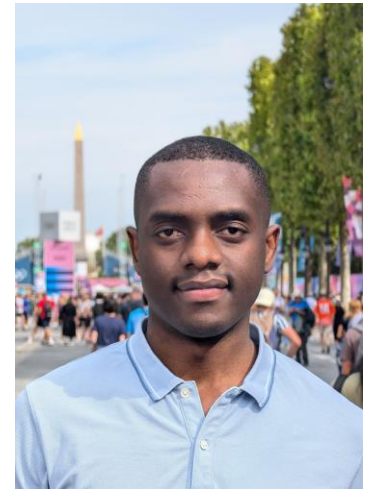


# *Introduction à l'apprentissage par renforcement*

Par Sonny Mupfuni

# *Meetup : Python, ML and DL*

*Membre de Python Software  
Foundation Network*



Sonny Mupfunu

Msc en IA, Université Paris Cité

# Plan



L'apprentissage par renforcement et ses applications



Les fondamentaux de l'apprentissage par renforcement : agents, environnements, états, actions et récompenses



Le processus de décision de Markov (MDP) et son rôle central



Les algorithmes classiques comme Q-Learning et SARSA



Une démonstration pratique avec la bibliothèque Python PEARL (Production-Ready Reinforcement Learning) créée par Meta AI

A network diagram consisting of orange lines connecting silver pins on a white background. The lines form a complex web of interconnected nodes, with some lines crossing each other. The pins are small, cylindrical, and metallic, and they are distributed across the white surface. The orange lines are thin and flexible, creating a mesh-like structure. The background is a plain, light-colored surface.

# *1. L'apprentissage par renforcement et ses applications*

---

# *Intuition*



Sonny Mupfun

# Reinforcement Learning (RL)



Une branche de l'intelligence artificielle qui permet à un agent d'apprendre à partir de ses interactions avec un environnement



L'agent apprend par essais et erreurs en recevant des récompenses ou des pénalités



Inspiré du comportement animal et de la psychologie comportementale



Un apprentissage plus naturel que l'apprentissage supervisé



**SUPERVISED  
LEARNING**  
(LABELED DATA)



**UNSUPERVISED  
LEARNING**  
(UNLABELED DATA)



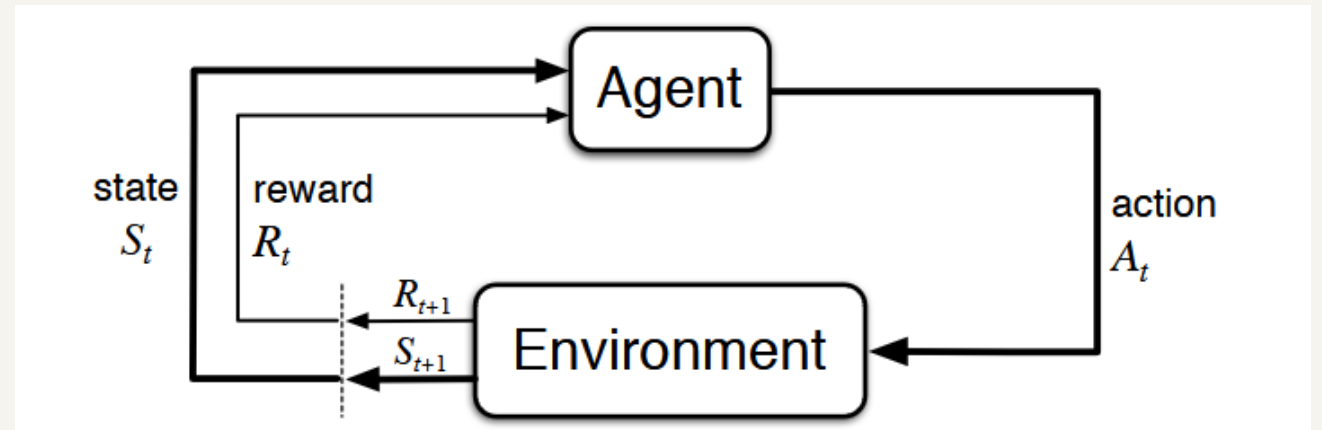
**SELF-SUPERVISED  
LEARNING**  
(LEARNS FROM DATA ITSELF)



**REINFORCEMENT  
LEARNING**  
(LEARN FROM EXPERIENCE)

# Principe fondamental

1. L'agent observe l'état de l'environnement
2. Il choisit et exécute une action
3. Il reçoit une récompense et observe le nouvel état
4. Il apprend de cette expérience pour optimiser ses actions futures

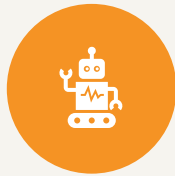




# *Applications de l'apprentissage par renforcement*



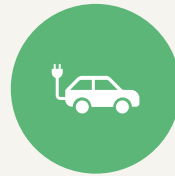
JEUX



ROBOTIQUE



TRANSPORT



ÉNERGIE



FINANCE



SANTÉ



LLM



# *Jeux*

---

- AlphaGo (Go)
- DQN (Atari 2600)
- AlphaStar (StarCraft II)
- Cicero (Diplomacy)
- Pluribus (Poker)

