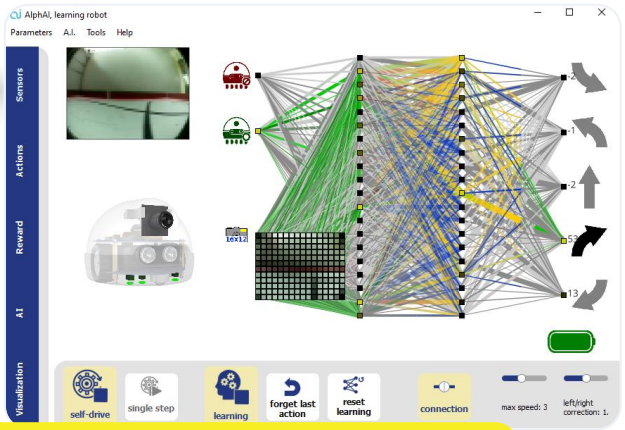
Learning Robots

# Startup de l’Université Paris-Saclay – Vulgarisation de l’IA

Entraîner une IA robot




Visualiser « Comment ça marche »

Activités ludiques & ambitieuses

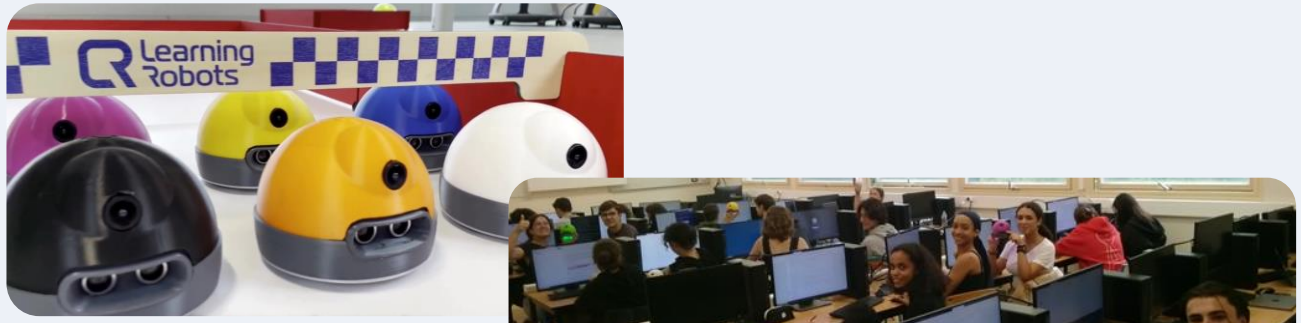
De l’Élémentaire …

… à l’Université


À l’international

En Entreprise

# Cours L1: “Découverte de l’Intelligence Artificielle”




Cours d’option ouvert à tous les étudiants de L1

12 * 2h

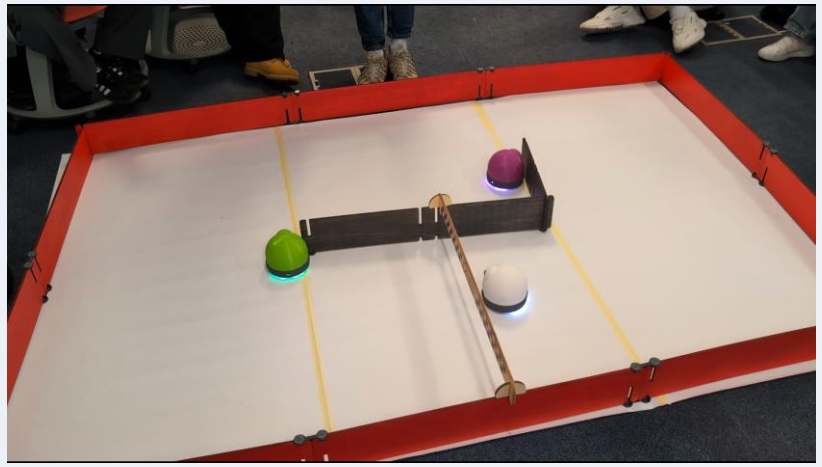
- **Concepts fondamentaux** du Machine Learning
- **Découvrir puis coder** 3 algorithmes (KNN, Réseaux de neurones, Q-learning)

