ULB UNIVERSITÉ LIBRE DE BRUXELLES

INFO-F311 - Projet d'IA 3

Apprentissage par renforcement

Auteur:
Manuel Rocca - 000596086

Professeurs:
Tom Lenaerts
Assistants:
Axel Abels
Martin Colot
Yannick Molinghen
Pascal Tribel

Année académique 2025-2026

Contents

1 Introduction

Pour ce troisième projet du cours d'Intelligence Artificielle, nous avons implémenté des algorithmes d'apprentissage par renforcement (*reinforcement learning* en anglais). Nous utilisons toujours l'environnement *LLE* comme pour les parties 1 et 2.

Trois différences (ou ajouts) sont introduites: les cases tourbillons (où l'agent meurt) la fonction de récompense, et surtout, l'aspect stochastique, non-déterministe de l'environnement.

1.1 Environnement stochastique

Un environnement stochastique est un environnement où, lorsque l'agent effectue une action, nous ne sommes pas certains de l'état dans lequel il va arriver. En pratique, si un agent dans un état s souhaite effectuer une action a pour arriver dans un nouvel état s', il y a une probabilité p qu'une autre action aléatoire soit prise.

1.2 La fonction de récompense

Une fonction de récompense est une fonction qui, pour chaque transition d'état, octroie une valeur. En d'autres termes, une valeur numérique est associée à chaque action prise par l'agent afin de le diriger de manière idéalement souhaitée.

Associer une valeur adéquate est tout sauf quelque chose de simple. Des comportements inattendu peuvent survenir si cette fonction est mal définie (par exemple: si un agent doit atteindre une sortie avec une récompense positive mais que le coût de vie, le coût par pas est très élevé, il aura plus tendance à chercher un moyen de faire le moins de pas possible en trouvant un endroit plus proche pour mourir permettant une minimisation de son score total).

2 Value Iteration

L'algorithme *Value Iteration* utilise l'équation de Bellman (1) de manière itérative pour mettre à jour les valeurs estimées de chaque état de l'environnement.

$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s') \right] \tag{1}$$

Équation de Bellman caractérisant les valeurs optimales

Les itérations de l'algorithme sont bornées par une valeur δ comme suit:

$$\max_{s \in S} |V_{k+1}(s) - V_k(s)| < \delta \tag{2}$$

En d'autres termes, à chaque itération, la différence/variation entre les valeurs des états précédents et les nouvelles valeurs est calculée en appliquant l'équation 1. Si celle-ci est inférieure du seuil δ donné, l'algorithme s'arrête.
L'algorithme converge après un certain nombre d'itérations. Ce nombre peut être, dans certains cas, très grand voire quasi infini. C'est pourquoi il est intéressant d'établir un seuil de variation maximal afin d'arrêter l'algorithme après une durée d'exécution raisonnable. Comme nous l'avons vu au cours théorique, parfois un dixième des itérations suffisent à obtenir des valeurs pratiquement égales aux valeurs obtenues à la convergence.

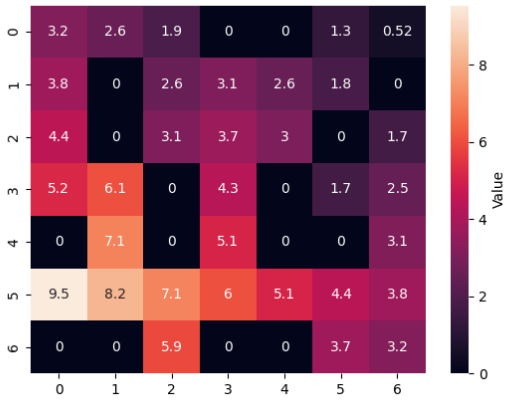
2.1 Entraînement de l'algorithme

Dans cette section, nous nous intéressons aux valeurs d'états finales obtenues par notre implémentation de l'algorithme *Value Iteration* pour des valeurs de $\delta \in \{1, 0.1, 0.01, 0.005, 0.001\}$ ainsi que le nombre d'itérations k. Pour chaque valeur de δ testée, nous usons comme facteur de réduction $\gamma = 0.9$.

