FACULTÉ DES SCIENCES
DÉPARTEMENT D'INFORMATIQUE

UNIVERSITÉ LIBRE DE BRUXELLES ULB

INFO-F-311: Intelligence Artificielle

Projet 2: Recherche adversariale

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 en considérant un ou plusieurs adversaires. On vous fournit des fichiers de base pour le projet que vous pouvez télécharger sur l'université virtuelle.

1.1. Mise en place

Pour faciliter le démarrage du projet, nous vous fournissons un fichier `pyproject.toml` qui comprend les dépendances du projet que vous pouvez soit installer manuellement avec `pip`, soit en utilisant un gestionnaire de dépendances comme `uv`.

1.1.1. UV

`uv` est un outil de gestion de projets et de dépendances Python qui vise notamment à être extrêmement rapide. Installez `uv` et téléchargez les dépendances du projet comme indiqué ci-dessous.

```
# Pour Linux/MacOS
curl -LsSf https://astral.sh/uv/install.sh | sh
# Pour Windows
powershell -ExecutionPolicy ByPass -c "irm https://astral.sh/uv/install.ps1 | iex"
uv sync
```

La commande `uv sync` va lire le fichier `pyproject.toml`, créer un environnement virtuel adéquat et y installer toutes les dépendances indiquées.

Si vous avez besoin d'ajouter une dépendance `x` à votre projet vous pouvez les ajouter avec `uv add x`. Par exemple, pour ajouter `numpy`, vous feriez:

`uv add numpy`

Pour mettre à jour vos dépendances du projet, faites:

`uv sync -U`

1.1.2. Manuellement

Vous pouvez aussi installer les dépendances manuellement avec `pip`.

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. Laser Learning Environment

Comme lors du premier projet, vous allez utiliser la librairie `lle` illustrée dans la Figure 1 pour ce projet. N'hésitez pas à consulter la documentation en ligne ainsi que les notebooks jupyter d'exemple pour vous familiariser à son utilisation.

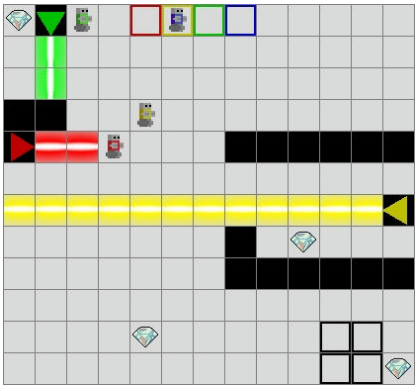

Figure 1. – Rendu de l'environnement

Rapport de bugs: Dans le cas où vous trouveriez des bugs dans `LLE`, n'hésitez pas à les faire remonter sous forme d'*issue* github avec un code minimal qui permet de reproduire le bug. Chaque bug confirmé donnera lieu à un point supplémentaire à la personne qui l'a rapporté avec un maximum de 2 points sur le projet.

2. Un `World` compétitif

`LLE` a été adapté en mode compétitif (implémenté dans le fichier `competitive_world.py`) dans lequel exactement deux agents s'affrontent pour remporter un maximum de points. Dans cet environnement, les agents agissent l'un après l'autre, chacun à son tour, en commençant par l'agent 0 que vous incarnez. Lorsqu'un agent meurt, il ne peut plus effectuer que l'action `STAY` jusqu'à la fin de la partie.

Une partie s'achève dès que l'un des deux agents atteint une tuile de sortie.

2.1. Évaluation d'un état

Dans les algorithmes de recherche adversariale, il est nécessaire d'évaluer la valeur d'un état afin de prendre la meilleure action possible. Ici, la valeur d'un état est calculée comme la différence entre les points accumulés par l'agent 0 et ceux accumulés par l'agent 1.

Au fur et à mesure d'une partie, chaque agent accumule des points en fonction de ses actions:

- Collecter une gemme rapporte +1 point
- Atteindre une sortie rapporte +1 point
- Mourir rapporte −1 point

La valeur V d'un état s se calcule comme indiqué dans l'Équation 1.

$$V(s) = \text{points}(\text{agent}_0) - \text{points}(\text{agent}_1) \tag{1}$$

3. Algorithmes de recherche

Dans les algorithmes de recherche que vous allez implémenter, vous devez systématiquement renvoyer l'action à effectuer pour l'agent 0. Par conséquent, ces fonctions n'acceptent que des états pour lesquels c'est à l'agent 0 de jouer et lèvent une `ValueError` avec un message approprié dans le cas contraire.

Important: Vous pouvez modifier **TOUT** ce que vous voulez dans le code qui vous est donné à l'exception des tests unitaires. Ce code vous est uniquement donné comme guide.

3.1. Algorithmes déterministes

Dans le fichier `adversarial_search.py`, implémentez respectivement l'algorithme du minimax et $\alpha\beta$ -pruning dans les fonctions `minimax` et `alpha_beta`.

3.2. Expectimax

Dans minimax et $\alpha\beta$-pruning, nous avons supposé que l'adversaire agissait de manière optimale. Cependant, ce n'est pas toujours le cas pour des humains. L'algorithme « expectimax » permet de modéliser le comportement probabiliste d'humains qui pourraient prendre des choix sous-optimaux. La nature de l'algorithme « expectimax » demande que nous connaissions la probabilité que l'adversaire prenne chaque action. Nous allons ici supposer que l'adversaire prend des actions uniformément aléatoires.

4. Rapport

Écrivez un rapport (de préférence à l'aide d'un outil d'écriture scientifique tel que Typst ou Latex) dans lequel vous présentez les expériences décrites dans la Section 4.1, c'est-à-dire votre mode opératoire, les paramètres de vos expériences, les résultats, une discussion des résultats, …

4.1. Expériences

On vous demande de réaliser une expérience dans laquelle vous faites s'affronter vos algorithmes sur trois cartes de votre conception *judicieusement choisies*.

Écrivez le code nécessaire à ce que deux joueurs (incarnés par des algorithmes) s'affrontent dans un `CompetitiveWorld`. Implémentez aussi un algorithme qui prend des décisions aléatoires.

Sur chacune de vos cartes, faites s'affronter vos algorithmes contre un agent aléatoire. Pour chaque algorithme et pour des profondeurs de recherche allant de 1 à 10, reportez graphiquement le score final obtenu (Équation 1).

Note: Veillez à ce que vos résultats soient pertinents:

- Répétez vos expériences plusieurs fois si leur issue peut varier d'une fois à l'autre. Le cas échéant, représentez la moyenne ainsi que l'écart type (ou un intervalle de confiance).
- Lorsque vous répétez l'expérience, intervertissez les places des agents au cas où un agent aurait un avantage sur l'autre à cause de la carte.

Ensuite, choisissez trois combinaisons d'algorithmes (à l'exception de l'aléatoire), faites-les s'affronter sur vos cartes, et reportez le résultat graphiquement.

Remise

Le livrable de ce projet se présente sous la forme d'un fichier zip contenant les sources Python du projet ainsi que votre rapport au format PDF.

Le travail est **individuel** et doit être rendu sur l'Université Virtuelle pour le 14/10/2024 à 23:59.