

Intelligence artificielle

Reinforcement Learning

Hatem Ghorbel & Stefano Carrino

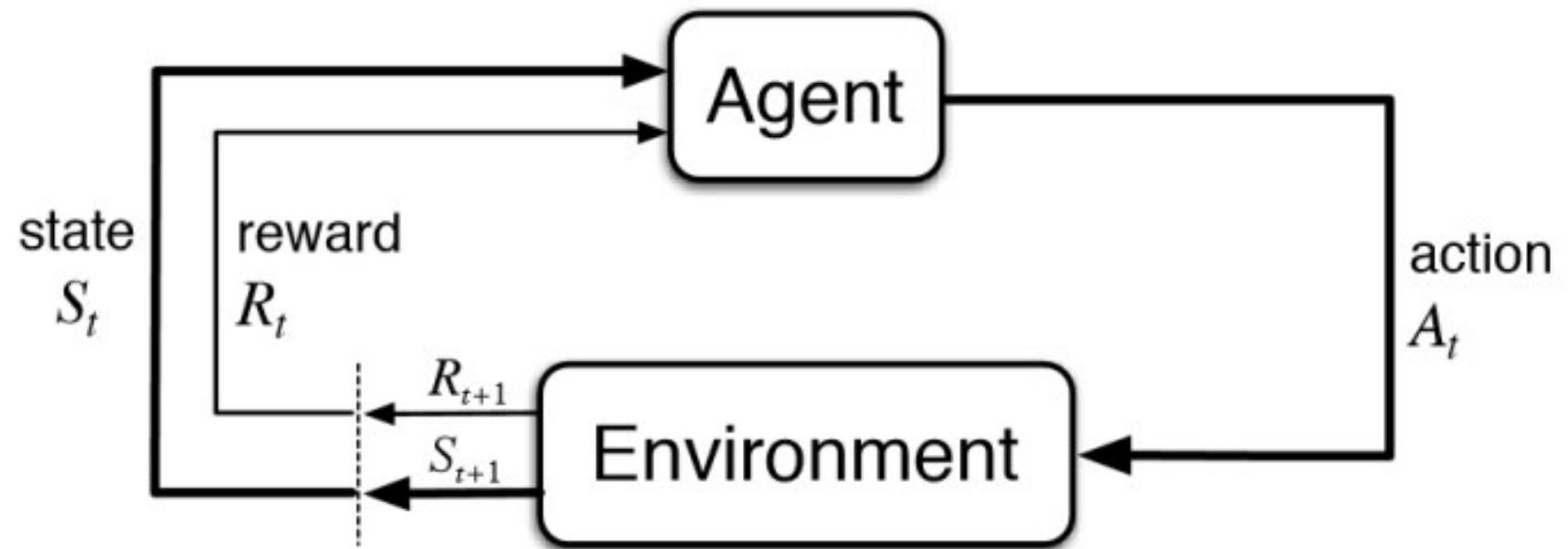
L'apprentissage par renforcement

- A aujourd'hui, la branche de l'IA la plus proche de la vraie ***Intelligence Artificielle***

Principes

- L'apprentissage par renforcement est un type de technique d'apprentissage automatique qui permet à un agent **d'apprendre** dans un environnement interactif **par essais et erreurs** en utilisant le retour d'informations de ses propres actions et expériences.
- Dans les approches standards d'apprentissage on apprend grâce à une base de données disponible
- Dans l'apprentissage par renforcement l'apprentissage se fait grâce à une interaction continue avec l'environnement
 - Cette interaction peut être réelle ou simulée

Principles



Hall of fame



alphago (2015) => alphazero (2017)



Alphastar (2019)



OpenAI Five (2019)

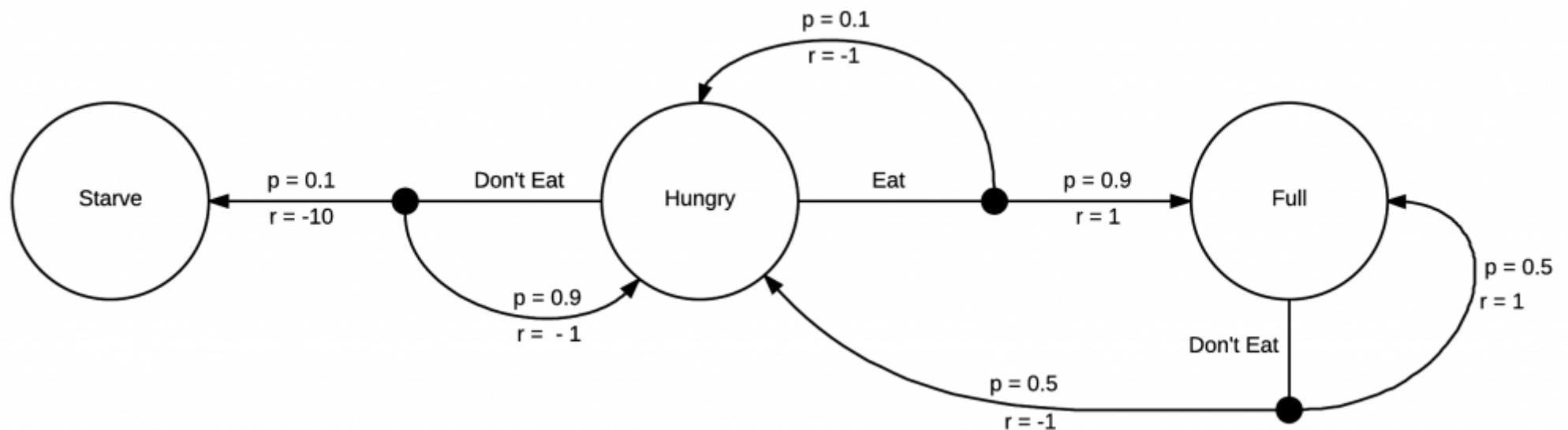
Lien: <https://openai.com/blog/emergent-tool-use/>

Glossaire

- Voici quelques termes clés décrivant les éléments d'un problème RL
 - **Environnement** (*environment*): monde «physique» dans lequel l'agent opère
 - Actions
 - Observations
 - **Etat** (*state*) : situation actuelle de l'agent
 - **Récompense** (*reward*) : retour d'information de l'environnement
 - **Stratégie** (*policy*) : méthode permettant de mapper l'état de l'agent aux actions (voir prochain slide)
 - **Valeur** (*value*) : récompense future qu'un agent recevrait en prenant une mesure dans un état particulier
 - **Step Vs Episode** : ex. échecs: un mouvement d'une pièce vs une partie

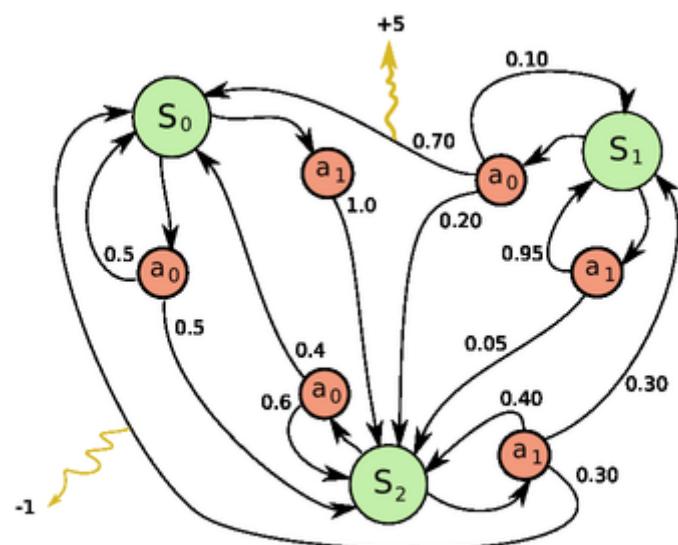
Policy

Une *policy* (stratégie), écrite $\pi(s, a)$, décrit une façon d'agir. C'est une fonction qui prend un état et une action et retourne la probabilité de faire une action dans cet état. Par conséquent, pour un état donné, il doit être vrai que $\sum \pi(s, a) = 1$. Dans l'exemple ci-dessous, lorsque nous avons faim, nous pouvons choisir entre deux actions, manger ou ne pas manger.



