} \end{array} \right.$$

Les **algorithmes de RL Off Policy**, de la forme  $\widehat{q}_{\pi}(s, a) \leftarrow X$  vérifient :

$$E_{\mu}[X] = q_{\pi}(s, a)$$

Off Policy ?

$$\widehat{v}_{\pi}(s_t) \leftarrow r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots + \gamma^{T-t} r_T$$

MC

pour estimer  $E_{\pi}[G_t | S_t = s_t]$



$$\widehat{v}_{\pi}(s_t) \leftarrow r_t + \gamma \widehat{v}_{\pi}(s_{t+1})$$

TD(0)

$E_{\pi}[G_t | S_t = s_t]$



$$\widehat{q}_{\pi}(s_t, a_t) \leftarrow r_t + \gamma \widehat{q}_{\pi}(s_{t+1}, a_{t+1})$$

SARSA

$E_{\pi}[G_t | S_t = s_t, A_t = a_t]$



$$\widehat{q}_{\pi}(s_t, a_t) \leftarrow r_t + \gamma \sum_{a'} \pi(a' | s_{t+1}) \widehat{q}_{\pi}(s_{t+1}, a')$$

SARSA-Expected

$E_{\pi}[G_t | S_t = s_t, A_t = a_t]$



# Experience Replay : apprendre des exp. passées

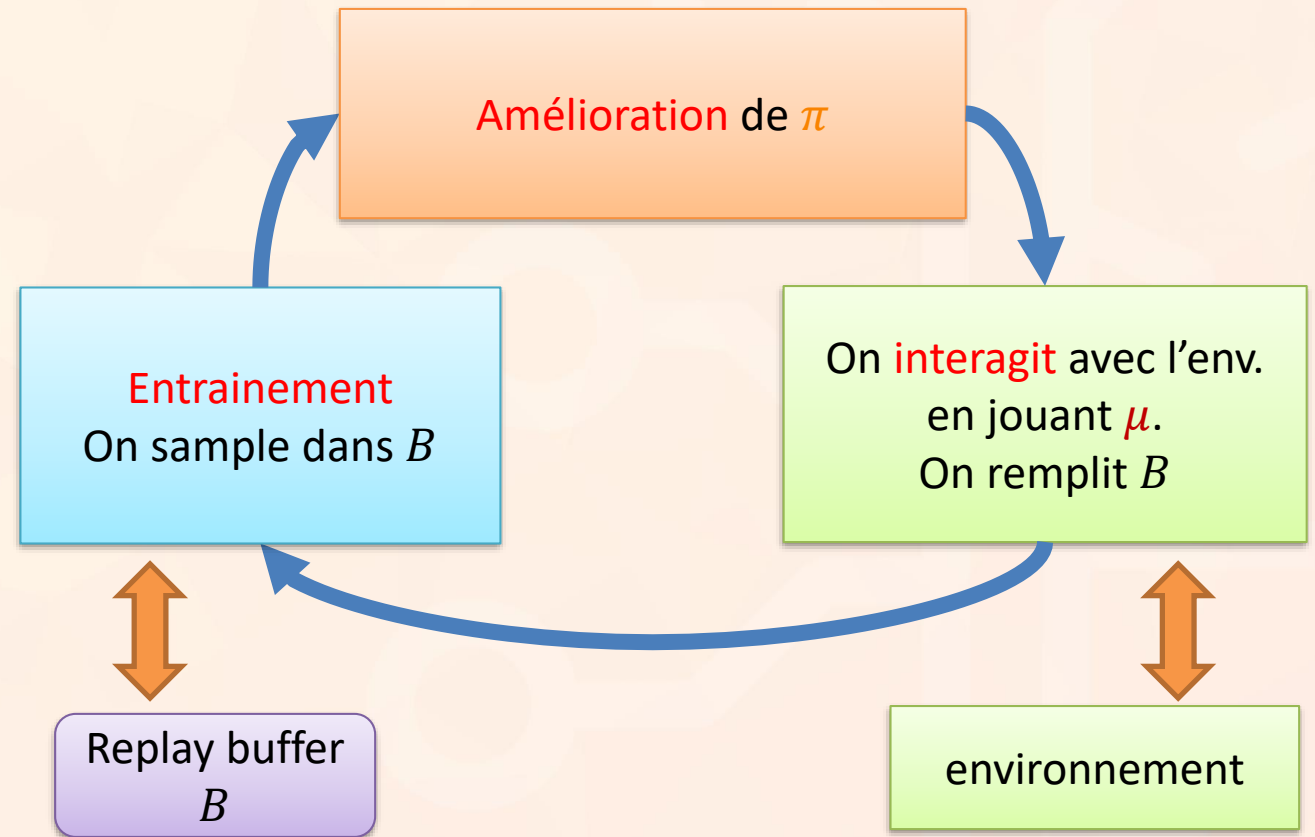