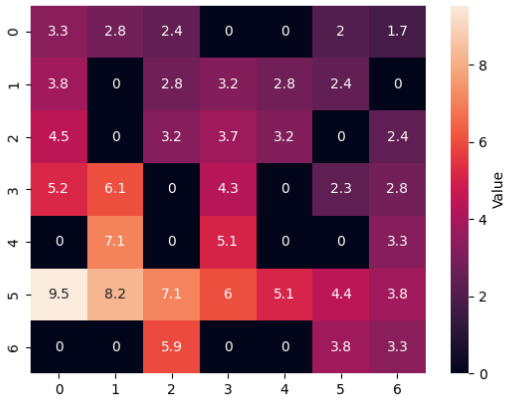
δ	k
1	10
0.1	14
0.01	18
0.005	18
0.001	20

Table 1: Nombre d'itérations de l'algorithme *Value Iteration* en fonction d'un δ donné.

Plus il y a d'itérations, plus nous nous approchons de la convergence. Voici les heatmaps pour $\delta = 1$ et $\delta = 0.001$ affichant les valeurs d'états dans l'environnement stochastique après k itérations:



(a) $\delta = 1$, $k = 10$



(b) $\delta = 0.001$, $k = 20$

Nous nous rendons très vite compte que la valeurs des états les plus éloignés de l'état initial en $(0, 6)$ (en haut à droite) convergent plus rapidement que ceux plus proches de l'état d'origine. Cela s'explique par le fait que toutes les valeurs d'états calculées par *Value Iteration* auront une valeur nulle tant qu'une récompense n'a pas été trouvée. Démontrons cela de manière concise:

Démonstration. En observant l'équation 1, il est clair que la valeur est nulle dans deux cas:

- si $T(s, a, s')$ est nulle. Or cette valeur n'est jamais nulle, il y a toujours une probabilité donnée de passer d'un état s à un état s' suivant.
- si $R(s, a, s') + \gamma V^*(s')$ est nul. Initialement tous les $V(s)$ sont nuls, donc le seul terme qui peut donner une valeur non-nulle à cette expression est la récompense $R(s, a, s')$ (γ est une constante non-nulle).

En somme, les valeurs d'état calculées se propagent à partir des transitions accordant une récompense expliquant ainsi la convergence plus rapide dans ces environs. □

Pour conclure cette section nous souhaitons proposer une analyse du nombre d'itérations requises pour atteindre la convergence. Pour ce faire, nous avons changé la condition d'arrêt de notre algorithme. Dorénavant, il s'arrête dès que la variation entre les valeurs d'états de deux itérations successives est nulle. Ce faisant, nous sommes arrivés à $k = 50$ (et une heatmap exactement identique à celle obtenue avec un $\delta = 0.001$ en $k = 20$ itérations).

Ceci complète donc l'hypothèse fournie dans au cours théorique concernant l'utilité d'un grand nombre d'itérations. En effet, après la moitié du nombre d'itérations requises pour obtenir la convergence nous obtenons déjà des valeurs similaires voire identiques.

2.2 Stratégie optimale

3 Q-learning

L'algorithme *Q-learning* calcule, pour chaque état et pour chaque action possible à partir de cet état, une valeur réelle. À partir de cette valeur réelle, il détermine une stratégie optimale à suivre en prenant l'action ayant la valeur la plus élevée pour chaque état.

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

Mise à jour de la *q-value* en un état s en prenant une action a, avec $V(s) = \max_{a \in A} Q(s,a)$

Contrairement à *Value Iteration* qui ne considère que des transitions en n'interagissant pas avec l'environnement, *Q-learning* fait un apprentissage *offline* en interagissant directement avec l'environnement. En d'autres termes, si l'agent souhaite déterminer la récompense obtenue en passant de son état s à un nouvel état s' avec une action a il doit exécuter cette action. Donc s'il doit mourir pour savoir que ce n'est pas bon, il le fera.

Une autre différence par rapport au *Value Iteration* est le choix des actions. Le *Q-learning* choisit l'action suivante sur base d'une politique bien déterminée. Il nous a été demandé d'en implémenter deux:

- Exploration ε-greedy: il y a une probabilité ε de choisir une action aléatoire et une probabilité $1 - \varepsilon$ de prendre une action qui maximise $Q(s,a)$. Ceci est formalisé par l'équation fournie dans les consignes reprise ci-dessous (avec r un nombre uniformément aléatoire entre 0 et 1):

$$\pi(s) = \begin{cases} \arg\max_{a \in A} Q(s,a) & \text{si } r \leq \varepsilon \\ a \sim A & \text{sinon} \end{cases} \tag{4}$$

  En fait, ε est une variable qui, plus elle est élevée, plus elle favorise l'exploration et, plus elle est basse, plus elle favorise l'exploitation. L'exploration est lorque l'agent teste une action aléatoire indépendamment des *q-values* déjà calculées pour en découvrir de nouvelles (et potentiellement découvrir d'autres récompenses). L'exploitation fait tout l'inverse. Elle choisit l'action avec la *q-value* la plus élevée. Elle "ne prend pas de risques".

