LU3IN013 - Projet de recherche
Rapport intermédiaire

Sujet 2 : Modèle de l'activation des souvenirs pour la planification et l'apprentissage de la navigation dans un labyrinthe

---

Encadrant : Olivier SIGAUD

Hoang Hieu LE (28707917)
Margaux MARSELOO (28630611)
Gabriella RAKOTOARISON (28631085)

Notre code : https://github.com/GabyRkt/Prioritized-Memory-Access

1 Introduction

À travers ce projet, nous étudions un article de neurosciences computationnelles écrit par Marcelo G. Mattar et Nathaniel D. Daw. Dans ce dernier, ils tentent d'expliquer par le biais de l'apprentissage par renforcement comment un rat se sert de sa mémoire pour apprendre à naviguer et à planifier ses déplacements dans un labyrinthe.

Notre objectif consiste à bien comprendre le modèle des auteurs afin de pouvoir reproduire les résultats de cet article le plus fidèlement possible.

2 Résumé de l'article

L'article écrit par Mattar et Daw est centré sur le rôle de l'hippocampe dans la planification et la prise de décision. Il propose un modèle informatique d'un tel apprentissage.

L'hippocampe est une structure cérébrale qui participe à la formation de la mémoire et à la navigation spatiale. Les auteurs s'appuient sur des recherches antérieures qui suggèrent que l'hippocampe rejoue les souvenirs notamment avant ou après une expérience ou encore pendant les périodes de sommeil. Ils font l'hypothèse que ce processus de relecture n'est pas aléatoire. Il suivrait un système de hiérarchisation des priorités.

La théorie avancée par Mattar et Daw suggère que le cerveau attribue différentes priorités à différents souvenirs en fonction de leur valeur ou de leur utilité perçue. Ce processus d'accès prioritaire à la mémoire est censé aider le cerveau à consolider et à renforcer les souvenirs, ainsi qu'à faciliter la planification et la prise de décision en permettant à l'agent de récupérer et de rejouer les souvenirs pertinents aux moments opportuns.

Pour rendre compte de cette théorie, les auteurs ont cherché à modéliser ce comportement à l'aide de l'apprentissage par renforcement. Ils réalisent la simulation d'une tâche de navigation spatiale au cours de laquelle un agent génère et stocke ses expériences. Il les utilise ultérieurement pour calculer la meilleure direction à prendre dans l'objectif d'atteindre une récompense. Ainsi, il anticipe les meilleurs choix en fonction des conséquences des expériences précédentes. La principale caractéristique du comportement adaptatif de l'agent est l'utilisation efficace de l'expérience pour maximiser la récompense.

Finalement, le but de leur article est de comprendre comment le cerveau hiérarchise, consolide et rejoue les souvenirs, pour mieux appréhender comment sont faits les choix et la planification des actions futures, ou encore comment les différentes fonctions de relecture des souvenirs s'associent, à quels instants et dans quel ordre, pour promouvoir un comportement adaptatif. Ils ont par ailleurs montré qu'il existait un équilibre entre les différents types de remémorations des expériences, entre la planification (*forward replay*) et la consolidation des souvenirs (*backward replay*), leurs contenus et leurs effets sur les futures décisions.

3 Travaux

3.1 Apprentissage par renforcement

Lorsqu'il s'agit de planification et d'apprentissage de la navigation, il est fréquent d'utiliser des méthodes d'apprentissage par renforcement. Afin d'en apprendre plus sur ces concepts, nous avons effectué des travaux pratiques sur la programmation dynamique et l'apprentissage par renforcement proposés par Olivier Sigaud aux étudiants de Master de Sorbonne Université.

L'apprentissage par renforcement est un type de mécanisme d'apprentissage qui repose sur un processus de décision markovien (MDP). Dans ce contexte, nous représentons le labyrinthe par un graphe composé d'un ensemble d'états S et d'un ensemble d'actions A. Aussi, l'agent considéré ne

connaît pas les fonctions de transition T et de récompense R.