Markovian Decision Process - MDP

- Processus de décision markovien
 - est un modèle stochastique où un agent prend des décisions et où les résultats de ses actions sont aléatoires (= suivent une certaine loi statistique)

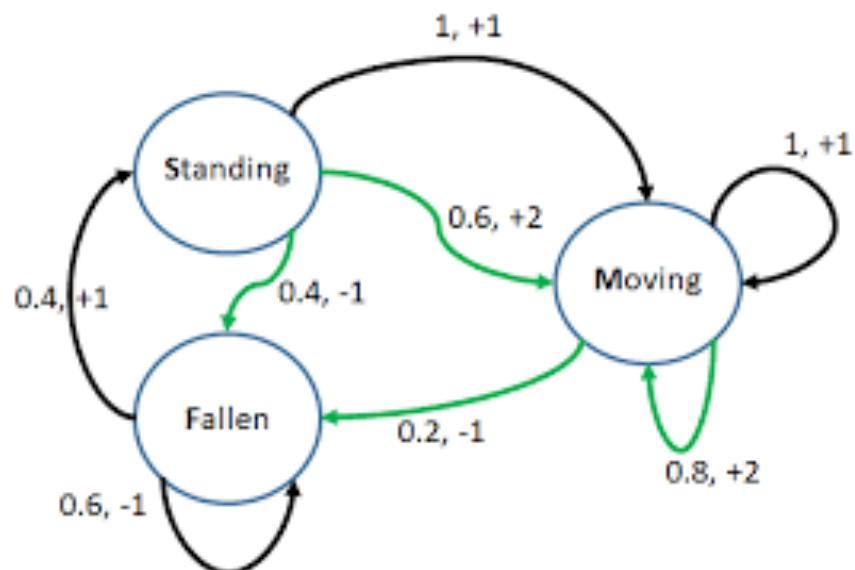


Exemple de **processus de Décision Markovien** à trois états et à deux actions.

Source: https://fr.wikipedia.org/wiki/Processus_de_d%C3%A9cision_markovien

Markovian Decision Process - MDP

- Processus de décision markovien
 - est un modèle stochastique où un agent prend des décisions et où les résultats de ses actions sont aléatoires



Exemple de processus de Décision Markovien :
un robot qui essaye de se déplacer.
- Flèche verte : action agressive
- Flèche noire : action lente

Exercice



- Compléter, pour l'image sur la gauche (état courant)
 - Environnement :
 - Etat :
 - Récompense :
 - Stratégie :
 - Valeur (*value*):
 - Episode:

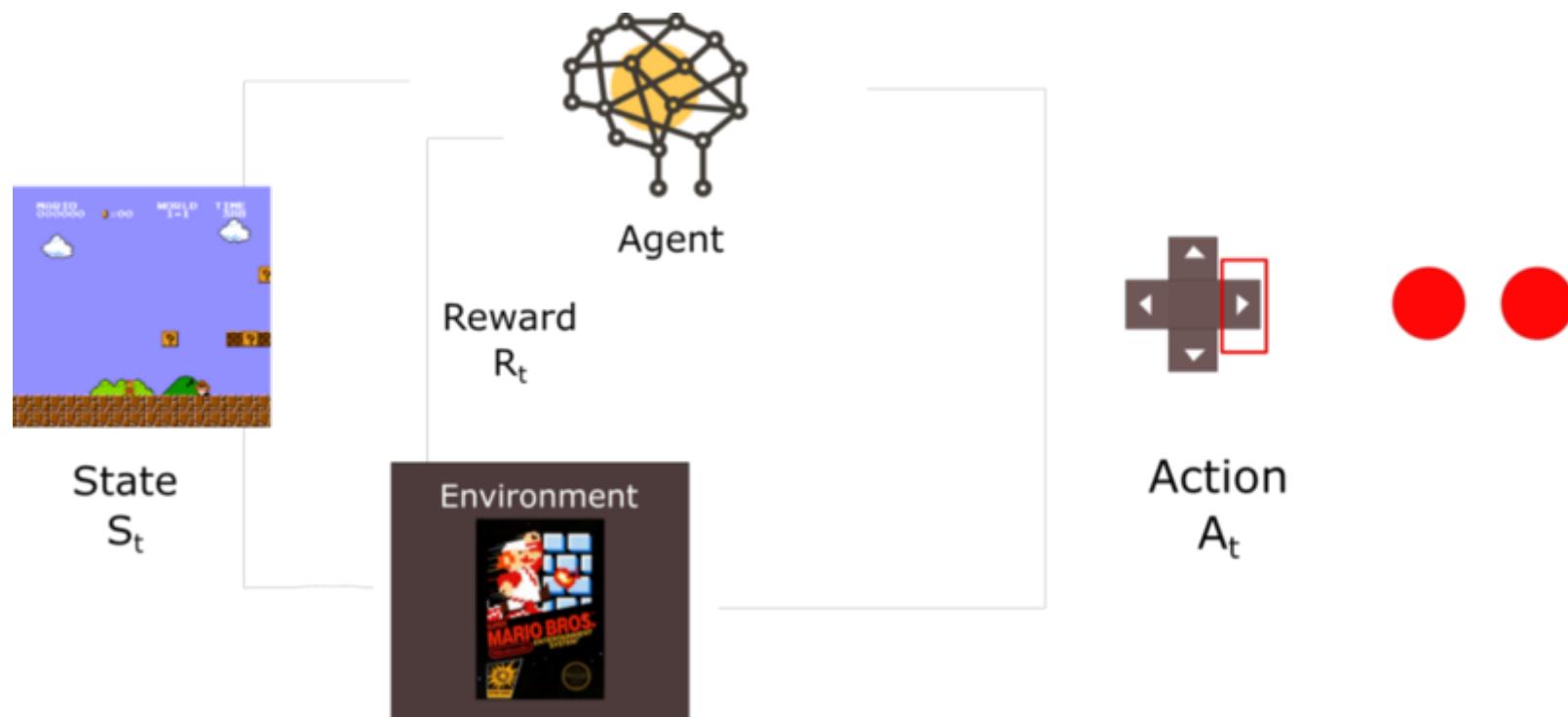
Exercice

- Dessiner le MDP pour le problème suivant
 - But : Un robot pirate doit trouver le trésor (sans mourir).
 - Actions possible : UP, LEFT, RIGHT and DOWN (si valides).
 - Considérez qu'une rafale de vent peut à tout moment changer le résultat d'une action, en poussant le robot d'une case vers le haut par rapport au résultat attendu (20% de chance).
- Rewards :
 - Chaque pas : -0.1
 - Mort : -1
 - Trésor : +1



↑
Direction
du vent

RL - Algorithme



Goal : maximize the **expected cumulative reward**

- Maximiser la récompense (possible) future

$$G_t = R_{t+1} + R_{t+2} + \dots$$

$$G_t = \sum_{k=0}^T R_{t+k+1}$$

Goal : maximize the **expected cumulative reward**

- Toutes récompenses ont la même valeur ?