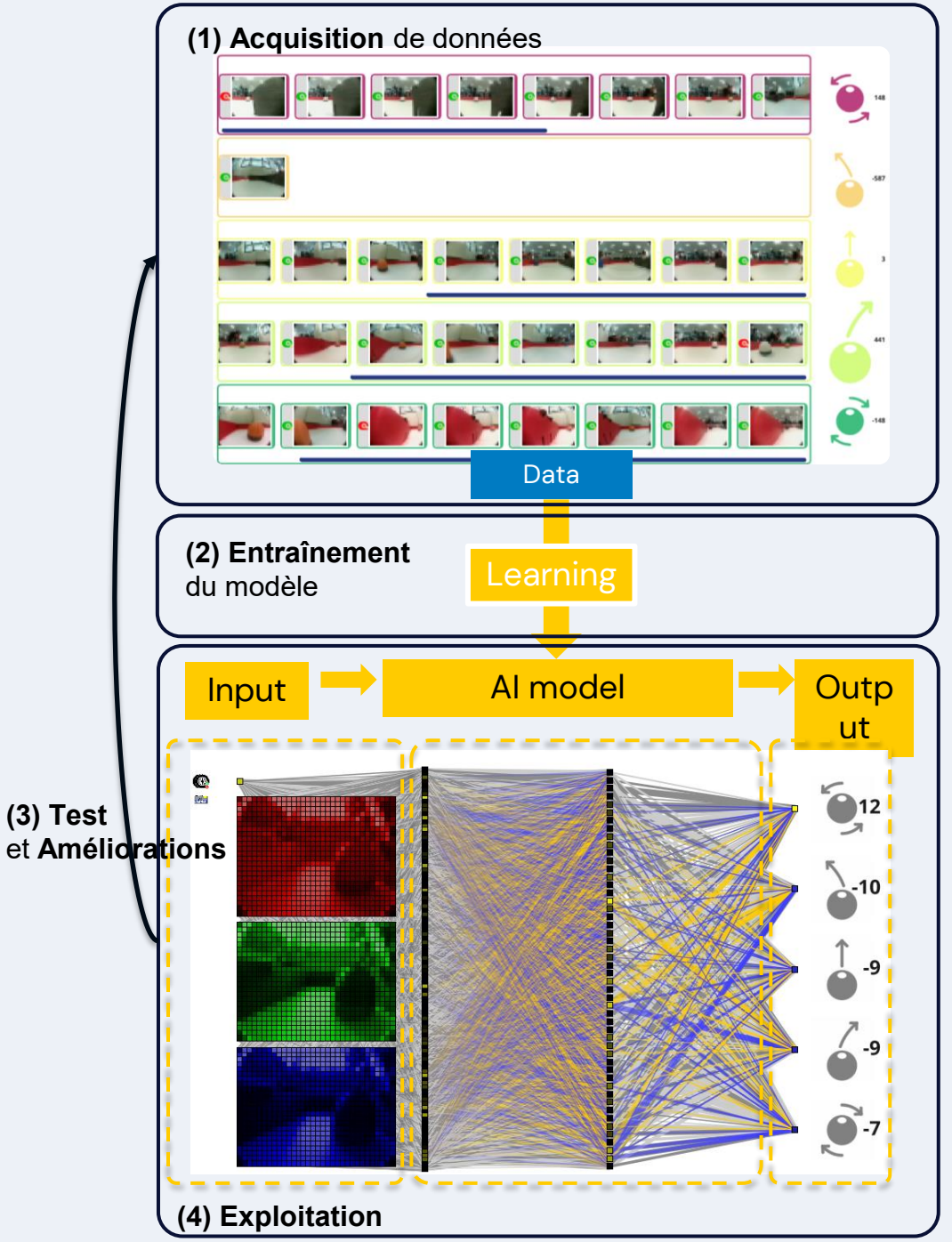
2022 : 1 classe de 28 étudiants
2023, 2024, 2025 : 4 classes (~90 étudiants) chaque année

| | | |
|---|---|---|
| TP1 | Course de robots | *découvrir l'apprentissage supervisé en entraînant un robot dans une "course de robots" (on entraîne en pilotant le robot, c'est à dire en faisant l'acquisition du training set)* |
| TP2 | Programmation du robot en python | *programmation très basique pour apprendre les bases de python* |
| TP3 | Perfectionnement en python | *programmation de la décision du robot en fonction de la couleur d'un des pixels de la caméra* |
| TP4 | Algorithme des K plus proches voisins | *découverte de l'algorithme KNN dans notre interface graphique* |
| TP5 | Programmation des K plus proches voisins | *programmation de KNN en python (le code de l'élève est ensuite chargé dans notre logiciel, qui permet de l'utiliser pour entraîner le robot)* |
| TP6 | TP évalué | |
| TP7 | Détection d'intrus | *Edition manuelle dans le logiciel des poids d'un mini-réseau de neurones devant actionner une alarme lorsque une distance mesurée est trop faible ou trop forte (la fonction modélisée étant non linéaire, on découvre la nécessité d'utiliser une couche d'au moins 2 neurones intermédiaires)* |
| TP8 | Réseaux de neurones avec scikit-learn | *Reproduction de l'activité avec ANN construit avec scikit-learn dans le code élève* |
| TP9 | Q-learning avec AlphAI | *Découverte du Q-learning dans le logiciel (le robot apprend en apprentissage par renforcement à avancer par défaut, mais tourner en présence d'obstacle ; table Q-learning 2 états obstacle / pas d'obstacle x 5 actions)* |
| TP10 | Programmation Q-learning | *Programmation de l'algo dans un cas particulièrement simplifié* |
| TP11 | Récapitulatif | *Cours avec slides récapitulatives sur les notions apprises et slides* |
| TP12 | TP évalué | |
| Examen | | |

# ① Découvrir par la manipulation

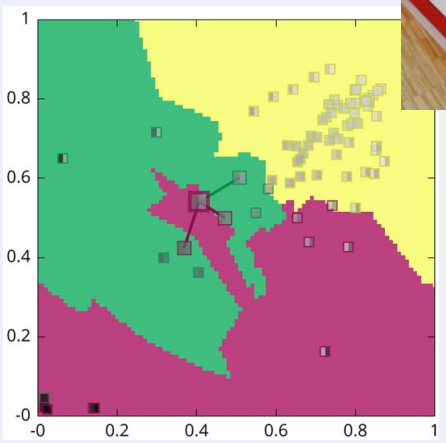
Séance 1 : Course de robots autonomes !!



