ENSAM – Casablanca **Année Universitaire 2025-2026**
Département Génie Informatique **Pr. Soukaina Bouhsissin**

Intelligence Artificielle

TP 2 : Agent Réactif dans une Grille

Objectifs du TP

- Comprendre la prise de décision locale dans un environnement inconnu.
- Implémenter un agent réactif se déplaçant dans une grille.
- Simuler les capteurs (murs, obstacles) et les actions (avancer, tourner).
- Observer les limites des agents sans mémoire.

Un robot est placé dans une **grille (grid)** de taille 8×8.

Chaque case vaut 0 (libre) ou 1 (mur / wall)

Le robot commence en S = (1, 1) et doit atteindre G = (6, 6)

Il peut se déplacer Nord, Est, Sud, Ouest

Il suit une priorité : droite → avant → gauche → demi-tour

Exemple de labyrinth:

```
LAB = [
 [1,1,1,1,1,1,1,1],
 [1,0,0,0,1,0,0,1],
 [1,0,1,0,0,0,1,1],
 [1,0,1,0,1,0,0,1],
 [1,0,0,0,1,1,0,1],
 [1,1,1,0,0,0,0,1],
 [1,0,0,0,1,0,0,1],
 [1,1,1,1,1,1,1,1]
]
```

Partie 1 : Capteurs et Mouvements

Dans cette partie, vous allez implémenter les fonctions de base permettant au robot de percevoir son environnement et de se déplacer.

Question 1.1 : Implémentation des fonctions de base

Implémentez les trois fonctions suivantes :

- **in_bounds(G, i, j)** : vérifie si la position (i, j) est dans les limites de la grille G
- **is_wall(G, i, j)** : vérifie si la position (i, j) est un mur (valeur 1) ou hors limites

- **move(i, j, d)** : calcule la nouvelle position après un déplacement dans la direction d

Note : Les directions sont codées par : DIRS = [(-1,0), (0,1), (1,0), (0,-1)] pour N, E, S, O

```
DIRS = [(-1,0), (0,1), (1,0), (0,-1)]  # N, E, S, O

def in_bounds(G, i, j):
    return

def is_wall(G, i, j):
    return

def move(i, j, d):
     return
```

Question 1.2 : Différence entre in_bounds et is_wall

Quelle est la différence entre les fonctions in_bounds et is_wall ? Pourquoi avons-nous besoin des deux ?

Question 1.3 : Comportement hors limites

Que se passe-t-il si le robot essaie de sortir de la grille ? Comment votre implémentation gère-t-elle ce cas ?

Partie 2 : Décision Locale (Règles Simples)

Dans cette partie, vous allez implémenter la stratégie de décision de l'agent réactif basée sur la règle de la **"main droite"**.

Question 2.1: Implémentation de choose_action

Implémentez la fonction choose_action(G, i, j, d) qui choisit la prochaine direction selon la priorité : droite → avant → gauche → demi-tour.

```
# Stratégie "main droite" : priorité droite > avant > gauche > demi-tour
def choose_action(G, i, j, d):
    return d
```

Question 2.2 : Description de la stratégie

Décrivez en vos propres mots la stratégie utilisée par l'agent pour choisir sa direction.

Question 2.3 : Ordre de priorité

Pourquoi tester les directions dans l'ordre [droite, avant, gauche, demi-tour] ? Qu'est-ce que cela simule ?

Question 2.4 : Cas bloqué

Que se passe-t-il si toutes les directions sont bloquées (murs partout) ? Comment votre fonction gère-t-elle ce cas ?

Question 2.5 : Variante "main gauche"

Proposez une modification de la fonction choose_action pour implémenter une stratégie "main gauche" au lieu de "main droite".

Partie 3 : Simulation du Déplacement

Dans cette partie, vous allez implémenter la boucle principale de simulation qui fait avancer le robot jusqu'au but (ou jusqu'à atteindre le nombre maximum d'étapes).

Question 3.1 : Implémentation de run_agent

Implémentez la fonction run_agent(G, start, goal, d0=0, max_steps=200) qui simule le déplacement de l'agent.

```
def run_agent(G, start, goal, d0=0, max_steps=200):
    return path

path = run_agent(LAB, S, G, d0=1)
print('Longueur du chemin :', len(path))
```

Question 3.2 : Résultats de l'exécution

Exécutez le programme plusieurs fois. Notez :

- Le nombre d'étapes pour atteindre le but
- La longueur du chemin parcouru
- Le comportement observé

Question 3.3 : But atteint ?

Le but a-t-il été atteint ? Si non, pourquoi ? Si oui, le chemin est-il optimal ?

Partie 4 : Visualisation du Chemin

Pour mieux comprendre le comportement de l'agent, nous allons visualiser sa trajectoire sur la grille.

Question 4 : Implémentation de print_path

Implémentez la fonction print_path(G, path, S, Ggoal) qui affiche la grille avec :

- 'S ' pour le départ
- 'G ' pour le but
- '▉ ' pour les murs
- 'V ' pour les cases visitées
- ' ' (espaces) pour les cases libres non visitées

```
def print_path(G, path, S, Ggoal):

print_path(LAB, path, S, G)
```

Partie 5 : Analyse et Réflexion

Cette dernière partie est consacrée à l'analyse critique du comportement de votre agent réactif.

Question 5.1 : Garantie d'atteindre le but

L'agent atteint-il toujours le but ? Pourquoi ou pourquoi pas ? Donnez un exemple de configuration où il échouerait.

Question 5.2 : Problème des boucles infinies

Que se passe-t-il si l'agent tourne en boucle (visite les mêmes cases indéfiniment) ? Comment pourriez-vous détecter ce problème ?

Question 5.3 : Besoin de mémoire

L'agent a-t-il besoin d'une mémoire ? Si oui, quelle forme devrait-elle prendre ? Que devrait-il mémoriser ?

Question 5.4 : Amélioration de la stratégie

Comment pourriez-vous améliorer la stratégie de l'agent ? Proposez une solution utilisant un historique des positions visitées.