*Mieux vaut un œuf aujourd'hui
ou un poulet demain ?*

*Mieux vaut un œuf aujourd'hui
ou un poulet après-demain ?*

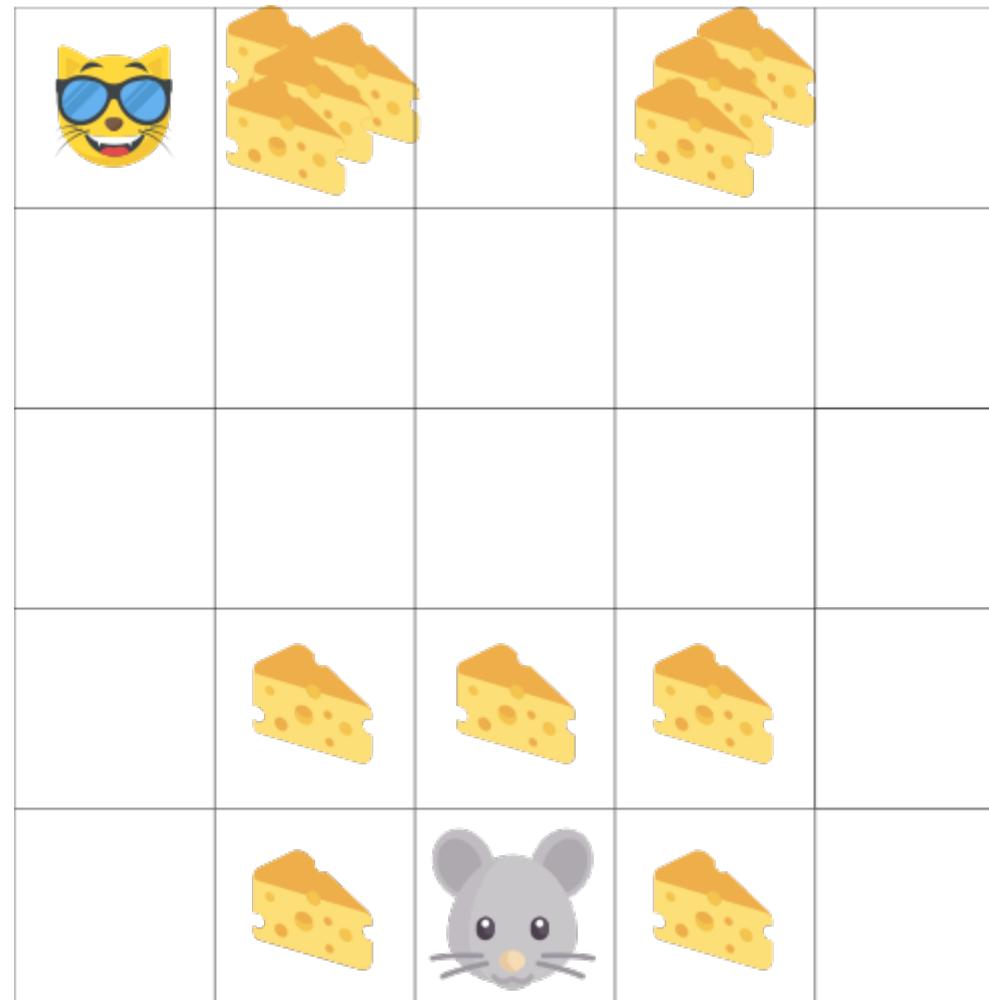
...

*Mieux vaut un œuf aujourd'hui
ou un poulet dans 1 année ?*

Goal : maximize the **expected cumulative reward**

- Les récompenses qui viennent plus tôt (ex. au début d'un jeu) sont plus probables, car elles sont plus prévisibles que la récompense à long terme.

Goal : maximize the **expected cumulative reward**



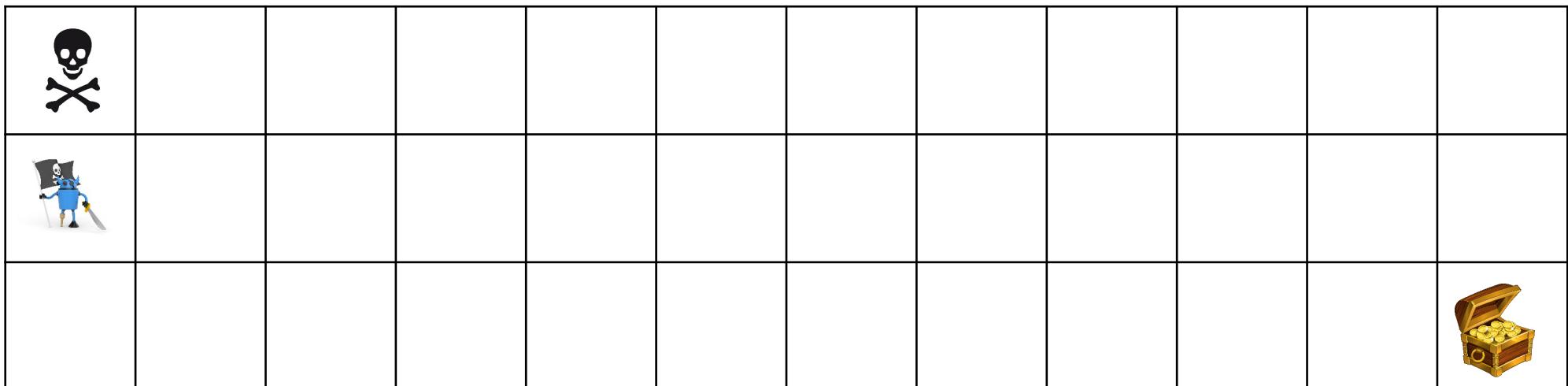
Goal : maximize the **discounted** expected cumulative reward

- Les récompenses qui viennent plus tôt sont plus probables, car elles sont plus prévisibles que la récompense à long terme... **et donc plus intéressantes**

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \text{ where } \gamma \in [0, 1)$$

$$R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \dots$$

Goal : maximize the **cumulative reward**



Rewards :

- Chaque pas : -0.1
- Mort : -1
- Trésor : +1

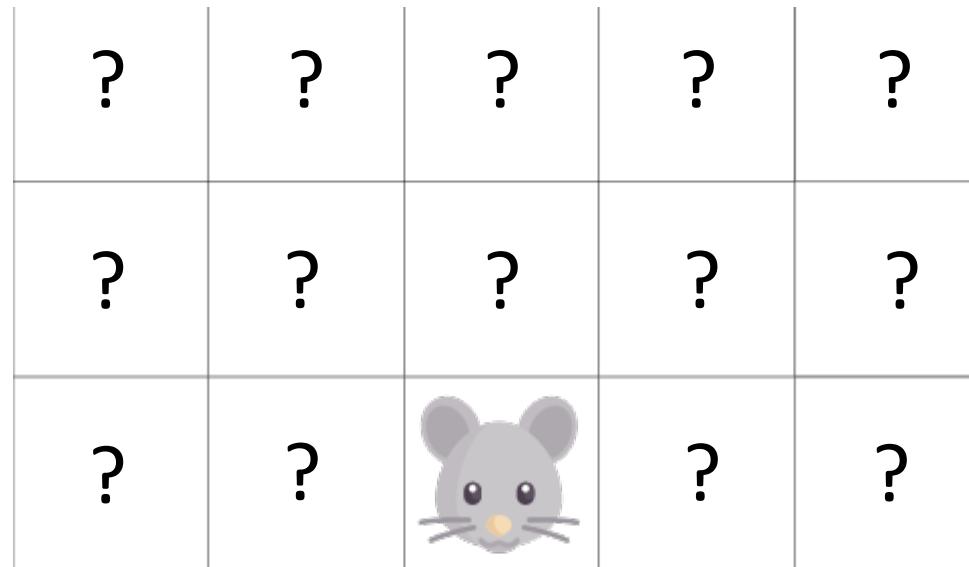
Attention !

