FACULTÉ DES SCIENCES
DÉPARTEMENT D'INFORMATIQUE

UNIVERSITÉ LIBRE DE BRUXELLES ULB

INFO-F-311: Intelligence Artificielle

Projet 1: Recherche

2024-2025

Axel Abels **Tom Lenaerts** **Yannick Molinghen** **Pascal Tribel**

1. Préambule

Dans ce projet, vous allez implémenter des techniques d'intelligence artificielle basées sur de la recherche dans des graphes. On vous fournit des fichiers de base pour le projet que vous pouvez les télécharger sur l'université virtuelle.

L'utilisation d'outils tels que ChatGPT est autorisée, et nous vous demandons d'expliquer brièvement comment vous les avez utilisés en Section 3.

1.1. Environnement

L'environnement dans lequel vous allez travailler et qui est illustré dans la Figure 1 s'appelle le *Laser Learning Environment* (LLE). Dans LLE, plusieurs agents se déplacent sur une grille pour collecter des gemmes puis atteindre les cases de fin (encadrée en noir en bas à droite dans la Figure 1). Notez qu'un agent qui atteint la sortie ne peut plus bouger de la partie.

Le dynamique principale de LLE s'articule autour des lasers. Des lasers de couleur bloquent le passage aux agents d'une autre couleur, mais un agent de la même couleur peut bloquer le rayon laser pour permettre à d'autres agents de passer, comme illustré dans la Figure 1.

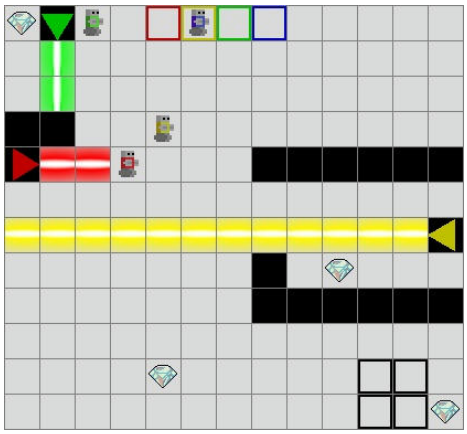

Figure 1. – Rendu de LLE

1.1.1. Utiliser LLE

LLE nécessite une version de Python ≥ 3.10. Vous pouvez installer LLE votre gestionnaire de modules Python préféré, typiquement `pip` (ou équivalent comme `uv`, `poetry`, `conda`, ...):

```
pip install laser-learning-environment
```

Le code ci-dessous vous montre comment interagir et visualiser l'environnement comme dans le Listing 1.

```python
import cv2
from lle import World, Action

def show(world: World):
  img = world.get_image()
  cv2.imshow("Visualisation", img)
  cv2.waitKey(1)

world = World.level(1)
world.reset()
show(world)
path = [Action.SOUTH] * 5
path += [Action.EAST] * 3
path += [Action.SOUTH] * 5
path += [Action.WEST] * 3
for action in path:
  events = world.step(action)
  print(events)
  show(world)
  input("Appuyez sur 'enter' pour passer à l'action suivante...")

for agent in world.agents:
  print(agent.has_arrived)
  print(agent.is_alive)
```

Listing 1. – Exemple d'utilisation de LLE

1.2. Tests automatiques

Nous vous fournissons des tests automatiques pour tester votre implémentation. Pour les lancer, vous devez installer `pytest` :

```
pip install pytest
```

Vous pouvez tester vos méthodes avec la commande `pytest`. Pour lancer uniquement les tests d'un fichier en particulier, vous pouvez le donner en paramètre :

```
pytest tests/tests_bfs.py
```

Il y a un fichier de test pour chacune des tâches que vous devez remplir.

1.3. Fichiers existants

On vous donne les fichiers `search.py` et `priority_queue.py` ainsi que ceux du module `problem`. Le fichier `priority_queue.py` est là pour vous éviter d'implémenter une file à priorités et vous ne devriez pas avoir besoin de le modifier. Implémentez la modélisation des problèmes de recherche (Section 2.1) dans les fichiers du module `problem` et les algorithmes de recherche (Section 2.2) dans `search.py`.

2. Objectifs du projet

Ce projet est divisé en deux grandes parties:

- la modélisation de plusieurs problèmes de recherche (Section 2.1)
- l'implémentation d'algorithmes de recherche (Section 2.2)

Vous lancer dans le projet, nous vous conseillons commencer par la modélisation du `ExitProblem` (Section 2.1.1) puis de passer à l'implémentation des algorithmes de recherche BFS et DFS (Section 2.2). Lorsque ceux-ci fonctionnent, passez à l'implémentation d'A^* et des questions d'heuristique. Une fois que tous les algorithmes sont fonctionnels sur le `ExitProblem`, passez à la modélisation des problèmes plus complexes.