Sonny Mupfuni



# *Robotique*

---

- Apprentissage de la marche
- Manipulation d'objets
- Robots industriels



# *Transport*

---



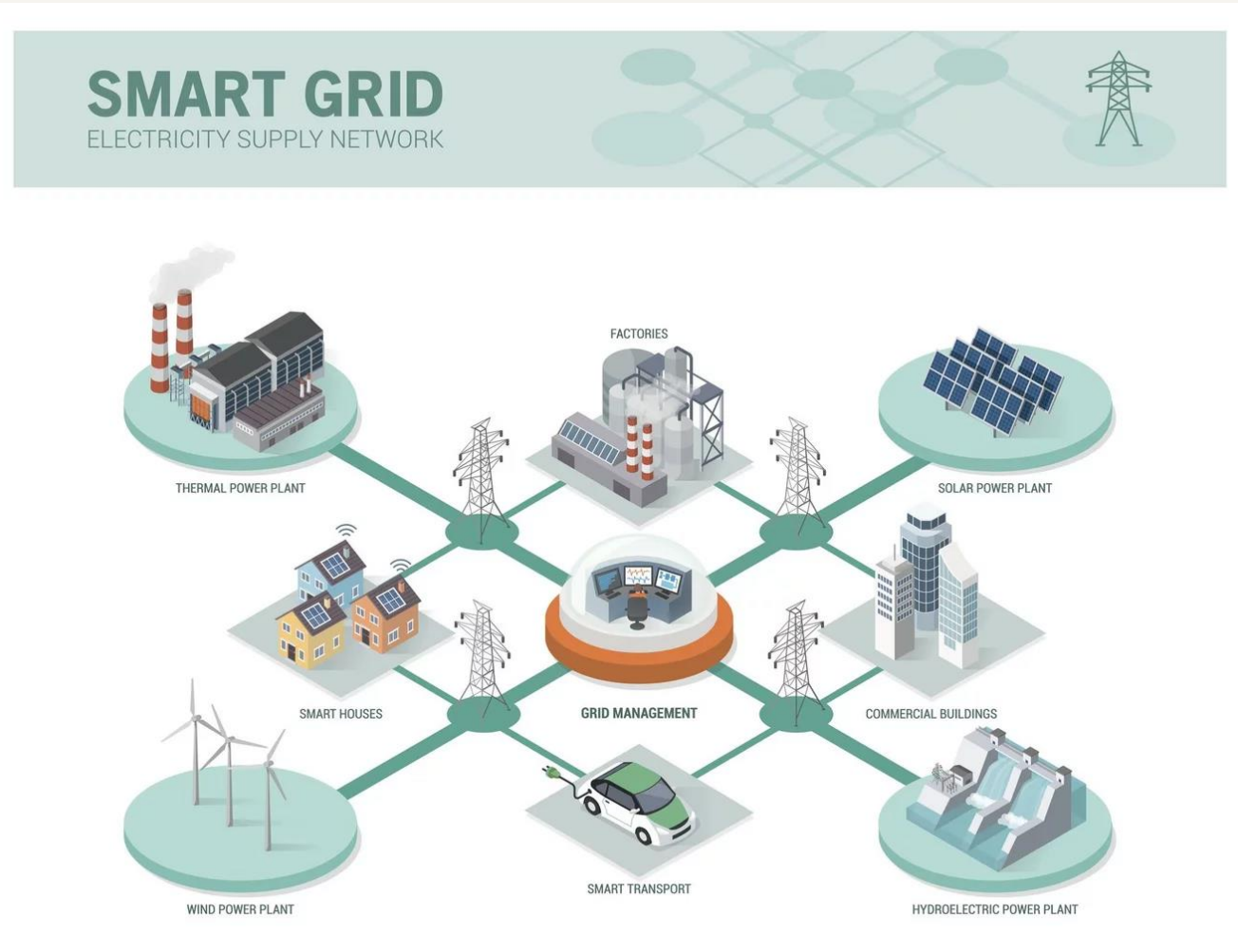
- Véhicules autonomes
- Optimisation du trafic
- Systèmes de navigation

Sonny Mapian



# *Energie*

- Gestion intelligente des réseaux électriques
- Optimisation de la consommation
- Maintenance prédictive



# *Finance*

---

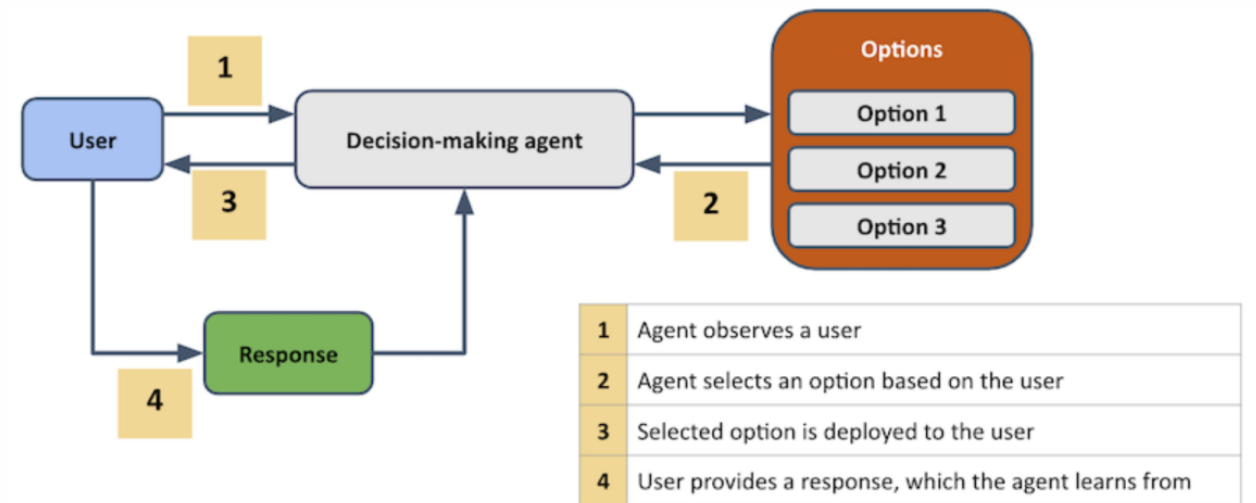
- Trading algorithmique
- Gestion de portefeuille
- Détection de fraudes