- Exploration softmax: utilise la fonction softmax adaptée à notre environnement. L'équation fournie dans les consignes est la suivante (avec τ la "température"):

$$\pi_a(s) = \frac{e^{\frac{Q(s,a)}{\tau}}}{\sum_{a' \in A} e^{\frac{Q(s,a')}{\tau}}} \tag{5}$$

En termes concrets, pour un état s donné, l'équation va calculer, pour chaque action a disponible, une certaine probabilité. Donc chaque action sera pondérée d'une probabilité p. Nous choisissons ensuite une action aléatoirement en prenant en compte des poids associés.

3.1 Entraînement de l'algorithme

Nous analysons dans cette section les performances de l'agent dans l'environnement en utilisant l'algorithme *Q-learning* sur 20 000 itérations, interactions avec l'environnement en utilisant des paramètres et des politiques différentes.

Les paramètres généraux sont:

- l'environnement: probabilité de prendre une action aléatoire au lieu de celle choisie $p = 0.1$
- l'algorithme: *learning rate*[1] $\alpha = 0.1$ et *discount factor* $\gamma = 0.9$

3.1.1 Exploration ε-greedy

Comme exprimé ci-dessus, chaque expérience comporte 20 000 itérations. Nous utilisons un ε différent par expérience. Nous en faisons quatre. Pour les trois premières, nous utilisons $\varepsilon \in \{0, 0.1, 0.5\}$ tandis que pour la dernière nous faisons diminuer ε linéairement de 1 à 0.01 au cours de l'entraînement.

3.1.2 Exploration softmax

Pour cette politique d'exploration nous utilisons les valeurs de température $\tau \in \{0.01, 1, 10\}$ pour les trois premières expériences. La dernière fait usage d'une fonction diminuant exponentiellement[2] allant de 100 à 0.01 au cours de l'entraînement.

3.1.3 Résultats

Voici le graphique reprenant les résultats des huit expériences menées:

4 Discussion

Cette section vise à répondre aux questionnements initiés à la section 4.3. des consignes.

[1] Taux définissant à quel point une ancienne valeur remplace la nouvelle; un taux plus élevé permet à l'agent de s'adapter plus rapidement aux nouvelles valeurs calculées tandis qu'un taux plus bas nous aurons des résultats plus stables mais un apprentissage plus lent.

[2] Pour ce faire nous avons utilisé l'expression $f(x) = f(x_0)e^{-kx}$. Nous avons $f(x)$, x et x_0. Il suffit donc de résoudre une équation pour obtenir finalement $k = \frac{4\ln(10)}{20000}$.

4.1 Aspect aléatoire

L'algorithme *Value Iteration*, ne comprend pas d'aspect aléatoire. En effet pour calculer la valeur d'un état s, nous considérons chaque action a possible à partir de s pour arriver à différents états s'. Chaque action possède une certaine probabilité d'être effectuée mais comme nous travaillons sur toutes les transitions possibles (la somme de tous les états s'), nous arrivons à "une probabilité totale de 1". Il faut plutôt voir les probabilités associées à chaque action a comme un poids dans l'expression permettant de mieux valuer une transition d'un état s à un état s' avec une action a. Pour concrétiser tout ça, nous avons retiré la seed rendant ainsi notre algorithme en théorie aléatoire. Or, à chaque exécution (avec les mêmes paramètres bien entendu), nous avons obtenus des heatmaps complètement identiques.

À l'opposé, l'algorithme *Q-learning* interagit directement avec l'environnement qui possède une probabilité p d'effectuer une action aléatoire au lieu de l'action souhaitée. De plus, les politiques d'exploration ont également un aspect aléatoire. En effet, la politique ε-greedy comprend une probabilité ε de choisir une action aléatoire (encourageant l'exploration) et, la politique softmax qui calcule un poids, une probabilité pour chaque action avant d'en sélectionner une aléatoirement en prenant en compte le poids calculé.

4.2 Stratégies d'exploration

5 Conclusion