L'Off Policy permet de réutiliser les transitions qui ont été tirées avec d'anciennes politiques.

**Experience Replay** : On ne va pas apprendre qu'une fois de chaque transition  $(s_t, a_t, r_t, s_{t+1})$  : on va les stocker dans une mémoire  $B$  appelée *replay buffer*.

Interêts :

- **Sample efficiency** : on utilise plusieurs fois chaque transition tirée
- **Décorrélation** : Les transitions tirées de  $B$  sont décorellées
- **Parallélisation** : on va pouvoir entrainer en même temps qu'on joue (de manière parallèle)
- **Évite le catastrophic forgetting** (càd oublier les informations extraites des plus anciennes transitions)



# Q Learning



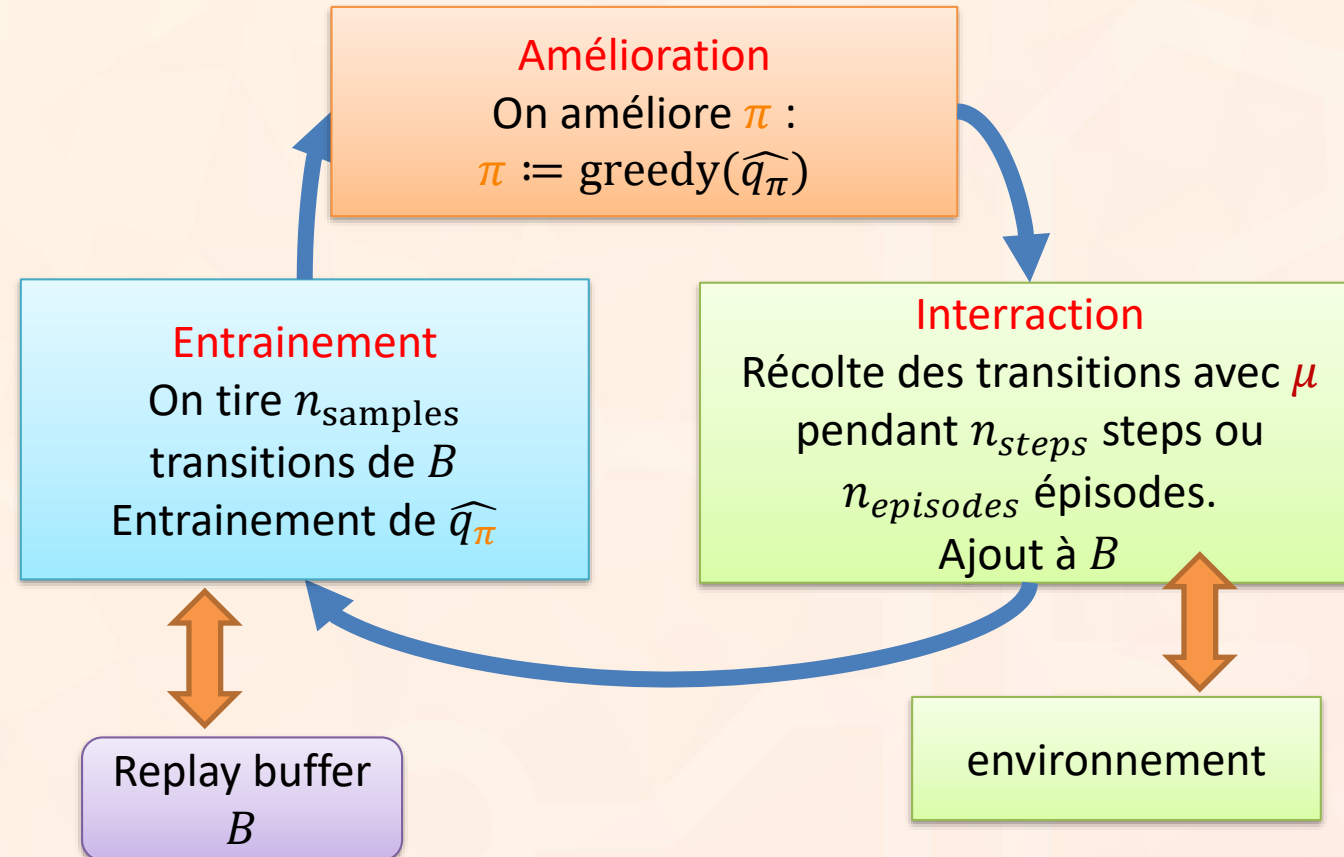
Dans le cas de  $\pi = \text{greedy}(\widehat{q}_\pi)$ , **SARSA-Expected** correspond à :