# *Systemes de recommandation*

- Recommandations de produits (e-commerce)
- Suggestions de contenu (streaming)
- Optimisation publicitaire en temps réel
- Personnalisation des fils d'actualité

## What is Contextual Bandits



# *Santé*

---

- Personnalisation des traitements
- Diagnostic assisté
- Planification des interventions

Sonny Mupfun

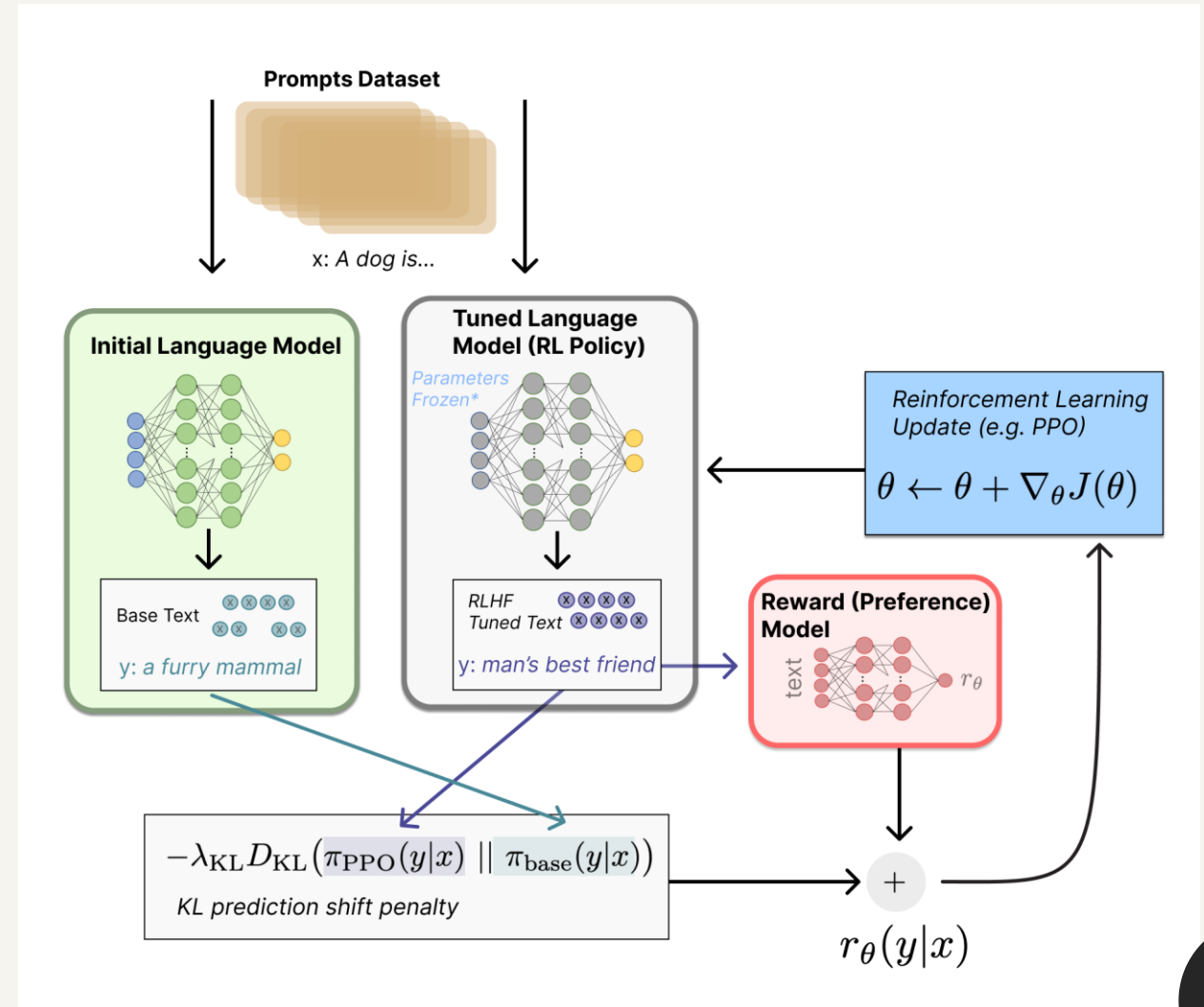





# Alignement des LLMs et *raisonnement*

- Apprentissage des préférences humaines
- Amélioration de la sécurité et de l'éthique
- Réduction des réponses nuisibles
- Résolution de problèmes complexes
- Chaînage logique (Chain of Thought)
- Vérification des réponses