2.1. Modélisation

La modélisation des problèmes de recherche se fait dans le module `problem`. Votre première tâche consiste à définir le concept d'état « successeurs » à un autre état s, c'est-à-dire les états que l'on peut atteindre depuis s en effectuant une action. Pour ce faire, complétez la méthode `get_successors` de classe abstraite `SearchProblem` dans le module `problem`.

2.1.1. Trouver la sortie

Vous allez maintenant vous attaquer au premier problème de recherche qui consiste à ce que les agents atteignent la sortie (vivants). Implémentez la méthode `is_goal_state(problem_state)` en conséquence. Cette méthode prend un `WorldState` en paramètre et détermine si l'objectif du `ExitProblem` a été atteint.

En ce qui concerne l'heuristique, vous pouvez utiliser la distance de Manhattan.

Conseil: N'oubliez pas que l'environnement peut accepter plusieurs agents. Par conséquent, même s'il n'y a qu'un seul agent, `world.agents_positions` et `world.exit_pos` renvoient des listes de positions.

2.1.2. Collecter les gemmes

Pour ce deuxième problème, modélisez le problème qui consiste à collecter toutes les gemmes de l'environnement puis à rejoindre les cases de sortie. Complétez la classe `GemProblem` en ce sens, et testez-la avec vos algorithmes de recherche.

2.1.3. Trouver les coins

Modélisez le problème qui consiste à passer par les quatre coins du `World` puis à atteindre une sortie. Pour ce faire, complétez la classe `CornerProblem` et testez-la avec vos algorithmes de recherche.

Vous constaterez rapidement que la classe `WorldState` ne contient pas assez d'information pour résoudre le `CornerProblem`. Créez une nouvelle classe `CornerState` qui contient toutes les informations dont vous avez besoin pour pouvoir résoudre ce problème. Comme cette classe sera différente pour chacun·e, il n'y a pas de tests concernant la modélisation du problème mais uniquement sur sa résolution.

2.2. Algorithmes de recherche

Avant d'implémenter les algorithmes de recherche, pensez à tester votre modélisation avec les tests unitaires, comme décrit dans la Section 1.2.

1. Implémentez l'algorithme du parcours en largeur (*Breadth First Search*, BFS) dans le fichier `search.py` via la fonction `bfs`.
2. Implémentez l'algorithme du parcours en profondeur (*Depth First Search*, DFS) dans le fichier `search.py` via la fonction `dfs`.

3. Implémentez l'algorithme de recherche A^* dans le fichier `search` via la fonction `astar`. N'oubliez pas d'implémenter la méthode `heuristic` dans chaque `Problem`.

Conseil: Si vous stockez vos états de recherche dans un ensemble ou dans un dictionnaire, n'oubliez pas d'y implémenter correctement les méthodes `__hash__` et `__eq__` des objets que vous y stockez, sinon, votre algorithme va boucler indéfiniment.

3. Rapport

On vous demande d'écrire un court rapport (2 à 4 pages) dans lequel vous allez mener trois expériences. Chaque expérience a pour but de comparer BFS, DFS et A^* sur le `GemProblem` selon un critère spécifique: la taille du chemin trouvé, la durée d'exécution et le nombre d'états visités.

Pour mener ces expériences, vous allez devoir créer vous-même un ensemble de 4 cartes personnalisées[1] *intéressantes* pour la résolution du `GemProblem`. Dans votre rapport, veillez à choisir judicieusement le type de graphique (ligne, barrettes, camembert, ...) et à les soigner en termes de titre, de légende, d'échelle, de cohérence des couleurs, ...

3.1. Mode opératoire

- Expliquez quelle a été votre démarche pour créer vos cartes et présentez-en en image.
- Expliquez l'idée derrière les heuristiques utilisées pour le `GemProblem` et pour le `CornerProblem.`

3.2. Expériences

Pour les quatre cartes que vous avez conçues, représentez graphiquement[2]

1. la longueur du chemin
2. la durée moyenne d'exécution
3. le nombre de nœuds visités

dans trois graphiques différents. Pour la durée d'exécution (dont la mesure peut varier d'une fois à l'autre, veillez à reproduire l'expérience plusieurs fois et à indiquer combien de fois vous avez reproduit l'expérience. La représentation graphique de l'écart type ou d'un intervalle de confiance est un plus.

3.3. Discussion

- Discutez des résultats de vos graphiques: étaient-ils attendus ou non ? Quelles conclusions peut-on en tirer ?
- Si vous aviez un budget de temps de 5 minutes, quelle taille de problème (en nombre de nœuds) chaque algorithme peut-il résoudre ? A quelle taille de carte de LLE cela correspond-il ?

3.4. LLMs

Expliquez brièvement si vous avez utilisé des outils tels que ChatGPT et dans quelle mesure.

Remise

Le livrable de ce projet se présente sous la forme d'un fichier zip votre code Python et votre rapport au format PDF.

Le travail est **individuel** et doit être rendu sur l'Université Virtuelle pour le 30/09/2024 à 23:59.

[1] Référez-vous à la documentation.
[2] Utilisez une librairie comme matplotlib ou seaborn.