Très important de bien étudier l'impact de chaque reward.

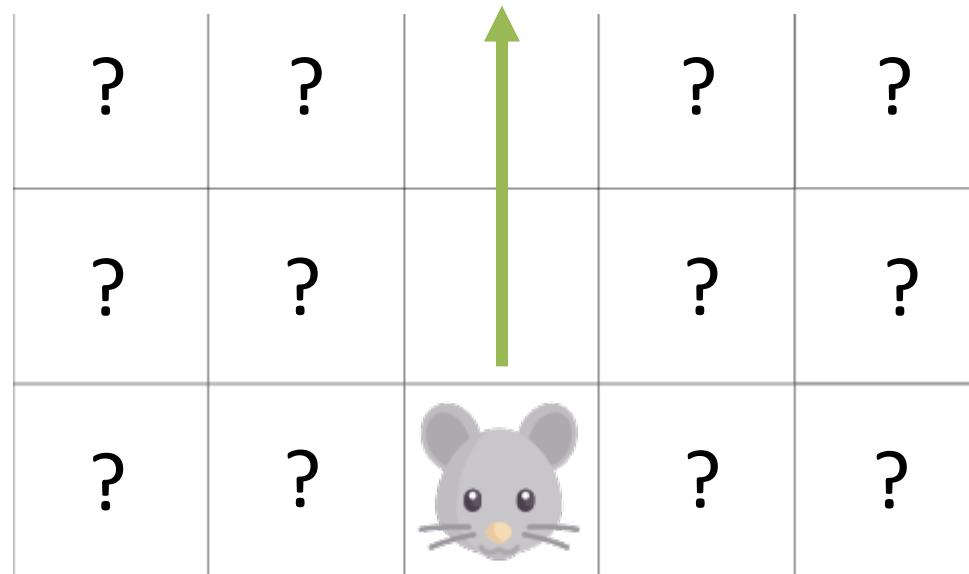
Exploration/Exploitation trade-off

- L'**exploration** consiste à trouver plus d'informations sur l'environnement.
- L'**exploitation** consiste à exploiter des informations connues pour maximiser la récompense.
- N'oubliez pas que l'objectif de notre agent RL est de maximiser la récompense cumulative attendue !

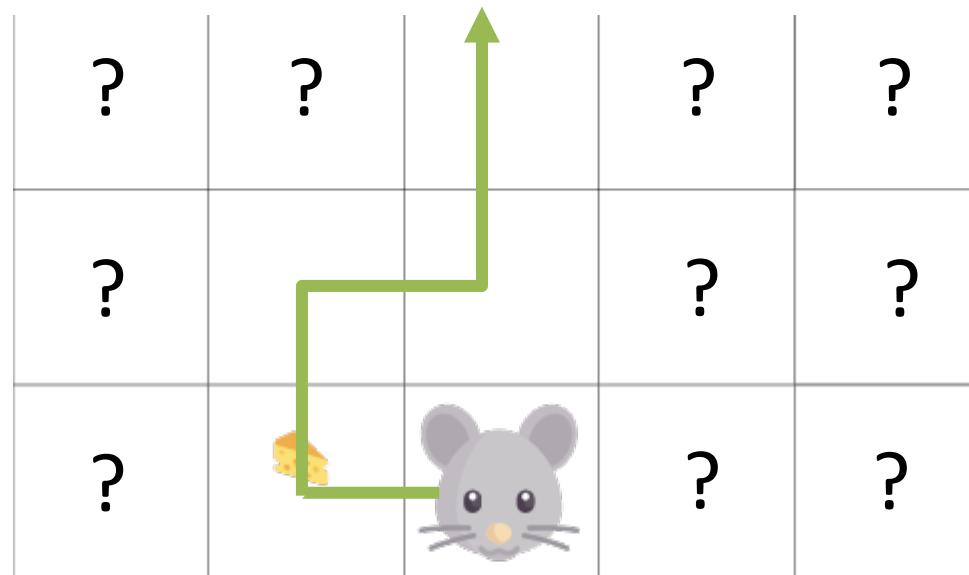
Exploration/Exploitation trade-off



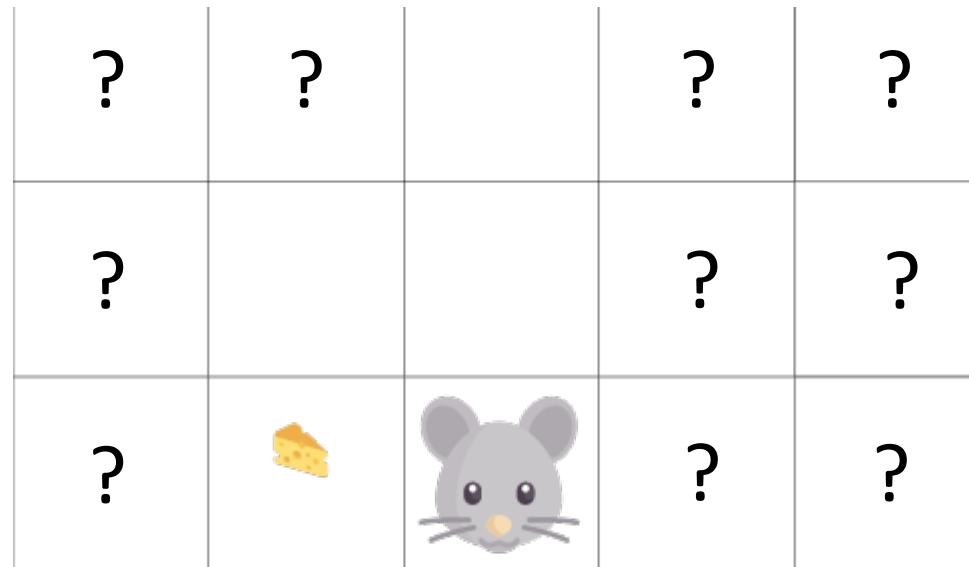
Exploration/Exploitation trade-off



Exploration/Exploitation trade-off



Exploration/Exploitation trade-off



Quelle action choisir au prochain épisode ?

Left: reward +1

Up: reward 0

Right: ?

Exploration/Exploitation trade-off

- **Epsilon-greedy algorithm**
 - Nous spécifions un taux d'exploration « epsilon », qui varie entre 0 et 1
 - 1 => actions totalement aléatoires
 - 0 => actions totalement greedy (avare)