Sonny Mupfuni





## *2. Les fondamentaux de l'apprentissage par renforcement*

---

# *Les concepts fondamentaux du RL*



AGENTS



ENVIRONNEMENT



ETATS



ACTIONS



RECOMPENSE

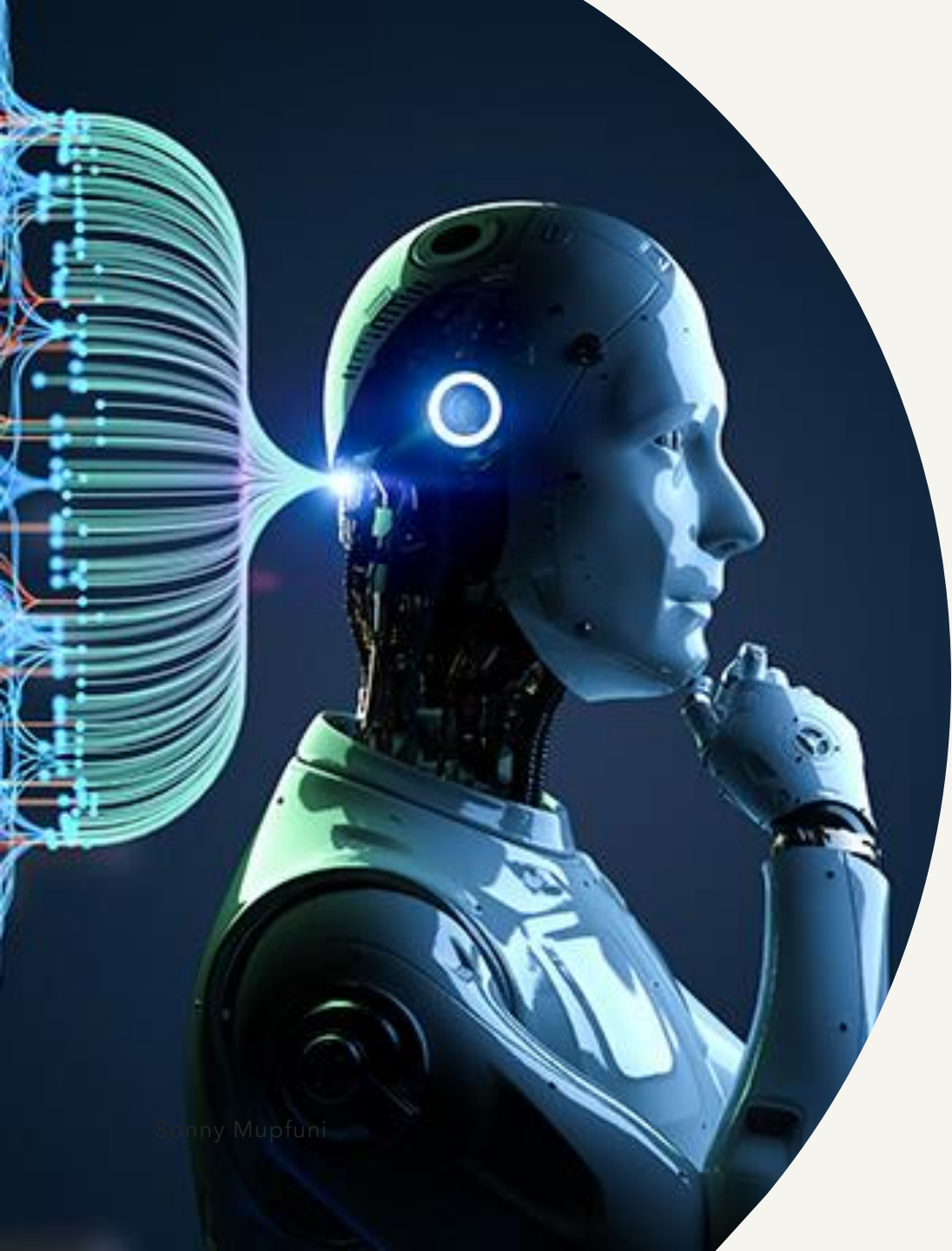


POLITIQUE

# *Agent en RL*

---

- Entité autonome (comme un logiciel) qui prend des décisions et agit
- Apprend de ses interactions
- Cherche à maximiser une récompense cumulée



# *Caractéristiques d'un agent*

---

- Capacité d'observation (capteurs)
- Prise de décision autonome et capacité d'agir (effecteurs)
- Mémoire des expériences passées
- Adaptation du comportement

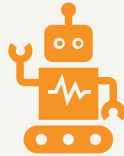
# Exemples d'agents



## Transport

Agent de contrôle des  
feux de circulation  
(Siemens)

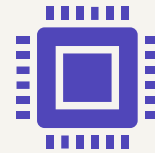
Système de pilotage  
automatique Tesla



## Industrie

Robot de tri des  
déchets (AMP Robotics)

Bras robotique de  
picking Amazon



## Systèmes de contrôle

Gestionnaire IA des  
centres de données

Google

Logiciel de trading  
automatique

MetaTrader



## Autres types d'agents

Agents conversationnel  
et assistants virtuels  
(ChatGPT, Claude,  
Llama-Instruct, etc)

Agents LLM :  
Planification et  
exécution de tâches (  
ReAct, AutoGPT,...)

# *Environnement en RL*

---

- Système avec lequel l'agent interagit
- Peut être réel ou simulé
- Répond aux actions de l'agent



## *Types d'environnements*

---

Déterministes vs  
Stochastiques

---

Entièrement vs  
Partiellement observables

---

Épisodiques vs Continus

---

Statiques vs Dynamiques

# *Env. Déterministe vs Stochastique*

**Déterministe** : Jeu d'échecs

- Chaque action mène à un état unique et prévisible
- Le résultat d'un mouvement est toujours le même

**Stochastique** : Poker

- Les cartes sont distribuées aléatoirement
- Les résultats des actions sont probabilistes



# *Env. Entièrement vs Partiellement observable*

## **Entièrement observable** : Jeu d'échecs

- Position de toutes les pièces visibles
- État complet du jeu connu à tout moment
- Pas d'information cachée pour aucun joueur

## **Partiellement observable** : Super Mario

- Vue limitée au cadre de l'écran
- Ennemis et obstacles non visibles hors écran
- Nécessité d'avancer pour découvrir l'environnement