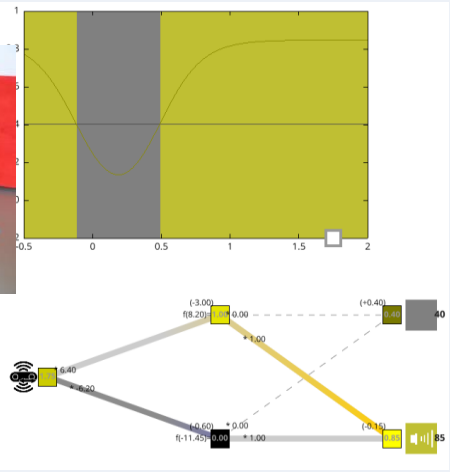


# ② Visualiser & Comprendre

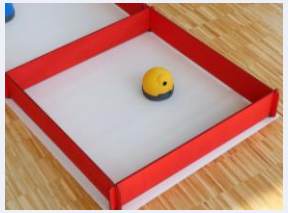
## K plus proches voisins

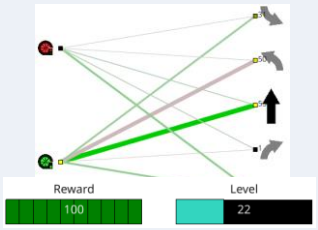



## Réseaux de neurones



## Apprentissage par Renforcement : Q-learning

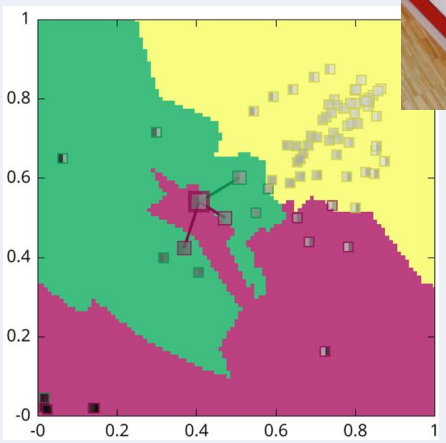




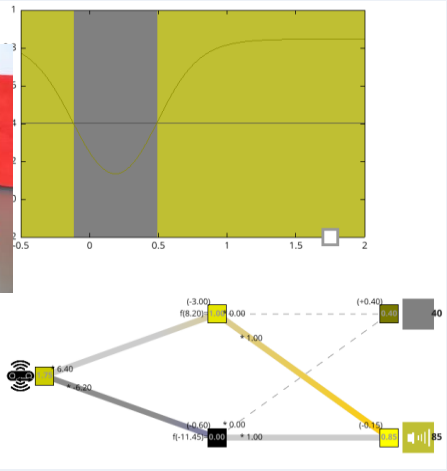
# ③ Programmation Python des algorithmes


```python
from random import uniform

def print_table(table):
    for line in table:
        for value in line:
            print(round(value, 2), end=' ')
        print()  # Nouvelle ligne

def init(n_state: int, n_action: int):
    global q_table
    q_table = [[uniform(-1,1) for j in range(n_action)] for i in range(n_state)]
    print_table(q_table)

def init(n_state: int, n_action: int):
    global q_table
    q_table = []
    for i in range(n_state):
        line = []
        for j in range(n_action):
            line.append(uniform(-1,1))
        q_table.append(line)
    print_table(q_table)

def learn(old_state: int, action: int, reward: float, new_state: int):
    alpha = 0.2
    gamma = 0.8
    target = (1 - gamma) * reward + gamma * max(q_table[new_state])
    q_table[old_state][action] = (1 - alpha) * q_table[old_state][action] + alpha * target
    print_table(q_table)

def take_decision(state: int) -> int:
    """
    Determine the appropriate action knowing the current state (or perform
    exploration!).
    @param state: an integer coding for the current state.
    @return: the index of the chosen action (starting from 0 at the top).
    """
    action = 0
    return action

def get_reward(sensors: list, speed: float) -> float:
    on_line = sum(sensors) > 0  # Booléen indiquant si le robot est sur la ligne ou non.
    if on_line:
        # Le capteur central rapporte plus de points, les capteurs latéraux moins.
        reward = sensors[2] + 0.6 * (sensors[1] + sensors[3]) + 0.3 * (sensors[0] + sensors[4])
        # La récompense est proportionelle à la vitesse.
        reward *= speed
    else:
        # La récompense est négative, mais encourage la marche arrière.
        reward = -speed - max_speed
    return reward
```

## K plus proches voisins




## Réseaux de neurones




## Apprentissage par Renforcement : Q-learning