Exploration/Exploitation trade-off

- **Epsilon-greedy algorithm**

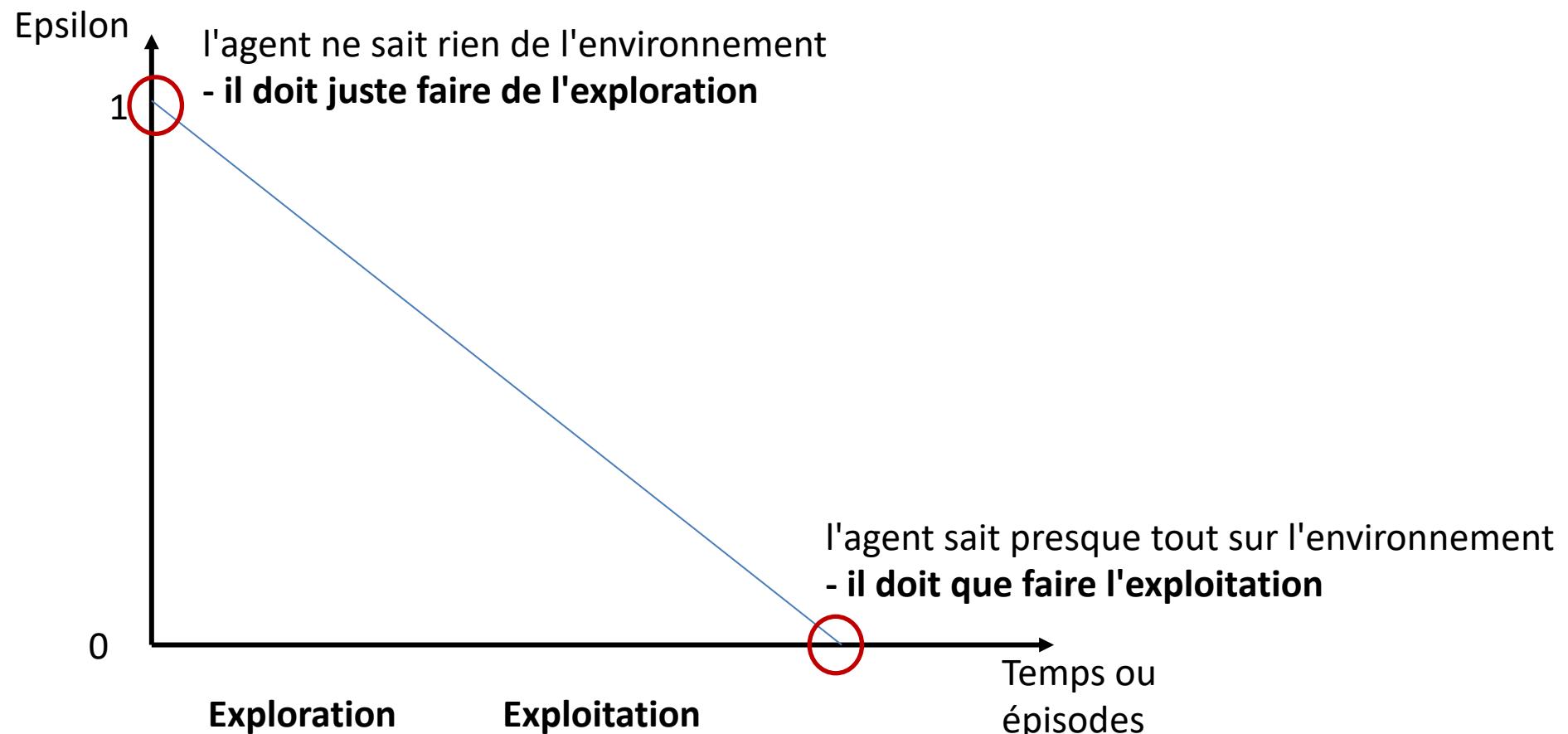
- Nous spécifions un taux d'exploration « epsilon », qui varie entre 0 et 1.
 - 1 => actions totalement aléatoires
 - 0 => actions totalement greedy (avare)
- Approche (exemple) :
 1. epsilon = 1 au démarrage
 - Au début, ce taux doit avoir sa valeur la plus élevée, car nous ne savons rien de l'environnement.
 - Cela signifie que un agent doit faire beaucoup d'exploration en choisissant des actions au hasard.
 2. Nous générerons un nombre aléatoire. Si ce nombre > epsilon, alors nous ferons «exploitation» (cela signifie que nous utilisons ce que nous savons déjà pour sélectionner la meilleure action à chaque étape). Sinon, nous allons faire de l'exploration (ex. action *random*).
 3. Après chaque étape epsilon est décrémenté (ex. epsilon = epsilon – 0.01)

Exploration/Exploitation trade-off



Exploration/Exploitation trade-off

- Epsilon-greedy algorithm



Trois approches pour le *reinforcement learning*

- Value-based
- Policy-based
- Model-based

Trois approches pour le *reinforcement learning*

- **Value-based**
- Dans RL basé sur la valeur, l'objectif est d'optimiser la fonction de valeur $V(s)$.
- La fonction de valeur est une fonction qui nous indique la récompense future maximale attendue par l'agent pour chaque état.
- La valeur de chaque état correspond au montant total de la récompense qu'un agent peut s'attendre à accumuler à l'avenir, à partir de cet état.

$$v_{\pi}(s) = \mathbb{E}_{\pi} [R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s]$$