### Observation Space

*State: complete description of the state of the world (no hidden information).*



*Observation: partial description of the state of the world.*



# *Env. Episodique vs Continu*

- **Épisodique** : Jeu de Tennis, Jeu d'échecs
  - *Chaque point est un épisode distinct*
  - *Le résultat d'un point n'affecte pas directement le suivant*
  - *Reset après chaque épisode*
- **Continu** : Robot autonome, Thermostat intelligent, Trading automatisé
  - *Interaction continue sans fin claire*
  - *Les actions passées influencent le futur lointain (les récompenses sont continues)*
  - *Pas de reset après chaque épisode*

# *Env. Statique vs Dynamique*

**Statique** : Jeu de Go, Jeu d'échecs

- L'environnement attend l'action de l'agent
- Pas de changement pendant la réflexion

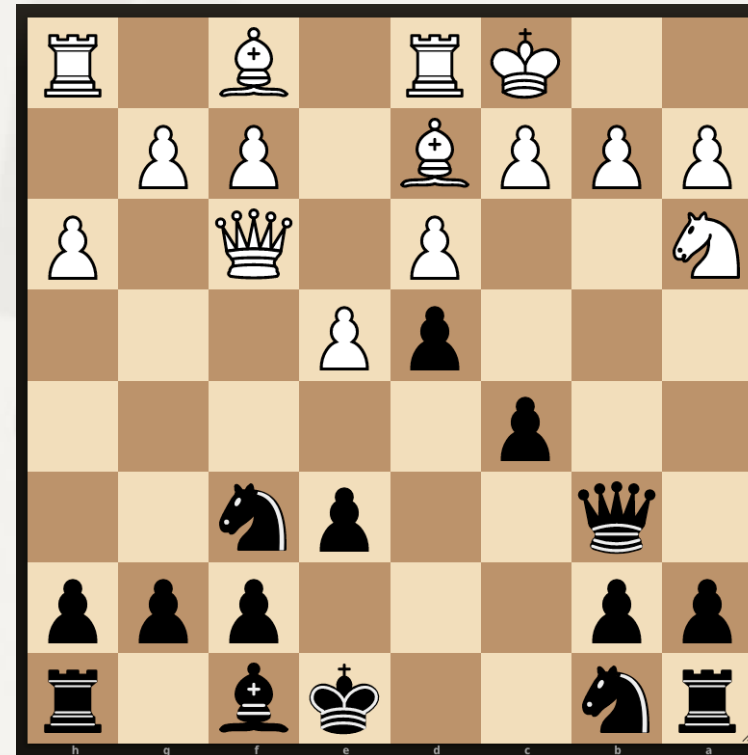
**Dynamique** : Voiture autonome

- L'environnement change constamment
- Les autres véhicules bougent pendant la prise de décision



# *Etats (States)*

- Représentation de la situation
- Information disponible pour la prise de décision
- Base pour choisir les actions



# *Exemples d'Etats*



Position et vitesse  
dans un jeu



Configuration d'un  
robot



Situation de marché  
en trading



# *Les actions*

## **Définition**

- Choix possibles de l'agent
- Modifications de l'environnement
- Transitions entre états

## **Types d'actions**

- Discrètes
  - *Nombre fini de choix*
  - *Ex: Gauche, droite, haut, bas*
- Continues
  - *Valeurs dans un intervalle*
  - *Ex: Angle de rotation, vitesse*



# *Récompenses*

## **Définition**

- Signal de feedback
- Mesure de la qualité des actions
- Guide pour l'apprentissage

## **Caractéristiques**

- Immédiates vs différées
- Positives vs négatives
- Sparse vs denses

## **Conception des récompenses**

- Alignement avec l'objectif
- Éviter les récompenses trompeuses

# *Politique* *(Policy $\pi$ )*

- **Définition**

Stratégie de l'agent

Mapping état  $\rightarrow$  action

Peut être déterministe ou stochastique

- **Objectifs**

Maximiser la récompense cumulée (Politique optimale notée  $\pi^*$ )

Trouver l'équilibre exploration/exploitation

S'adapter aux changements

# *Exemples des politiques*

- **Politique uniforme aléatoire**

Selectionne toujours une action au hasard

Toutes les actions ont la même probabilité d'être choisie

- **Politique gloutonne (greedy)**

Selectionne toujours l'action ayant la plus grande probabilité (exploite toujours)

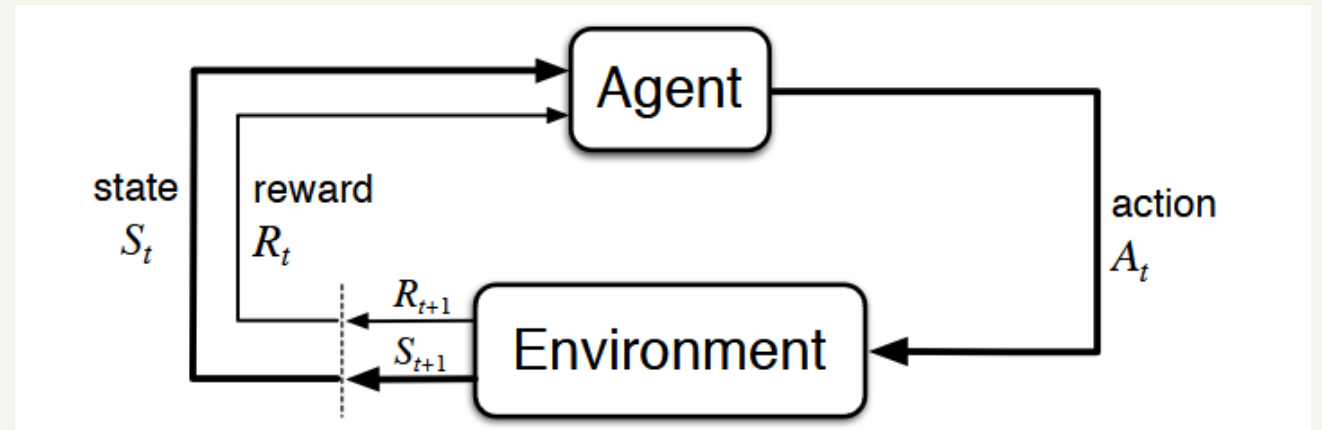
- **Politique epsilon-gloutonne (epsilon greedy)**