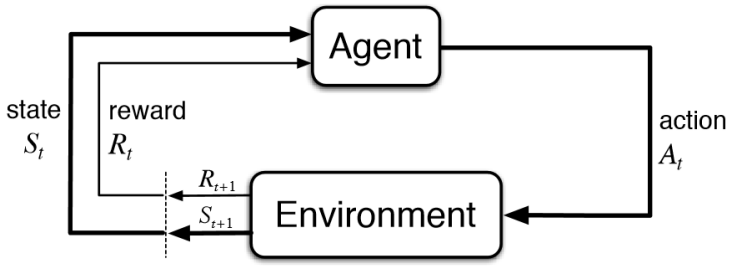


Figure 1: Markov Decision Process (MDP)

En réalisant une action, l'agent passe d'un état s à un état s' et reçoit une récompense r. Le but de l'agent est d'établir une stratégie, appelée politique, qui maximise la récompense cumulative attendue. La politique détermine la probabilité de choisir une action a dans un état s. Pour apprendre cette politique, plusieurs algorithmes existent dont *Q-learning* qui est utilisé dans cet article.

Dans *Q-learning*, l'agent apprend une fonction état-action Q qui permet de déterminer la valeur *Q(s,a)* apportée par le choix d'une action a dans un certain état s. Il utilise ensuite la *Q-table*, les valeurs *Q(s,a)*, pour mettre à jour sa politique actuelle. La méthode *Q-learning* converge vers une stratégie optimale.

Dans ce projet, nous utilisons une version modifié de *Q-learning*, le modèle DYNA-Q. Ce dernier fait partie de l'architecture DYNA, une famille d'algorithmes *model-based*, ce qui signifie que l'agent apprend la matrice de transisitions au cours de ses expériences. Ainsi, DYNA-Q permet d'accélérer l'apprentissage *Q-learning* traditionnel en incorporant des mises à jour supplémentaires se basant sur un couple *(s,a)* choisi dans la matrice de transisions.

Ce type d'algorithmes nécessite des mécanismes d'exploration pour éviter une convergence locale prématurée. La politique ε-*greedy* est une politique qui choisit la meilleure action (celle avec la valeur la plus élevée) avec une probabilité de 1-$\varepsilon \in [0,1]$ et une action aléatoire avec une probabilité ε. Lorsque cet algorithme va choisir une action aléatoire, il va donc considérer toutes ces actions également bonnes même si certaines sont meilleures que d'autres. Pour pallier ce problème, la politique *softmax* peut être utilisée. Elle consiste à sélectionner les actions aléatoires avec des probabilités proportionnelles à leurs valeurs.

Pour calculer les valeurs de la fonction Q, les auteurs utilisent des *Bellman Backups*. Ce processus consiste à faire des mises à jour locales en utilisant une relation de récurrence entre la valeur de *Q(s,a)* avec s l'état actuel et a l'action choisie par l'agent, la récompense immédiatement reçue après ce pas et la valeur estimée de Q pour l'état d'arrivée. Cette règle de mise à jour couramment utilisée s'appelle l'apprentissage par différence temporelle (*TD*). Appliquée pour chaque état que visite l'agent, elle permet de propager les informations sur une récompense obtenue aux états et aux actions qui la précèdent. Dans *Q-learning*, seul le couple *(s,a)* courant, peut être mis à jour via un *Bellman backup*.

3.2 Notions importantes de l'article

Le *forward replay* correspond à la planification ou à la prédiction c'est-à-dire la mise à jour de ce que l'agent pense être son futur immédiat. Les modèles de relecture des souvenirs "vers l'avant" suggèrent qu'ils reflètent les expériences précédentes et les rejoue pour anticiper le chemin qu'il va emprunter.

Le *backward replay* correspond à la mise à jour des expériences passées. La relecture des souvenirs "vers l'arrière" commence à l'endroit où se trouve l'agent dans l'environnement et retrace

typiquement son chemin à l'envers.

