FACULTÉ DES SCIENCES
DÉPARTEMENT D'INFORMATIQUE

UNIVERSITÉ LIBRE DE BRUXELLES ULB

# INFO-F-311: Intelligence Artificielle

## Projet 4: Reinforcement Learning

**Axel Abels** [axel.abels@ulb.be]
**Yannick Molinghen**
**Pascal Tribel**
**Tom Lenaerts**

## 1 Préambule

Dans ce projet, vous allez implémenter des algorithmes d'apprentissage par renforcement (reinforcement learning). On vous fournit des fichiers de base pour le projet que vous pouvez télécharger sur l'université virtuelle.

### 1.1 Poetry

Le projet nécessite Python $\geq 3.10$ 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 update
```

Ces commandes vont automatiquement créer un environnement virtuel puis installer les dépendances du projet. Vous pouvez aussi utiliser `pip` pour installer les dépendances du projet en regardant les dépendances à installer dans le fichier `pyproject.toml`.

## 2 L'environnement

Dans ce projet, vous allez entraîner plusieurs algorithmes de sorte à leur faire résoudre le labyrinthe illustré dans la Figure 1, où les cases grises représentent des emplacements libres sur lesquelles l'agent peut se déplacer et les cases noires représentent des murs.

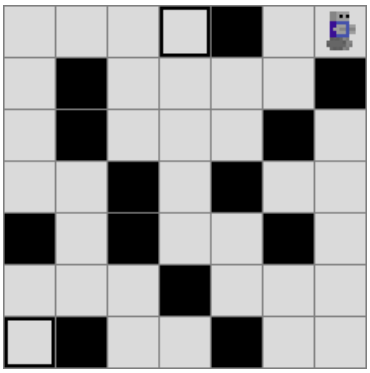

Figure 1. – Illustration du labyrinthe à résoudre

Lors de chaque nouvel épisode, l'agent sera positionné en haut à droite du labyrinthe. Le but de l'agent est de sortir du labyrinthe; soit par la sortie du haut, soit par la sortie plus gratifiante en bas à gauche. La fonction de récompense est représentée par l'Équation 1.

$$R(s,a,s') = \begin{cases} 10 \text{ si } s' \text{ est la sortie supérieure} \\ 100 \text{ si } s' \text{ est la sortie inférieure gauche} \\ -1 \text{ sinon} \end{cases} \quad (1)$$

Si $\gamma = 1$, la stratégie optimale sera donc de se rendre à la sortie inférieure gauche.

**Non-déterminisme**

L'environnement étant corrompu, il y a à chaque pas une probabilité $p$ que l'agent soit déplacé aléatoirement dans le labyrinthe au lieu de se retrouver là ou son action l'aurait mené.

La classe `Labyrinth` dans le fichier `env.py` vous permet d'instancier et d'interagir avec l'environnement. La fonction `random_moves` dans le fichier `main.py` vous donne un exemple minimal d'interaction.

**Notes:**

- La mise à jour de l'affichage (`env.render()`) peut être relativement lente, évitez donc d'y faire appel si vous n'en avez pas besoin.
- Puisqu'il y a de l'aléatoire, deux exécutions d'un même algorithme pourraient donner deux résultats différents. Afin de tester un cas particulier plusieurs fois, vous pouvez fixer une graine (seed) à l'aide de `np.random.seed(seed)`. Si vous en faites usage, n'oubliez pas de générer vos résultats sur des seeds différentes!

# 3 Algorithmes

Vous devez implémenter deux algorithmes sur la résolution du labyrinthe: value iteration et q-learning.

## 3.1 Value iteration

L'algorithme de «value iteration» a pour but d'approximer itérativement la valeur $V(s)$ de chaque état s à l'aide de l'Équation 2.

$$V_{k+1}(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_k(s')] \tag{2}$$

Malheureusement, vous n'avez pas accès à la fonction de transition ($T$). Votre implémentation devra donc l'estimer en interagissant avec l'environnement.

L'algorithme de value iteration prend en paramètre un entier $k$ qui détermine combien d'itérations effectuer. Implémentez l'algorithme *value iteration* dans le fichier `value_iteration.py`.

**Notes:**

- Quand vous implémentez l'algorithme, faites attention à vous baser sur $V_k$ pour calculer $V_{k+1}$ et à ne pas modifier $V_k$ durant l'itération.
- N'oubliez pas que la valeur d'un état terminal est 0 par définition.
- Vous retrouverez dans main.py une fonction d'aide *plot_values* qui permet d'afficher une heatmap représentant les valeurs de chaque état.

## 3.2 Q-learning

L'algorithme de Q-learning consiste à interagir avec l'environnement pour mettre à jour une fonction d'évaluation $Q(s,a)$. Une fois que votre agent est entraîné, vous pouvez exploiter la stratégie apprise en prenant l'action ayant la plus haute q-value dans chaque état.