Explore avec probabilité epsilon

Exploite avec probabilité 1-epsilon

# *Rappel du Principe fondamental*

1. L'agent observe l'état de l'environnement
2. Il choisit et exécute une action
3. Il reçoit une récompense et observe le nouvel état
4. Il apprend de cette expérience pour optimiser ses actions futures





### *3. Les processus décisionnels de Markov (MDP)*

---

# *Un MDP*



Un cadre pour modéliser la prise de décision séquentielle



Représente des situations où les décisions actuelles impactent le futur



Permet de planifier dans un environnement incertain



# *Un MDP*

- Un processus décisionnel de Markov est formellement défini par un tuple : **(S, A, P, R, γ)**, où :

## **1. États (S) :**

*Un ensemble fini d'états possibles dans lesquels l'agent peut se trouver.*

**Formellement** :  $S = \{s_1, s_2, \dots, s_n\}$  où  $n$  est le nombre d'états.

**Exemple** : dans un jeu de labyrinthe, chaque case pourrait représenter un état distinct.

## **2. Actions (A) :**

*Un ensemble fini d'actions possibles que l'agent peut prendre.*

**Formellement** :  $A = \{a_1, a_2, \dots, a_m\}$  où  $m$  est le nombre d'actions.

**Exemple** : dans un labyrinthe, les actions peuvent être se déplacer vers le haut, le bas, la gauche ou la droite.

# *Un MDP*

Un processus décisionnel de Markov est formellement défini par un tuple : **(S, A, P, R, γ)**, où :

## **3. Fonction de transition (P) :**

Définition : Elle décrit la probabilité de passer d'un état à un autre après avoir effectué une action particulière.

Formellement :  $P(s' | s, a)$  = Probabilité de passer à l'état  $s'$  après avoir effectué l'action  $a$  dans l'état  $s$ .

Exemple :  $P(\text{case\_2} | \text{case\_1}, \text{aller\_droite}) = 0,8$  signifie qu'il y a 80 % de chances de passer de la case 1 à la case 2 en allant vers la droite.

# *Un MDP*

Un processus décisionnel de Markov est formellement défini par un tuple : **(S, A, P, R, γ)**, où :

## **3. Fonction de récompense (R) :**

Définition : Elle décrit la récompense que l'agent reçoit après une transition vers un nouvel état ou lors d'une action.

Formellement :  $R(s, a, s')$  = Récompense obtenue après avoir effectué l'action  $a$  dans l'état  $s$ , en passant à l'état  $s'$ .

Exemple :  $R(\text{case\_1}, \text{aller\_droite}, \text{case\_2}) = -1$  signifie qu'il y a une petite pénalité pour le déplacement.  $R(\text{case\_objectif}, \text{aller\_droite}, \text{case\_objectif}) = +10$  signifie qu'une récompense importante est donnée pour avoir atteint l'objectif.

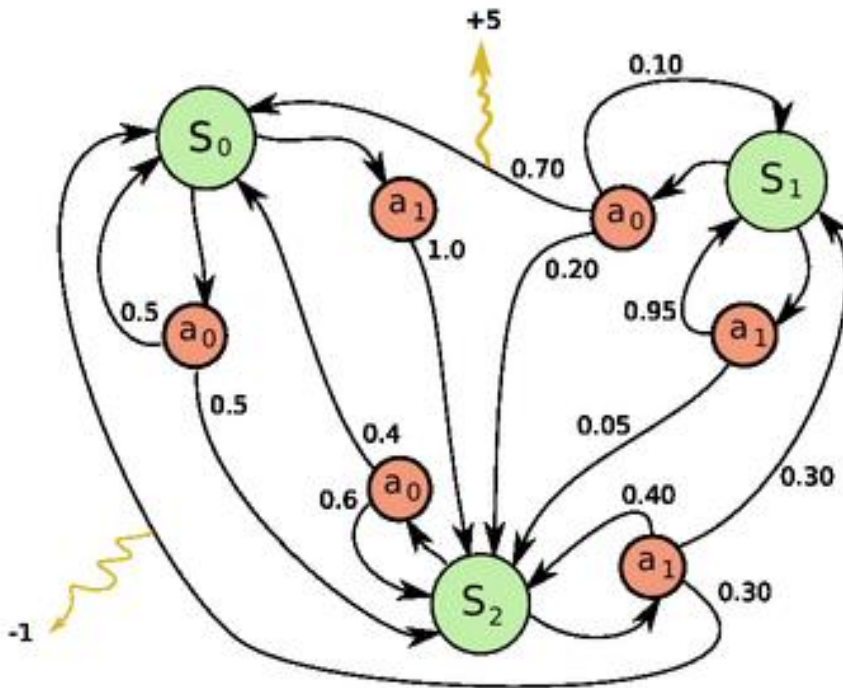
# *Un MDP*

Un processus décisionnel de Markov est formellement défini par un tuple : **(S, A, P, R,  $\gamma$ )**, où :

## **5. Facteur d'escompte ( $\gamma$ ) :**

- **Définition** : Un facteur qui détermine l'importance des récompenses futures par rapport aux récompenses immédiates. C'est un nombre entre 0 et 1 ( $0 \leq \gamma \leq 1$ ).
- **Formellement** :  $\gamma \in [0, 1]$ 
  - *Si  $\gamma$  est proche de 0, l'agent valorise les récompenses immédiates.*
  - *Si  $\gamma$  est proche de 1, l'agent valorise davantage les récompenses futures.*