$$\widehat{q}_\pi \leftarrow r_t + \gamma \max_{a'} \widehat{q}_\pi(s_{t+1}, a') \quad \text{Q Learning}$$

Cet algorithme est qui a l'avantage d'être **Off Policy**, **online** et de faible variance est connu sous le nom de **Q Learning**.

$$\begin{cases} \pi = \text{greedy}(\widehat{q}_\pi) \\ \mu = \epsilon \text{greedy}(\widehat{q}_\pi) \text{ (old)} \end{cases}$$

Q Learning loop :



Remarque : l'équation du Q Learning peut se voir comme une application de l'équation de Bellman optimale.

# Deep Reinforcement Learning



## Cas tabulaire :

s	0	1	...	$n_{state} - 1$
$\widehat{v}_{\pi}(s)$	0,00	-0,98	...	-7,96

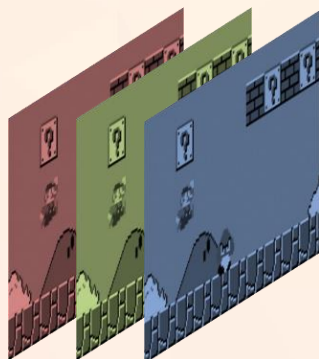
Apprentissage :

$$\widehat{v}_{\pi}(s) := \widehat{v}_{\pi}(s) + \alpha(X - \widehat{v}_{\pi}(s))$$

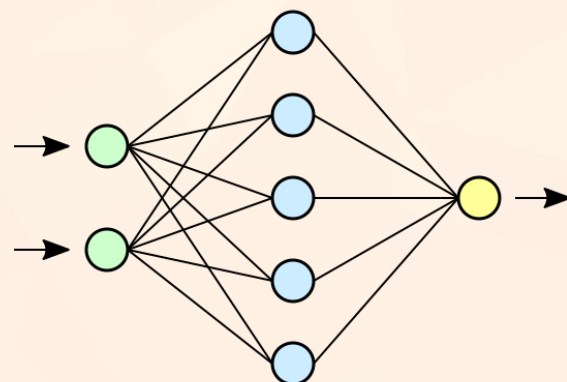
Comment faire quand les espaces d'observations (et d'actions) sont trop grands ?

## Deep RL :

State  $s$



Neural Network  
Weights :  $\varphi$



Estimated  
State Value

$$\widehat{v}_{\pi}^{\varphi}(s)$$

Apprentissage :

$$\varphi := \varphi - \alpha \nabla_{\varphi}(\text{Loss}(\widehat{v}_{\pi}^{\varphi}(s), X))$$

Avec  $\varphi$  paramètres de  $\widehat{v}_{\pi}^{\varphi}(s)$



Le Deep Learning est puissant mais mal compris

Plutôt que de voir une image comme un état parmi  $256^{3HW}$  on la voit comme un vecteur dans  $[0; 255]^{3HW}$

# Combinons tout ça : Deep Q Network (DQN)



---

**Algorithm 1** Deep Q-learning with Experience Replay

---

Initialize replay memory  $\mathcal{D}$  to capacity  $N$

Initialize action-value function  $Q$  with random weights

**for** episode = 1,  $M$  **do**

    Initialize sequence  $s_1 = \{x_1\}$  and preprocessed sequenced  $\phi_1 = \phi(s_1)$