Pour entraîner votre agent, il est nécessaire d'explorer l'environnement. Pour ce faire, vous devrez implémenter deux stratégies d'exploration.

Pour entraîner la fonction $Q(s,a)$, on emploie une méthode adaptée de l'équation de Bellman (c.f le cours théorique) illustrée dans l'Équation 3, où $\alpha$ est le *learning rate* et $\gamma$ est le *discount factor*.

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left[R(s,a,s') + \gamma \max_{\{a'\}} Q(s',a')\right] \tag{3}$$

**Note:**
- Comme pour la value iteration, n'oubliez pas que $V(s') = \max_{\{a'\}} Q(s',a') = 0$ lorsque $s'$ est un état terminal.

### $\varepsilon$-greedy

Cette stratégie naïve consiste à choisir une action aléatoire avec une probabilité ε, et à prendre la meilleure action (celle maximisant $Q(s,a)$) avec une probabilité $1-\varepsilon$.

Bien que facile à implémenter, cette stratégie n'adapte pas son degré d'exploration à notre degré d'incertitude. Pour y remédier, vous devrez également implémenter un bonus d'exploration.

### Bonus d'exploration

L'exploration fonctionnelle vise à améliorer la stratégie $\varepsilon$-greedy en adaptant le niveau d'exploration en fonction de l'incertitude associée à chaque action. Au lieu de maintenir un taux d'exploration fixe, une fonction d'exploration ajuste dynamiquement la probabilité de sélectionner des actions moins connues ou peu explorées, favorisant ainsi l'acquisition d'informations sur l'environnement.

Une approche possible consiste à utiliser une fonction qui augmente artificiellement la valeur des actions rarement choisies, incitant l'algorithme à les explorer davantage. Par exemple, dans un cadre d'apprentissage par renforcement, on pourrait utiliser une fonction qui ajuste la valeur estimée $Q(s,a)$ de chaque action $a$ dans un état $s$ en fonction du nombre de fois que cette action a été sélectionnée. Ainsi, au lieu de sélectionner l'action maximizant $Q(s,a)$, on selectionnera

$$a = \arg\max_{\{a\}} Q(s,a) + \frac{c}{\{N(s,a)+1\}} \tag{4}$$

où $N(s,a)$ est le nombre de fois que l'action $a$ a été effectuée à l'état $s$, et $c$ est une constante qui contrôle le taux d'exploration. Intuitivement, au plus une action est effectuée dans un état, au plus petit le bonus d'exploration $\frac{c}{\{N(s',a')+1\}}$ sera.

En ajoutant également ce bonus lors de la mise à jour de $Q$, on propage les bonus d'exploration en arrière, ce qui permet de valoriser les états et actions qui mènent à des états et actions peu explorés. Ainsi, on remplacera l'Équation 3 par

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left[R(s,a,s') + \gamma \max_{\{a'\}} Q(s',a') + \frac{c}{\{N(s',a')+1\}}\right] \tag{5}$$

Implémentez ces deux variantes ($\varepsilon$-greedy et bonus d'exploration) de l'algorithme de Q-learning dans le fichier `qlearning.py`.

**Note:**
- Notez que les valeurs habituelles pour $\gamma$ sont comprises entre 0.9 et 0.999, et entre 0.0001 et 0.1 pour $\alpha$. Dans vos expériences, veillez à utiliser des valeurs cohérentes.
- Vous retrouverez dans main.py une fonction d'aide *plot_qvalues* qui permet d'afficher une heatmap représentant les q-valeurs maximales de chaque état ainsi que la direction qui y correspond.

# 4 Rapport

On vous demande d'écrire un rapport concis, structuré en sections et sous-sections, et qui comprend un texte suivi (par opposition à des réponses courtes sans contexte).

N'oubliez pas de préciser les paramètres utilisés lors de vos expériences. De plus, afin que vos expériences soient reproductibles, veillez à créer une fonction par expérience dans le fichier `main.py`. Sauf indication contraire, le taux d'aléatoire de l'environnement ($p$, *malfunction_probability* dans le code) devra être fixé à 0.1.

## 4.1 Value iteration

Après avoir entraîné votre algorithme sur le labyrinthe, représentez sous forme de *heatmap*[1] (Figure 2) la valeur de chaque état du labyrinthe après 10, 20, 30 et 40 étapes d'entraînement.