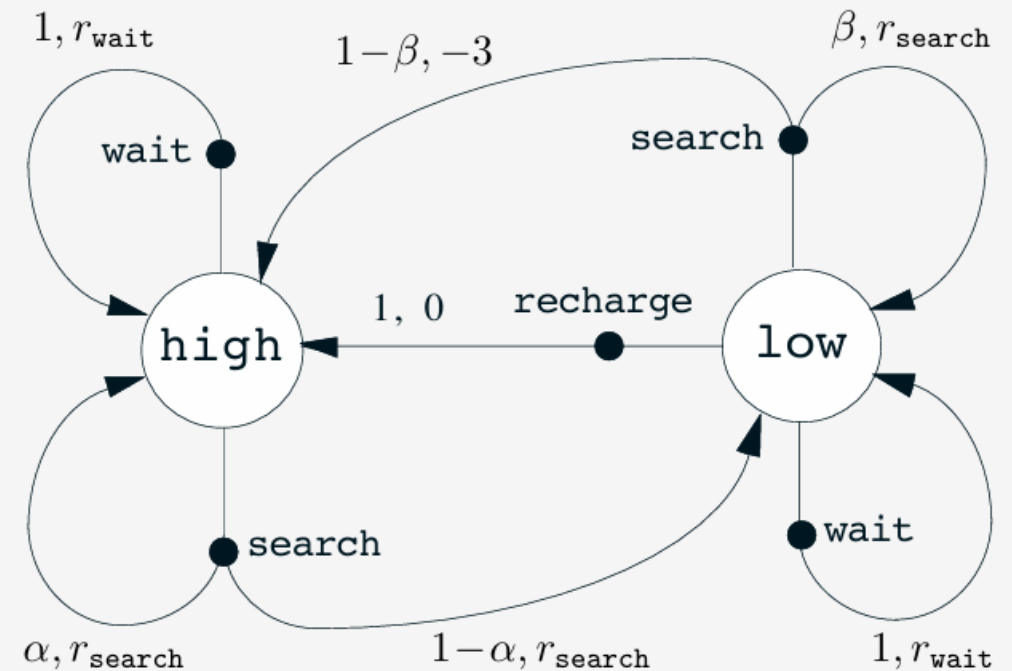
# Exemple d'un MDP



État actuel	Action	État suivant	Probabilité	Récompense
$S_0$	$a_0$	$S_0$	0.5	0
$S_0$	$a_0$	$S_2$	0.5	0
$S_0$	$a_1$	$S_2$	1.0	+5
$S_1$	$a_0$	$S_0$	0.7	0
$S_1$	$a_0$	$S_1$	0.1	0
$S_1$	$a_0$	$S_2$	0.2	0
$S_1$	$a_1$	$S_1$	0.95	0
$S_1$	$a_1$	$S_2$	0.05	0
$S_2$	$a_0$	$S_2$	0.6	0
$S_2$	$a_0$	$S_0$	0.4	-1
$S_2$	$a_1$	$S_1$	0.3	0
$S_2$	$a_1$	$S_2$	0.7	0

# Exemple d'un MDP : Robot Recyclant

$s$	$a$	$s'$	$p(s'   s, a)$	$r(s, a, s')$
high	search	high	$\alpha$	$r_{\text{search}}$
high	search	low	$1 - \alpha$	$r_{\text{search}}$
low	search	high	$1 - \beta$	$-3$
low	search	low	$\beta$	$r_{\text{search}}$
high	wait	high	1	$r_{\text{wait}}$
high	wait	low	0	-
low	wait	high	0	-
low	wait	low	1	$r_{\text{wait}}$
low	recharge	high	1	0
low	recharge	low	0	-





# *Les valeurs dans un MDP : $V(s)$ et $Q(s,a)$*

**L'agent utilise une politique  $\pi$  fixée et construit deux fonctions :**

## **1. Valeur ou utilité d'état $V(s)$**

- "Combien vaut cet état sur le long terme ?"
- Pour chaque état  $s$ , on associe une valeur réelle correspondant à la somme des récompenses espérées qu'on peut obtenir à partir de cet état.

## **2. Valeur ou qualité d'état-action $Q(s,a)$**

- "Que vaut cette action dans cet état ?"
- Pour chaque couple état-action, on associe une valeur réelle correspondant à la qualité de l'action dans cet état.

# *Les valeurs dans un MDP : $V(s)$ et $Q(s,a)$*

## L'équation de Bellman pour $V(s)$

### Forme générale

Pour une politique  $\pi$  donnée :

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

### Décomposition

1.  $\sum_a \pi(a|s)$  : Probabilité de chaque action selon la politique
2.  $\sum_{s'} P(s'|s, a)$  : Probabilité des états suivants
3.  $R(s, a, s')$  : Récompense immédiate
4.  $\gamma V^\pi(s')$  : Valeur future actualisée

# *Les valeurs dans un MDP : $V(s)$ et $Q(s,a)$*

## L'équation de Bellman pour $Q(s,a)$

### Forme générale

Pour une politique  $\pi$  donnée :

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

### Décomposition

1.  $\sum_{s'} P(s'|s, a)$  : Probabilité des transitions
2.  $R(s, a, s')$  : Récompense immédiate
3.  $\sum_{a'} \pi(a'|s')$  : Actions futures selon la politique
4.  $\gamma Q^\pi(s', a')$  : Valeur future actualisée

# *Les valeurs dans un MDP : $V(s)$ et $Q(s,a)$*

## Équations de Bellman optimales

Pour  $V^*$  (valeur optimale)

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

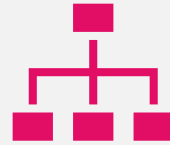
Pour  $Q^*$  (valeur état-action optimale)

$$Q^*(s, a) = \sum_{s'} P(s'|s, a) [R(s, a, s') + \gamma \max_{a'} Q^*(s', a')]$$

Relation entre  $V^*$  et  $Q^*$

$$V^*(s) = \max_a Q^*(s, a)$$

# Résolution d'un MDP



Résoudre un MDP consiste à trouver la **politique optimale**  $\pi^*$  : une politique supérieure ou égale à toutes les autres politiques (celle qui maximise la récompense cumulée attendue).