**for**  $t = 1, T$  **do**

        With probability  $\epsilon$  select a random action  $a_t$

        otherwise select  $a_t = \max_a Q^*(\phi(s_t), a; \theta)$

        Execute action  $a_t$  in emulator and observe reward  $r_t$  and image  $x_{t+1}$

        Set  $s_{t+1} = s_t, a_t, x_{t+1}$  and preprocess  $\phi_{t+1} = \phi(s_{t+1})$

        Store transition  $(\phi_t, a_t, r_t, \phi_{t+1})$  in  $\mathcal{D}$

        Sample random minibatch of transitions  $(\phi_j, a_j, r_j, \phi_{j+1})$  from  $\mathcal{D}$

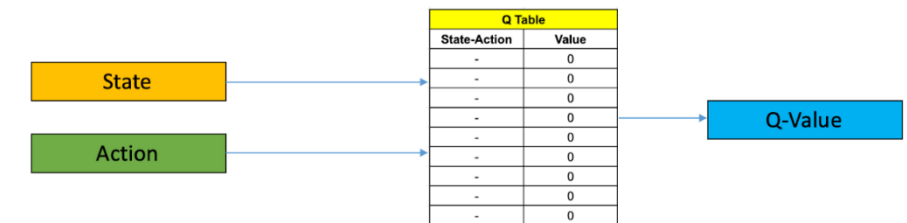
        Set  $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$

        Perform a gradient descent step on  $(y_j - Q(\phi_j, a_j; \theta))^2$  according to equation 3

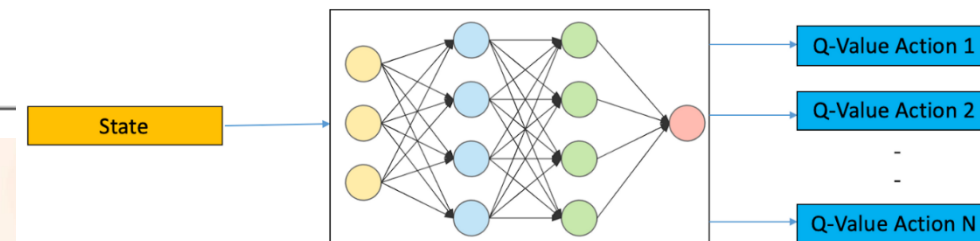
**end for**

**end for**

---



Q Learning



Deep Q Learning

## Playing Atari with Deep Reinforcement Learning



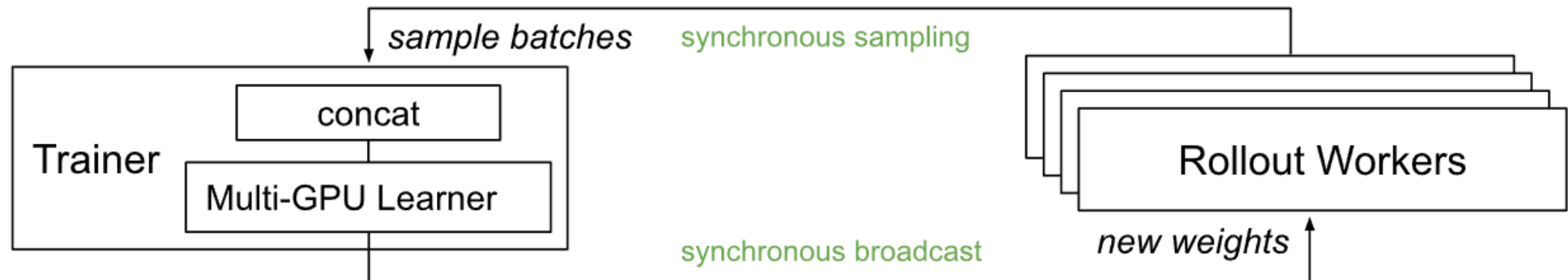
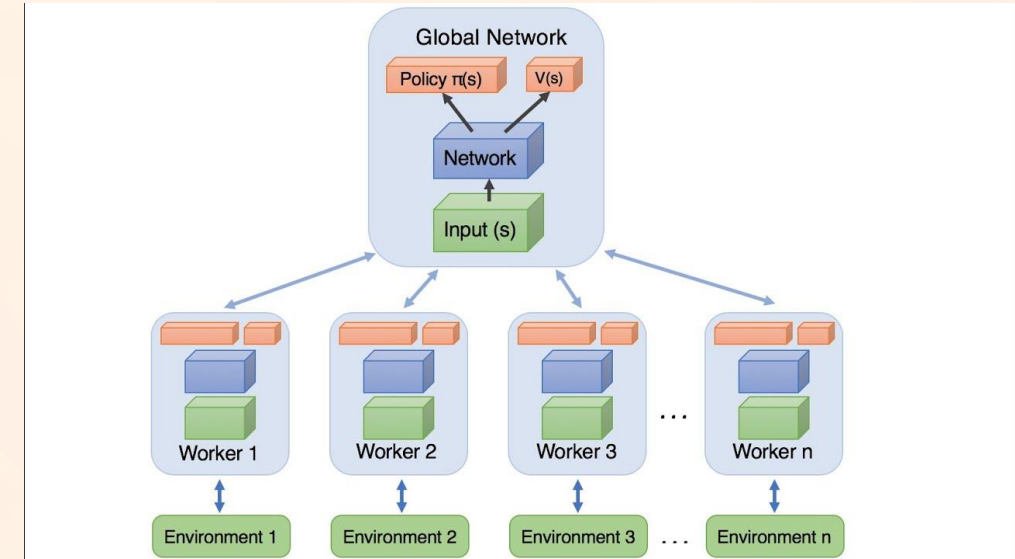
# Parallélisation en RL