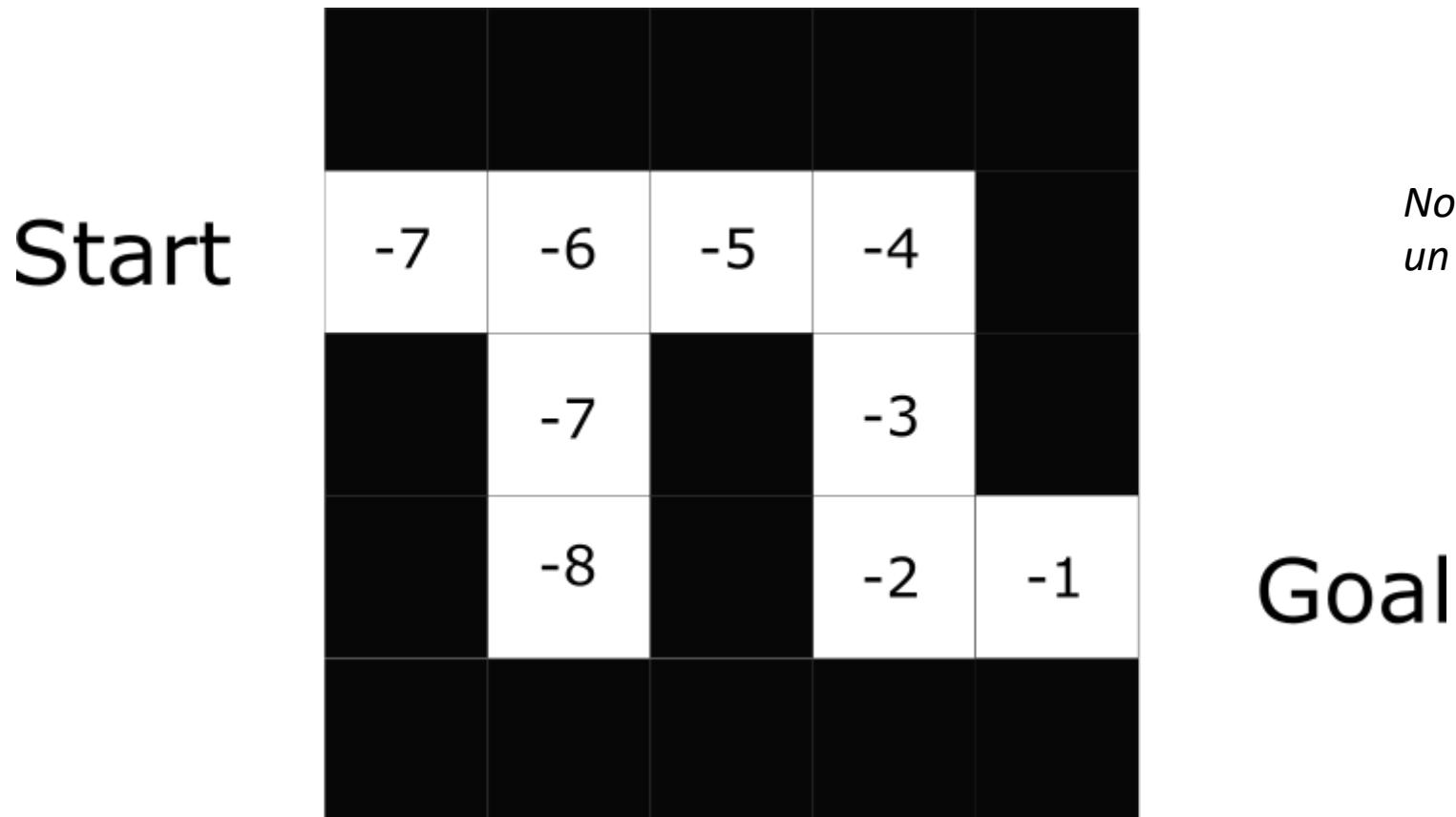
Expected

Reward
discounted

Given that state

Trois approches pour le *reinforcement learning*

- Value-based



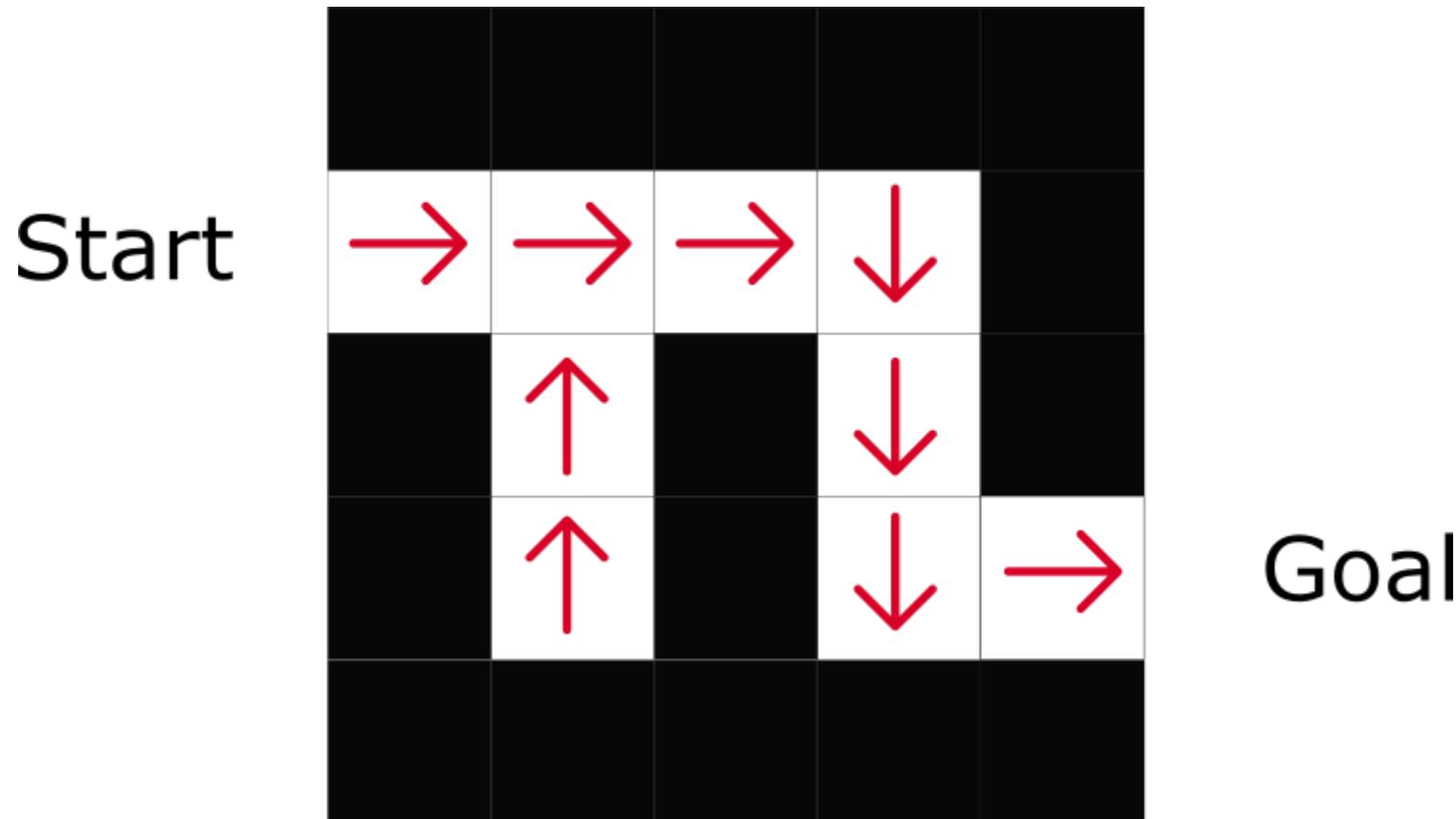
Trois approches pour le *reinforcement learning*

- **Policy-based**
- Dans la RL basée sur des stratégies/règles, nous souhaitons optimiser directement la fonction de règles $\pi(s)$ sans utiliser de fonction de valeur.
- La stratégie définit le comportement (action) de l'agent à un moment/état donné.

$$a = \pi(s)$$

Trois approches pour le *reinforcement learning*

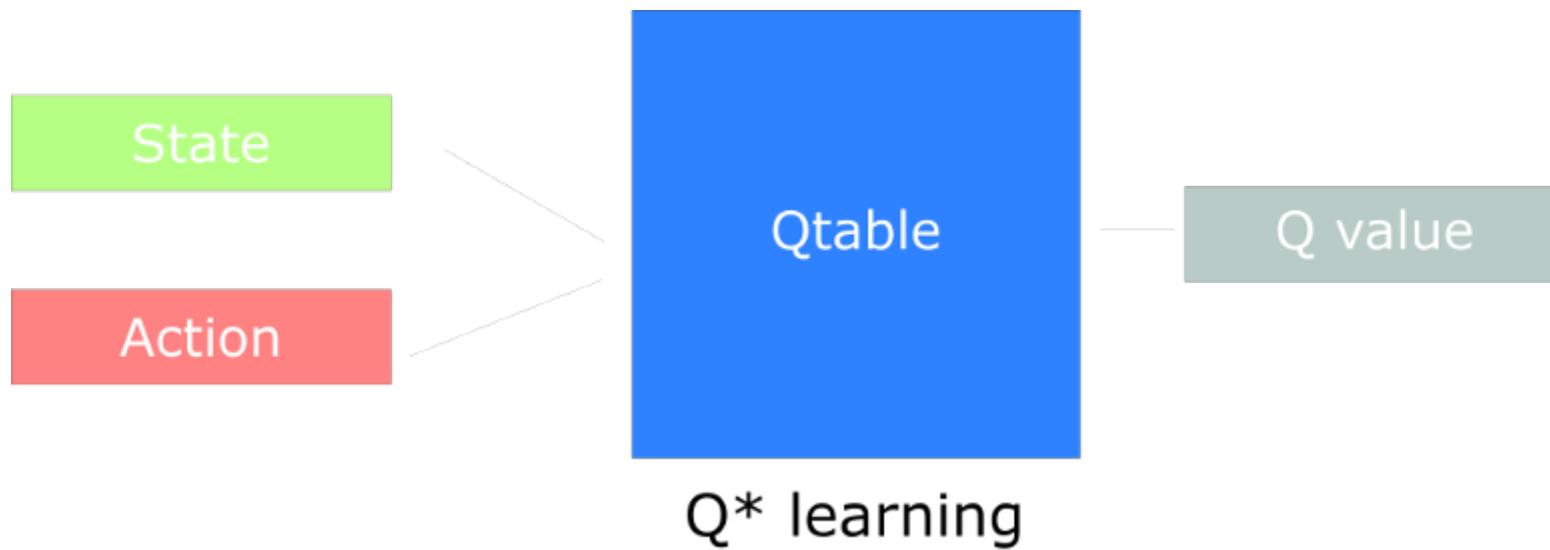
- Policy-based



Trois approches pour le *reinforcement learning*

- **Model-based**
- Dans RL basé sur un modèle, nous modélisons l'environnement. Cela signifie que nous créons un modèle du comportement de l'environnement.
- Travail spécifique pour chaque environnement
 - Besoin de parler de *Deep Neural Networks* => pas présenté ici

Un simple algorithme : Q-Learning



Q-Learning

- Pour parler de Q-Learning nous avons besoin de 3 concepts :
 - La fonction action-value (indiquée par la lettre 'Q')
 - La Q-table
 - L'équation de Bellman

Q-Learning - La fonction action-value

- La fonction de valeur d'action (ou « fonction Q ») prend deux entrées : «état» et «action». Elle renvoie la récompense future attendue de cette action à cet état.

$$Q^\pi(s_t, a_t) = \underline{E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | s_t, a_t]}$$

Q value for that state given that
action

Expected discounted cumulative reward ...