Il existe toujours au moins une politique optimale dans un MDP

# *Résolution d'un MDP*

**Programmation dynamique** : décomposer le problème en sous-problèmes plus simples (calcul de la valeur d'un état) et les résoudre de façon itérative.

- **Itération de la valeur** : résout les équations de Bellman par mise à jour successive des valeurs d'états jusqu'à convergence.
- **Itération de la politique** : utilise la programmation dynamique pour évaluer la politique (en résolvant le système d'équations de Bellman) et améliorer la politique itérativement.



A network diagram with orange lines and silver nodes on a white background. The lines connect various nodes, forming a complex web. The nodes are small, circular, and metallic in appearance. The lines are thin and orange. The background is a light gray with a subtle texture.

# *4. Méthodes et algorithmes d'apprentissage par renforcement*

---

# *Méthodes et algorithmes*

## **Apprentissage par renforcement**

- Q-Learning
- SARSA
- Méthodes de Monte-Carlo

## **Deep Reinforcement Learning**

- Value-Based Methods
- Policy Gradients Methods
- Actor-Critic Methods

# *Q-Learning*

Q-Learning est un algorithme qui apprend la fonction  $Q(s,a)$  sans nécessiter le modèle de l'environnement pour estimer la meilleure politique.

## 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.,  $\varepsilon$ -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

# Sarsa

Sarsa est similaire à Q-Learning. La différence réside dans la manière de mettre à jour  $Q(S,A)$  : SARSA utilise la même politique pour sélectionner les actions et pour apprendre (**on-policy**).

## 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.,  $\varepsilon$ -greedy)

    Loop for each step of episode:

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

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

# *Deep Reinforcement Learning*

## **Méthodes basées sur la valeur**

### **Deep Q-Network (DQN) et ses variantes**

- DQN vanilla : réseau de neurones pour approximer  $Q(s,a)$
- Double DQN : réduit la surestimation des valeurs

### **Caractéristiques**

- Adapté aux actions discrètes
- Utilise le replay buffer
- Network cible pour la stabilité
- Explore avec  $\epsilon$ -greedy

# Deep Reinforcement Learning

## Méthodes basées sur la politique

### Algorithmes clés

- REINFORCE : gradient de politique le plus simple
- PPO (Proximal Policy Optimization)
  - *Plus stable*
  - *Limite les changements de politique*
- TRPO (Trust Region Policy Optimization)
  - *Garanties théoriques*
  - *Plus complexe à implémenter*

### Caractéristiques

- Fonctionne en actions continues
- Apprentissage direct de la politique
- Peut être plus stable
- Souvent haute variance



# *Deep Reinforcement Learning*

## **Méthodes basées sur la politique**

### **Algorithmes clés**

- REINFORCE : gradient de politique le plus simple
- PPO (Proximal Policy Optimization)
  - *Plus stable*
  - *Limite les changements de politique*
- TRPO (Trust Region Policy Optimization)
  - *Garanties théoriques*
  - *Plus complexe à implémenter*

### **Caractéristiques**

- Fonctionne en actions continues
- Apprentissage direct de la politique
- Peut être plus stable
- Souvent haute variance

# *Autres Méthodes*

## **Model-Based RL**

- Apprend un modèle de l'environnement (world model)
- MuZero : champion aux jeux Atari
- Dreamer : planification dans l'espace latent

## **Apprentissage par imitation**

- Behavioral Cloning
- GAIL (Generative Adversarial Imitation Learning)
- IRL (Inverse Reinforcement Learning)

## **Multi-Agent RL**

- MADDPG pour environnements compétitifs
- QMIX pour environnements coopératifs

A network diagram consisting of numerous orange lines (edges) connecting small silver circular nodes (vertices) on a white background. The nodes are arranged in a non-uniform, interconnected pattern, resembling a mesh or a complex graph. The lines are thin and the nodes are small, creating a delicate, web-like structure.

## *5. Démonstration pratique*

---

# *Bibliothèques Python pour le RL*

Gymnasium (OpenAI Gym)

PEARL

Stable Baseline3

Dopamine

Rllib

Torch-RL

# *Pearl*

Une bibliothèque open-source d'agents d'Apprentissage par Renforcement (RL) créée par Meta AI

## **Caractéristiques**

- S'adapte aux environnements complexes
- Prêt pour la production
- Observabilité limitée
- Feedback Sparse

### Coming Back to Our Real-life Examples

Challenges	RL Features	Recommender Systems	Auction bidding	Creative Selection	Robotics	Supply Chain
Sparse Reward	Online Exploration	✓	✓	✓	✓	
Dangerous and Risky State and Actions	Safety		✓		✓	✓
Partial Observability	History Summarization	✓	✓		✓	✓
Changing Action Space	Dynamic Action Space Support	✓		✓	✓	✓
Offline Learning	Offline RL	✓	✓	✓	✓	✓