Problème : en travaillant on-policy, on obtient des données issus des mêmes épisodes donc corrélés, ce qui augmente la variance.

La parallélisation permet :

- de décorrélérer les batch de données, ce qui réduit la variance
- de run plusieurs environnements en même temps (gain de temps)
- de vectoriser les données (gain de temps via l'usage de GPUs)





# Taxonomie du RL



## Reinforcement Learning

### Model-based

#### Dynamic Programming

- Bellman equations
- Iterative Policy Eval.
- Policy Iteration
- Value Iteration

### Model-free

#### Policy based RL

REINFORCE

Actor Critic  
algorithms

#### Value based RL

##### TD-Learning methods

- TD(0)
- SARSA
- n-step methods
- **Q Learning**

Monte  
Carlo  
methods

# Problèmes généraux du RL



- Instabilité due à la variance des  $G_t$
- Instabilité due à l'entraînement simultané de plusieurs réseaux
- Peu de garanties de convergence dans le cas du Deep RL
- Grande sensibilité aux hyperparamètres (qui sont nombreux)

# Pour aller plus loin : les bibliothèques de RL



Pour les environnements :



Gym

Pour les agents :



StableBaselines3 :

- simple et rapide en application
- choix par défaut



RLlib :

- adapté au multi-agent et au hiérarchique
- scalable (construit sur Ray, librairie de parallélisation)
- à privilégier pour les gros projets

D'autres bibliothèques : CleanRL, ML agent (Unity), OpenAI Baselines, KerasRL ...

# Pour aller plus loin : ressources en RL



Reinforcement Learning : an introduction, Sutton & Barto

Cours de DeepMind 2021 sur le RL : <https://dpmd.ai/DeepMindxUCL21>

[Playing Atari with Deep Reinforcement Learning, DeepMind, 2013](#)

Spinning Up (ressource educative pour ceux voulant apprendre le RL) :

<https://spinningup.openai.com/en/latest/>

Blog de Lilian Weng sur le RL: <https://lilianweng.github.io>

Value-based RL basics : [Medium article](#)

Policy-based RL basics : [Medium article](#)



Fin  
Des questions ?

# Sources et ressources :



Reinforcement Learning : an introduction, Sutton & Barto

Cours de DeepMind 2021 sur le RL : <https://dpmd.ai/DeepMindxUCL21>

Blog de Lilian Weng : <https://lilianweng.github.io>

GitHub formation (slides & code) : [github.com/tboulet/Formation-Reinforcement-Learning](https://github.com/tboulet/Formation-Reinforcement-Learning)





# ANNEXE

# Off Policy par Importance Sampling



Il est possible de transformer un algo on-policy en un algorithme off-policy si on connaît  $\pi$  et  $\mu$