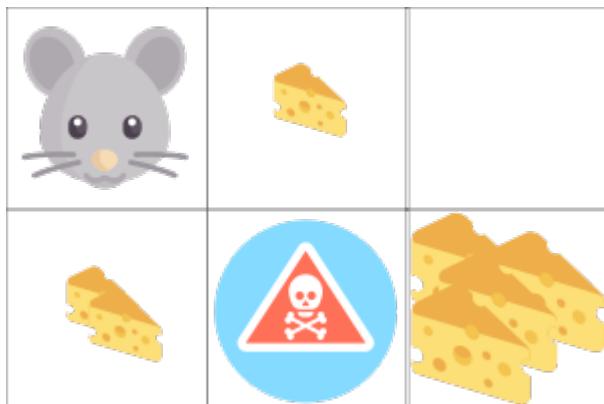
given that
state and
that action

Q-Learning – Le Q-table

- Un **tableau** dans lequel nous calculons la récompense future maximale attendue (*maximum expected future reward*), **pour chaque action dans chaque état**.

Q-Learning – Le Q-table

- Un **tableau** dans lequel nous calculons la récompense future maximale attendue (*maximum expected future reward*), **pour chaque action dans chaque état**.



Questions:

- Combiens d'états ?
- Combiens d'actions ?
- Nous construisons une table Q, avec m cols (m = nombre d'actions) et n lignes (n = nombre d'états).
- **Exercice**: construire un Q-table (vide) pour le *grid world* dans l'image à côté

Q-Learning – L'équation de Bellman

- Le(s) équation(s) de Bellman sont omniprésentes dans le RL et sont nécessaires pour comprendre le fonctionnement des algorithmes RL.
- Dans ce cours nous n'allons pas les démontrer et on présentera seulement l'équation de Bellman pour la fonction *action-value* (importante pour le Q-learning).

Q-Learning – L'équation de Bellman

$$NewQ(s, a) = Q(s, a) + \alpha[R(s, a) + \gamma \max Q'(s', a') - Q(s, a)]$$

New Q value for that state and that action

Current Q value

Reward for taking that action at that state

Learning Rate

Discount rate

Maximum expected future reward given the new s' and all possible actions at that new state

Q-Learning

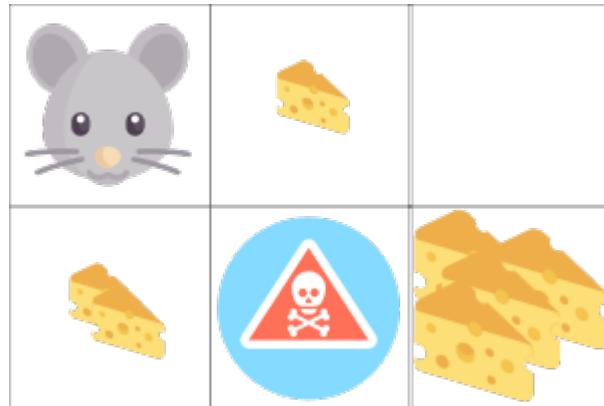
- Pseudo-code :

Par exemple zéro

1. Initialize Q-values ($Q(s, a)$) arbitrarily for all state-action pairs.
2. For life or until learning is stopped...
3. Choose an action (a) in the current world state (s) based on current Q-value estimates ($Q(s, \cdot)$).
4. Take the action (a) and observe the outcome state (s') and reward (r).
5. Update $Q(s, a) := Q(s, a) + \alpha [r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$

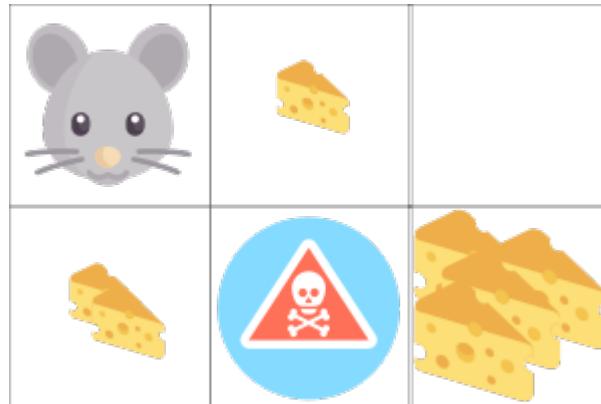
Comment choisir une action ?? Ex. Epsilon-greedy

Q-Learning : Exemple – un premier pas



- Rewards
 - Un fromage = +1
 - Deux fromages = +2
 - Gros tas de fromage = +10 (fin de l'épisode)
 - Manger du poison = -10 (fin de l'épisode)

Q-Learning : Exemple – un premier pas

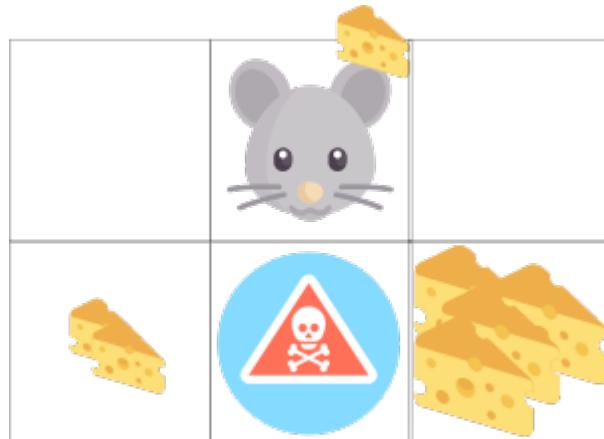


- Rewards
 - Un fromage = +1
 - Deux fromages = +2
 - Gros tas de fromage = +10 (fin de l'épisode)
 - Manger du poison = -10 (fin de l'épisode)

Q-table (initial)

	←	→	↑	↓
Start	0	0	0	0
Small cheese	0	0	0	0
Nothing	0	0	0	0
2 small cheese	0	0	0	0
Death	0	0	0	0
Big cheese	0	0	0	0

Q-Learning : Exemple – un premier pas



- Rewards
 - Un fromage = +1
 - Deux fromages = +2
 - Gros tas de fromage = +10 (fin de l'épisode)
 - Manger du poison = -10 (fin de l'épisode)

Q-table (step1)

	←	→	↑	↓
Start	0	0	0	0
Small cheese	0	0	0	0
Nothing	0	0	0	0
2 small cheese	0	0	0	0
Death	0	0	0	0
Big cheese	0	0	0	0

Q-Learning : Exemple – un premier pas

- Mise à jour de la valeur de la table, grâce à l'équation de Bellman

$$NewQ(s, a) = Q(s, a) + \alpha[R(s, a) + \gamma \max Q'(s', a') - Q(s, a)]$$

Dans l'exemple:

- alpha (learning rate) = 0.1
- lambda (discount factor) = 0.9

$$NewQ(start, right) = Q(start, right) + \alpha[\underline{\Delta Q(start, right)}]$$