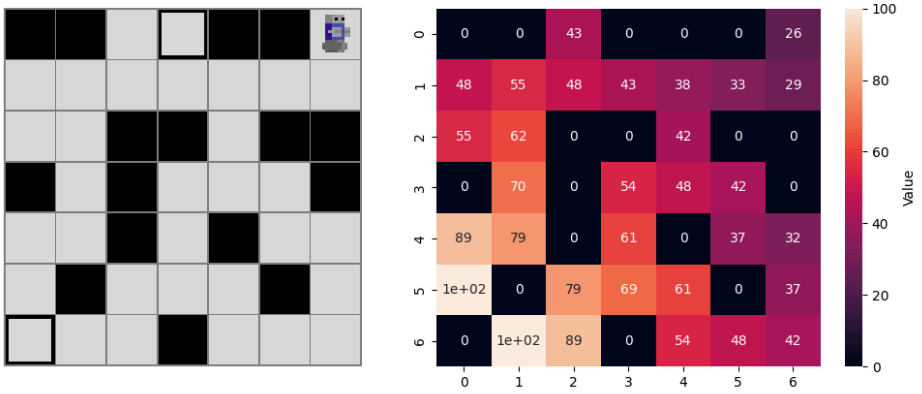


Figure 2. – Exemple de *heatmap* obtenu pour un autre labyrinthe.

## 4.2 Q-learning

### Effet de l'exploration

On vous demande de tracer sur un même graphique le score (la somme des récompenses au cours d'un épisode) moyen au cours de l'entraînement de votre agent $\varepsilon$-greedy pour les paramètres suivants:

- $c = 0, \varepsilon = 0.2$
- $c = 0, \varepsilon = 0.1$
- $c = 0, \varepsilon = 0.01$
- $c = 0, \varepsilon$ diminue linéairement de 1 à 0.01 au cours de l'entraînement

Et, dans un 2ème graphe à la même échelle, les variantes à bonus d'exploration avec les paramètres suivants:

- $\varepsilon = 0, c = 1$
- $\varepsilon = 0, c = 10$
- $\varepsilon = 0, c = 100$
- $\varepsilon = 0, c = 1000$

Pour chaque paramétrisation, entraînez vingt fois un agent depuis zéro pendant 10 000 étapes[2] et faites la moyenne des scores au cours du temps.

[1] Par exemple avec seaborn ou matplotlib.
[2] Une étape correspond à une action.

**Note:** Une partie de votre travail consiste à fixer certains paramètres ($\alpha$, nombre d'étapes d'entraînement, ...) et à déterminer comment représenter les données (quel traitement effectuer, quel type de graphique utiliser). Veillez à documenter votre méthodologie.

Pour chaque paramétrisation ci-dessus, générez une heatmap indiquant le nombre de fois que chaque état a été exploré. Afin de faciliter l'interprétation, utilisez une échelle logarithmique pour les couleurs. De plus, assurez vous que l'échelle est la même pour toutes les heatmaps.

**Sensibilité aux paramètres**

Choisissez une des paramétrisations qui parvient systématiquement à atteindre la sortie du bas. En fixant les autres paramètres, tracez sur un graphique la récompense moyenne par action (sur 20 répétitions de 10 000 étapes d'entraînement) en fonction d'un discount factor ($\gamma \in \{0.5, 0.75, 0.9, 0.95, 0.975, 0.99, 1\}$) croissant. Faites de même pour le taux d'apprentissage ($\alpha \in \{0, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1\}$), ainsi que pour le taux d'aléatoire de l'environnement ($p \in \{0, 0.01, 0.03, 0.1, 0.3, 1\}$). Pour les paramètres pertinents, tracez également la performance d'un agent entraîné par value itération ($k = 40$ étapes).

### 4.3 Discussion

- Comment l'aspect aléatoire de l'environnement affecte l'apprentissage? De quelle façon les méthodes de value iteration et q-learning gèrent-elles le non-déterminisme? En d'autre termes, l'aspect aléatoire de l'environnement est-il encodé dans les valeurs apprises? Si oui, comment?
- À mesure que l'aléatoire de l'environnement (la probabilité de non-déterminisme $p$) augmente, quel impact cela a-t-il sur la capacité de chaque algorithme à apprendre ? L'un des algorithmes gère-t-il mieux un niveau de hasard élevé ? Justifiez votre réponse à l'aide de vos résultats.
- Comment les stratégies d'exploration $\varepsilon$-greedy et avec bonus d'exploration influencent-elles l'équilibre entre exploration et exploitation ? Laquelle de ces stratégies semble atteindre un meilleur équilibre, et pourquoi ? Soutenez vos arguments à l'aide des graphes et heatmaps.
- Lorsque l'agent converge, il prend le chemin le plus court vers une sortie. Quels mécanismes lui permettent de trouver le chemin le plus court?
- Quel est l'effet du discount factor $\gamma$? À partir de quelle valeur la sortie optimale change-t-elle? Justifiez votre réponse à l'aide de vos résultats.
- Quel est l'effet du taux d'apprentissage $\alpha$? Justifiez votre réponse à l'aide de vos résultats.

## Remise

Le livrable de ce projet se présente sous la forme d'un fichier zip contenant vos sources python ainsi que votre rapport en PDF. Nous vous encourageons à utiliser un outil tel que Typst ou Latex (par exemple avec Overleaf) pour rédiger votre rapport.

Ce travail est **individuel** et doit être rendu sur l'Université Virtuelle pour le 01/12/2024 à 23:59.

Veuillez adresser vos questions à Axel Abels (axel.abels@ulb.be).