FACULTÉ DES SCIENCES
DÉPARTEMENT D'INFORMATIQUE

UNIVERSITÉ LIBRE DE BRUXELLES ULB

INFO-F-311: Intelligence Artificielle

Projet 1: Recherche

Yannick Molinghen **Pascal Tribel** **Tom Lenaerts**

1. Préambule

Dans ce projet, vous allez implémenter des techniques d'intelligence articifielle 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 5.

1.1. Poetry

Le projet nécessite Python 3.10 et utilise le système de build Poetry (https://python-poetry.org/docs/#installation).

Après avoir installé Poetry, ouvrez un terminal dans le répertoire du projet et lancez les commandes:

```
poetry shell
poetry install
```

Ces commandes vont automatiquement créer un environnement virtuel puis installer les dépendances du projet.

Important: Par la suite, lorsque vous voudrezlancer les tests ou un script du projet, faites attention à bien lancer la commande `poetry shell`, sinon vous n'aurez pas les bonnes dépendences installées et vous aurez des erreurs d'import. Vous pouvez vérifier que vous êtes dans un poetry shell grâce au préfixe dans le terminal qui ressemble à `(info-f-311-search-py3.10) user@computer`.

1.2. Tests automatiques

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`, `problem.py` et `priority_queue.py`, mais vous ne devrez travailler que dans les deux premiers. `search.py` contient les algorithmes de recherche que vous allez implémenter tandis que `problem.py` contient la définition des problèmes que vous allez résoudre.

1.4. L'environnement

Vous allez travailler avec un environnement appelé "Laser Learning Environment", illustré dans la Figure 1. Dans cet environnement, entre 1 et 4 agents se déplacent et collectent des gemmes sur une grille. 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.

Les cases de départ sont représentées par des carrés de la couleur de l'agent qui commence (en haut, au milieu), et les cases de sortie sont indiquées par des cases encadrées en noir (en bas à droite). Un

agent qui atteint la sortie ne peut plus bouger de la partie, et le jeu est terminé quand tous les agents ont atteint la sortie.

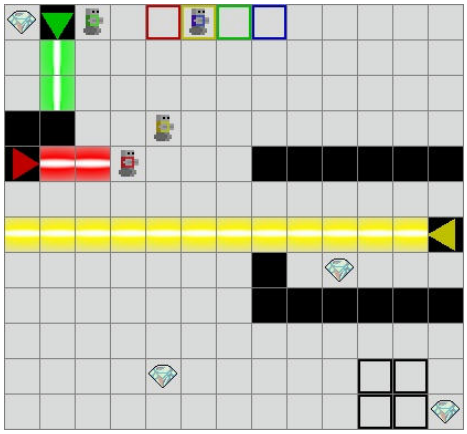

Figure 1: Rendu de l'environnement

Le code ci-dessous vous montre comment visualiser l'environnement comme dans la Figure 1.

```
import cv2
from lle import World

img = world.get_image()
cv2.imshow("Visualisation", img)
# Utilisez waitKey avec 0 pour bloquer et attendre que l'utilisateur appuie sur 'enter'
ou avec 1 pour continuer dans le code.
cv2.waitKey(0) # Attend que l'utilisateur appuie sur 'enter'
cv2.waitKey(1) # continue l'exécution du code
```

Rapport de bugs: L'environnement LLE est encore récent, et quelques bugs pourraient subsister. Dans le cas où vous pensez en avoir trouvé un (dans l'environnement ou dans les tests), n'hésitez pas à le signaler par email à l'adresse yannick.molinghen@ulb.be en joignant le code minimal nécessaire à le reproduire. Chaque première personne à rapporter un bug (et qui est confirmé) donnera lieu à un point supplémentaire avec un maximum de 2 points sur le projet.

2. `SimpleSearchProblem`

2.1. Modélisation du problème

Ce problème de recherche vise à atteindre la fin de l'environnement, c'est-à-dire faire en sorte que tous les agents aient atteint une sortie. Pour ce faire:

1. Implémentez la méthode `is_goal_state(problem_state)` en conséquence. Cette méthode prend un `SimpleStateProblem` en paramètre et détermine si l'objectif du `SimpleSearchProblem` a été atteint.
2. Implémentez la méthode `get_successors(current_state)` qui renvoie les états que l'on peut atteindre depuis l'état actuel.

Conseils:

- Fiez-vous à la documentation indiquée dans le code, que ce soient la documentation des fonctions et méthodes ou leur indication de type (*type hint*).
- N'oubliez pas que l'environnement est prévu pour plusieurs joueurs. Même s'il n'y a qu'un joueur, `world.agent_positions` renverra une liste de positions. De même, la méthode `world.step()` prend en paramètre une liste d'actions.
- Jetez un oeil aux méthodes et attributs `world.step()`, `world.available_actions()`, `world.get_state()`, `world.set_state()` et `world.done`.
- N'hésitez pas à définir vos propres classes, et n'oubliez pas d'y implémenter correctement les méthodes `__hash__` et `__eq__` afin de bien fonctionner avec des ensembles et dictionnaires.

2.2. Recherche en largeur

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

2.3. Recherche en profondeur

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

2.4. Recherche A^*

Implémentez l'algorithme A^* dans le fichier `search.py` via la fonction `astar`. Utilisez la distance de mahattan comme heuristique, et implémentez-la dans la méthode `heuristic(state)` de la classe `SimpleSearchProblem`.

3. `CornerSearchProblem`

3.1. Modélisation

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

3.2. Heuristique

Dans la class `CornerSearchProblem`, implémentez une heuristique cohérente (*consistent*) plus efficace que la distance de Mahnattan pour résoudre le `CornerSearchProblem`.

4. `GemSearchProblem`

4.1. Modélisation

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

4.2. Heuristique

Dans la classe `GemSearchProblem`, implémentez une autre heuristique cohérente (*consistent*) plus efficace que la distance de Manhattan pour résoudre le `GemSearchProblem`.

5. Rapport

On vous demande d'écrire un court rapport (maximum 2 pages) dans lequel vous

- reportez et comparez la taille des chemins trouvés pour les trois algorithmes de recherche sur le niveau 3.
- comparez le nombre de noeuds étendus au cours de la recherche pour BFS, DSF et A$*$ lors de la résolution du niveau 3 et discutez des raisons de ces différences.
- expliquez l'idée derrière les heuristiques que vous avez développées pour le `CornerSearchProblem` et pour le `GemSearchProblem`, en montrant quelques exemples qui comparents les valeurs des heuristiques.
- expliquez si vous avez utilisé des outils tels que ChatGPT et comment vous les avez utilisés.

Remise

Le livrable de ce projet se présente sous la forme d'un fichier zip contenant trois fichiers: `search.py`, `problem.py` et votre rapport au format PDF.

Le travail est **individuel** et doit être rendu sur l'Université Virtuelle pour le 08/10/2024 à 23:59.

Note: Veillez à **ne pas** inclure dans la remise

- les fichiers de votre éditeur de code
- les fichiers `.pyc`
- le répertoire `.git`