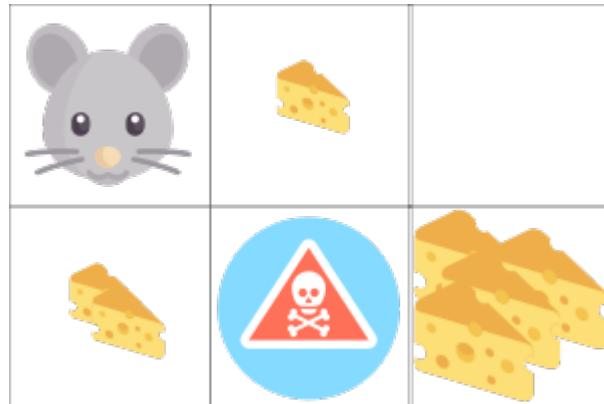
$$\underline{\Delta Q(start, right)} = R(start, right) + \gamma \max Q'(1cheese, a') - \underline{Q(start, right)}$$

$$\underline{\Delta Q(start, right)} = 1 + 0.9 * \underline{\max(Q'(1cheese, left), Q'(1cheese, right), Q'(1cheese, down))} - \underline{Q(start, right)}$$

$$\underline{\Delta Q(start, right)} = 1 + 0.9 * \underline{0} - \underline{0} = 1$$

$$NewQ(start, right) = 0 + 0.1 * 1 = 0.1$$

Q-Learning : Exemple – un premier pas



- Rewards
 - Un fromage = +1
 - Deux fromages = +2
 - Gros tas de fromage = +10 (fin de l'épisode)
 - Manger du poison = -10 (fin de l'épisode)

Q-table (initial)

	←	→	↑	↓
Start	0	0.1	0	0
Small cheese	0	0	0	0
Nothing	0	0	0	0
2 small cheese	0	0	0	0
Death	0	0	0	0
Big cheese	0	0	0	0

Iterate...

Q-Learning : Conclusion



How big is your Q-Table?

- Taille = Etats * Actions

Q-Learning: Limitations



How big is your Q-Table?

- N° états?
 - Il y a environ $2 * 10^{170}$ configurations légales possibles
- Actions
 - $19 \times 19 = 361$
- Note
 - La taille approximative de l'arbre des possibilités du jeu est de 10^{600}
 - $361!/100!$ des différentes parties de moins de 260 coups

Q-Learning: Limitations

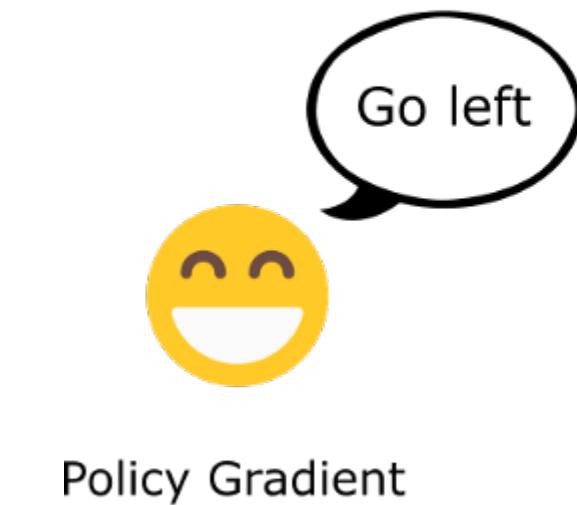
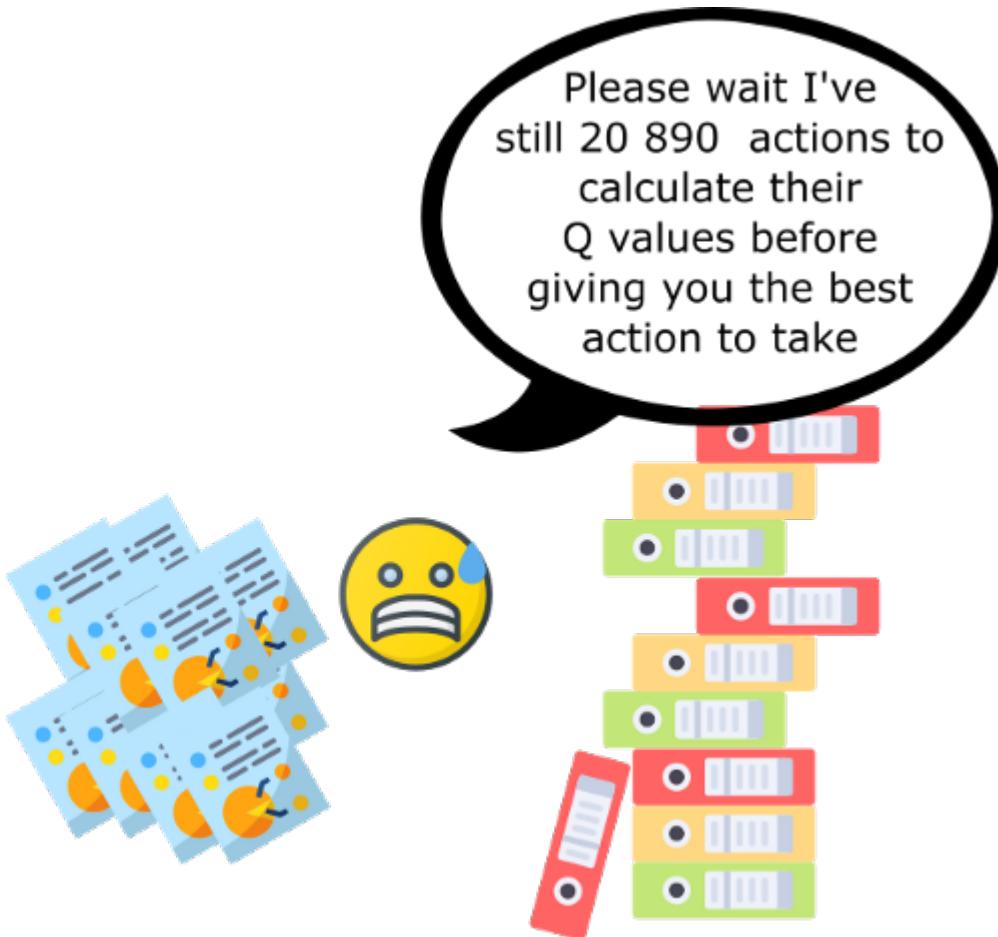
- How big is your Q-Table?
 - Ce problème peut être résolu en approximant le Q-Table
 - Avec des Neural Networks => **Deep Q-Learning**
- **Mais si le nombre d'actions est très très grand**
 - Ex. Robot dans un espace continu

$$NewQ(s, a) = Q(s, a) + \alpha[R(s, a) + \gamma \max Q'(s', a') - Q(s, a)]$$



Nécessité de calculer la valeur de Q de
Toutes les actions possibles !!

Q-Learning: Limitations



Conclusion

- Reinforcement Learning
 - Domaine de recherche très (très) actif
- Q-learning
 - Première approche de RL
 - Déjà de très bon résultats... mais limitée (espace en mémoire, nombre d'actions, ...)
- Aller plus loin
 - DQN (deep Q-learning networks)
 - DDQN (double DQN)
 - PPO (proximal policy approximation)
 - ...

A retenir

- Reinforcement Learning : buts, approche, terminologie
 - Etat, action, policy, reward, MDP,...
 - Value-based Vs Policy-based
- Exploration/Exploitation : algorithme epsilon-greedy
- Discounted expected cumulative reward : définition
- Q-learning :
 - Algorithme
 - Q-table
 - Equation de Bellman