**Importance Sampling** : estimer l'espérance d'une distribution à partir d'échantillon issus d'une distribution différente :

$$\mathbb{E}_p[f(\mathbf{x})] = \int p(\mathbf{x}) f(\mathbf{x}) d\mathbf{x} = \int q(\mathbf{x}) \left[ \frac{p(\mathbf{x})}{q(\mathbf{x})} f(\mathbf{x}) \right] d\mathbf{x} = \mathbb{E}_q \left[ \frac{p(\mathbf{x})}{q(\mathbf{x})} f(\mathbf{x}) \right]$$

Application au RL, exemple avec  $R_0$ :

$$E_{\pi}[R_0] = \sum_{a_0} \pi(a_0|s_0) R_{s_0}^{a_0} = \sum_{a_0} \mu(a_0|s_0) \frac{\pi(a_0|s_0)}{\mu(a_0|s_0)} R_{s_0}^{a_0} = E_{\mu} \left[ \frac{\pi(A_0|S_0)}{\mu(A_0|S_0)} R_0 \right]$$



Haute variance

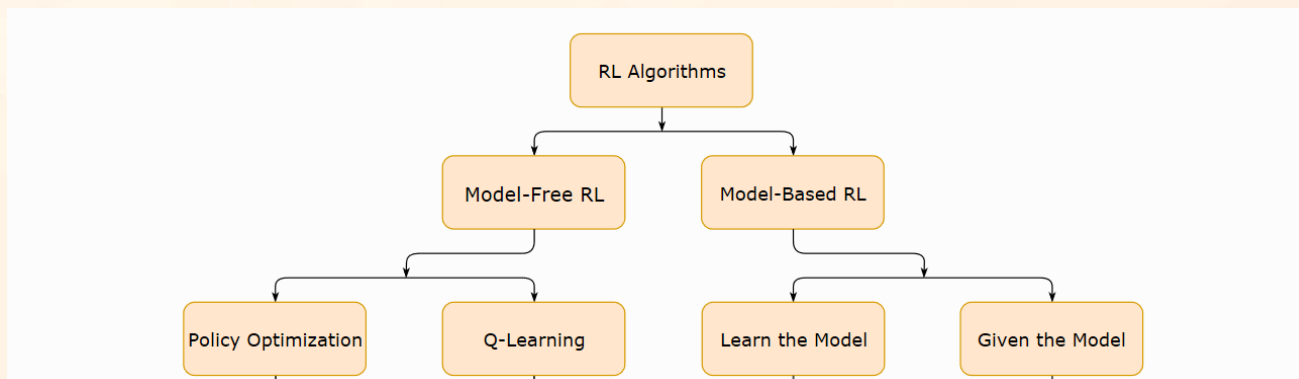
**Interprétation** : Si  $\tau$  (obtenue avec  $\mu$ ) est plus probable d'arriver avec  $\mu$  qu'avec  $\pi$ , il est logique qu'elle pèse moins dans le calcul de  $\hat{E}_{\pi}[f(x)]$ .

**TD(0) Off Policy** :  $\widehat{v}_{\pi}(s_t) \leftarrow (R_t + \gamma \widehat{v}_{\pi}(s_{t+1})) * \frac{\pi(a_t|s_t)}{\mu(a_t|s_t)}$



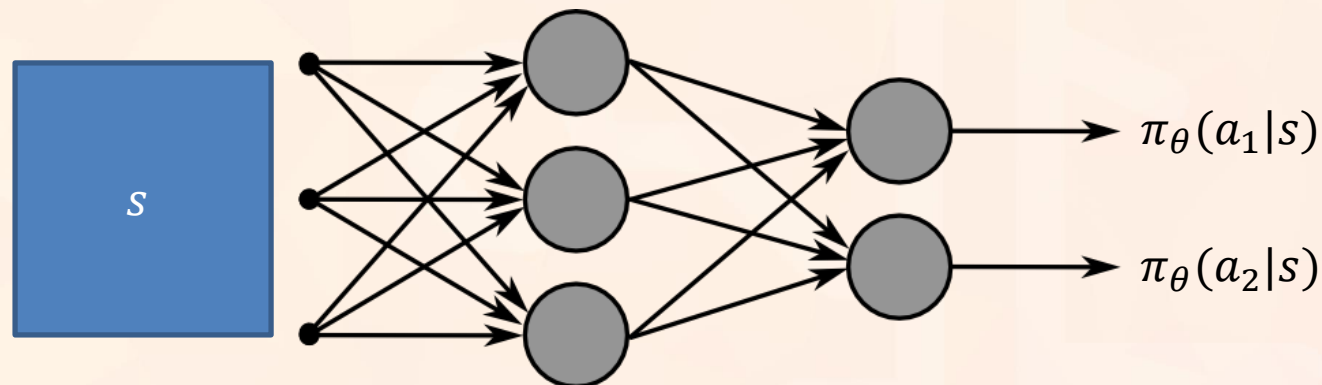
# Policy-based RL

# Policy Gradients : le principe



Politique  $\pi_\theta$  paramétrée:

$$\begin{aligned}\pi_\theta: S &\rightarrow [0,1]^{n_{actions}} \\ S &\rightarrow (\pi_\theta(a_k|s))_{1 \leq k \leq n}\end{aligned}$$



But : définir une fonction objectif  $J(\theta)$  différentiable par rapport à  $\theta$  pour faire une montée de gradient :

$$\theta := \theta + \alpha \nabla_\theta J(\theta)$$

Remarque : plutôt qu'une distribution d'action discrète en sortie on peut avoir une distribution continue  $\pi_\theta(s) = (m, \sigma)$   
On peut aussi utiliser une politique déterministe  $\pi_\theta(s) = a$

# Policy Gradients : la théorie



Fonction objectif :

$$J(\theta) = E_{\pi_\theta}[G_0] = \int_{\tau} G_0(\tau) \rho_{\pi_\theta}(\tau) d\tau$$

$$E[X] = \int_{\omega} X(\omega) \rho(\omega) d\omega \quad X \rightarrow (\Omega, A, \rho)$$

$$\text{Avec } \rho_{\pi_\theta}(\tau) = C_\tau \prod_{t=0}^T \pi_\theta(a_t | s_t)$$

Calcul du gradient :

$$\nabla_{\theta} J(\theta) = \nabla_{\theta} \int_{\tau} G_0(\tau) \rho_{\pi_\theta}(\tau) d\tau = \int_{\tau} G_0(\tau) \nabla_{\theta} \rho_{\pi_\theta}(\tau) d\tau = \int_{\tau} G_0(\tau) \rho_{\pi_\theta}(\tau) \nabla_{\theta} \ln(\rho_{\pi_\theta}(\tau)) d\tau = E_{\pi_\theta}[G_0 \nabla_{\theta} \ln(\rho_{\pi_\theta})]$$

$\nabla \ln(u) = \frac{\nabla u}{u}$

Estimation empirique de  $\nabla_{\theta} J(\theta)$  :

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_i \left[ G_0^i * \sum_{t=0}^{T_i} \nabla_{\theta} \ln \pi_{\theta}(a_t^i | s_t^i) \right] \quad \text{REINFORCE}$$

Problème de **causalité** : les premières rewards (dans  $G_0^i$ ) ont ici une influence sur les gradients des dernières actions

Solution : on fait passer  $G_0^i$  dans la somme et on enlève les rewards précédents  $t'$ .



# Policy Gradients : les problèmes



Estimation empirique de  $\nabla_{\theta} J(\theta)$  causal :

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_i \sum_{t=0}^{T_i} \gamma^t G_t^i \nabla_{\theta} \ln \pi_{\theta}(a_t^i | s_t^i) \quad \text{REINFORCE}$$

## REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for $\pi_*$

Input: a differentiable policy parameterization  $\pi(a|s, \theta)$

Algorithm parameter: step size  $\alpha > 0$

Initialize policy parameter  $\theta \in \mathbb{R}^{d'}$  (e.g., to  $\mathbf{0}$ )

Loop forever (for each episode):

Generate an episode  $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$ , following  $\pi(\cdot | \cdot, \theta)$

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

$$G \leftarrow \sum_{k=t+1}^T \gamma^{k-t-1} R_k$$

$$\theta \leftarrow \theta + \alpha \gamma^t G \nabla \ln \pi(A_t | S_t, \theta)$$

( $G_t$ )

Problème de **variance** : les policy gradients souffrent de gros problèmes de variance

Le fait d'avoir une mesure non-centrée d'à quel point l'action est bonne ( $G_t^i$ ) rend l'apprentissage instable.

Solution : ajout d'une baseline pour centrer cette mesure :

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_i \sum_{t=0}^{T_i} \gamma^t (G_t^i - b(s_t)) \nabla_{\theta} \ln \pi_{\theta}(a_t^i | s_t^i)$$

ajout d'une baseline

**On-policy** : ces algorithmes améliorent  $\pi$  à partir d'elle-même, et sont donc nécessairement **On-policy**.



# Policy Gradients : ajout d'une baseline



Ajout d'une baseline :

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_i \sum_{t=0}^{T_i} \gamma^t (G_t^i - b(s_t)) \nabla_{\theta} \ln \pi_{\theta}(a_t^i | s_t^i)$$

ajout d'une baseline

Effet : réduction de la variance sans pour autant augmenter le biais :

$$\nabla_{\theta} \int_{\tau} b(s_t) \rho_{\pi_{\theta}}(\tau) d\tau = b(s_t) \nabla_{\theta} \int_{\tau} \rho_{\pi_{\theta}}(\tau) d\tau = b(s_t) \nabla_{\theta} (1) = 0$$

Choix pour  $G_t^i - b(s)$  :

- $G_t - \widehat{v}_{\pi}(s_t)$
- $R_t + \gamma \widehat{v}_{\pi}(s_t) - \widehat{v}_{\pi}(s_{t+1})$
- $\widehat{q}_{\pi}(s_t, a_t)$
- $\widehat{q}_{\pi}(s_t, a_t) - \widehat{v}_{\pi}(s_t) := \widehat{A}_{\pi}(s_t, a_t)$  (advantage function)

L'idéal est un estimateur non biaisé et à faible variance de  $A$ .

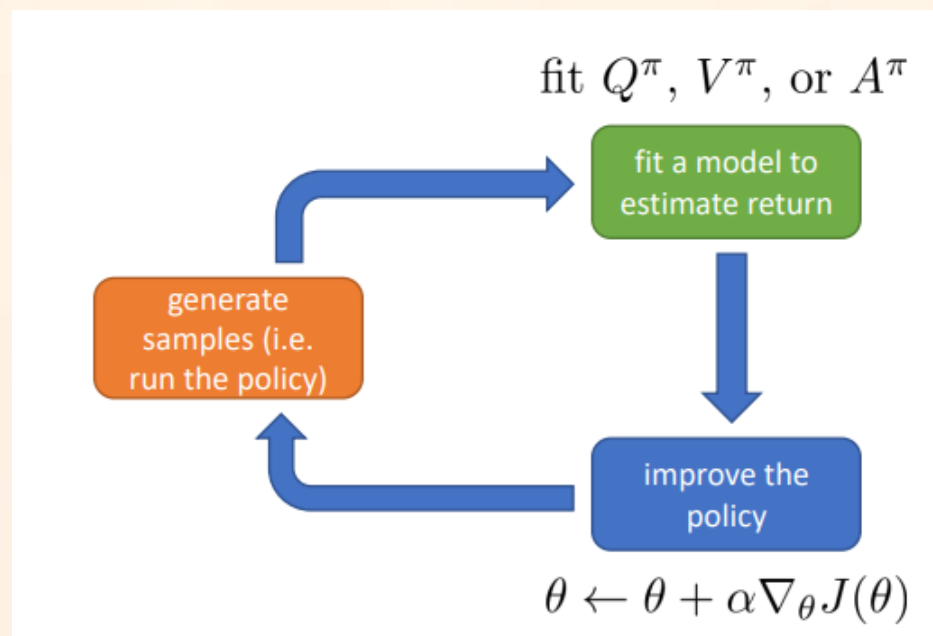
Le choix est un trade-off biais/variance/online

# Les algorithmes Actor Critic



Les Actor Critics sont des algorithmes qui entraînent à la fois une politique  $\pi$  (l'actor) ainsi qu'un "critic" :  $v$  et/ou  $q$  qui va aider à entraîner  $\pi$ .

Actor-Critic training loop



Entraînement de l'actor  $\pi_\theta$  :

$$\theta := \theta + \alpha \sum_{t'=0}^T \gamma^{t'} (G_{t'} - \widehat{v}_\pi^\phi(s)) * \nabla_\theta \ln \pi_\theta(a_{t'} | s_{t'})$$

Entraînement du critic  $\widehat{v}_\pi^\phi$  :

$$\phi := \phi - \alpha' \nabla_\phi (\text{Loss}(\widehat{v}_\pi^\phi(s), G_t))$$