L'*offline replay* se produit durant le sommeil, afin de consolider ses expériences.

Dans cet article, Mattar et Daw veulent montrer que ces replays ont lieu à des moments bien précis. Le *backward replay* se produirait lorsque l'animal rencontre une récompense, il veut maintenant propager cette information de récompense "vers l'arrière" pour savoir comment retourner à l'emplacement de la récompense. A l'inverse, le *forward replay* se produirait lorsque l'animal a besoin de savoir où aller. Il produit des trajectoires mentales qui ressemblent à la planification. Néanmoins, nous travaillons toujours sur la compréhension de cette partie de l'algorithme. En effet, nous n'avons pas encore compris comment les auteurs arrivent à prouver que les *backward replays* ont lieu en fin d'expérience et les *forward replays* en début d'expérience car dans notre compréhension actuelle du code, ce résultat n'est pas produit uniquement par le mécanisme qu'ils veulent mettre en avant mais par des conditions implémenter directement dans leur programme.

La *trace d'éligibilité* conserve un historique des pas précédents les plus récents. Cela permet de mettre à jour en même temps plusieurs estimations de la fonction Q lorsque l'agent découvre une récompense. Ces mises à jour sont pondérées par la récence et la fréquence des couples état-action considérés.

Le *replay buffer* a pour rôle de mémoriser tous les mouvements effectués par l'agent par ordre chronologique. Un mouvement ou un pas de l'agent est conservé sous la forme *(s, a, r, s')* avec s l'état initial, a l'action choisie par l'agent, s' l'état d'arrivée et r la récompense obtenue suite à ce pas. Ces souvenirs sont par la suite utilisés pour calculer le *gain* et le *need*.

Le *gain* est défini comme la récompense supplémentaire que peut espérer récolter l'agent en faisant une action dans un état donné. Il dépend directement des changements de politique. Il donne alors la priorité aux états précédemment parcourus par l'agent lorsqu'un résultat inattendu est rencontré.

Le *need* correspond au nombre de fois où on peut s'attendre à voir passer l'agent sur une case. Il donne la priorité aux états directement accessibles par l'agent qui sont immédiatement pertinents. Les auteurs ont choisi de l'estimer à partir du concept de *"Successor Representation"* (article de Peter Dayan). Le *need* est donc calculé à partir de la fonction de transition T.

L'*EVB*, ou *Expected Value of Backup*, correspond à l'amélioration attendue à la suite d'un changement de politique. Il s'agit d'une modification de *Q-learning* qui sert à prioriser certaines expériences passées. Elle simule un accès en mémoire. Nous calculons l'*EVB*, ou l'*utilité* de faire un *Bellman backup*, pour toute expérience, soit pour tout couple *(s,a)*. Celle-ci se calcule en multipliant le *gain* et le *need* associés à chaque expérience. Ensuite, nous choisissons l'expérience ayant l'*utilité* maximale. Cette expérience est considérée comme prioritaire et fait l'objet d'un *Bellman backup* supplémentaire pour permettre à l'agent de prendre de meilleurs décisions dans le future.

Dans cette simulation, nous avons vu que les expériences présentes dans la *trace d'éligibilité*, ainsi que celle ayant la plus grande *utilité* sont également mises à jour indépendamment du pas effectif de l'agent. Cela permet d'une part une propagation plus rapide des *Q-values* et d'autre part la valorisation des couples *(s,a)* pertinents.

3.3 Implémentation

Afin de réaliser ce projet, nous nous sommes inspirés de nombreux codes notamment celui des auteurs écrit en *matlab* et d'un code produit par des étudiants en Master 2. Aussi, nous avons choisi d'utiliser la bibliothèque *SimpleMazeMDP* accessible sur le GitHub d'Olivier Sigaud pour modéliser les deux environnements discrets et déterministes, proposés par Mattar et Daw, dans lesquels va évoluer notre agent.

Le premier environnement, intitulé *Linear Track* (figure 2) est représenté par deux segments

linéaires disjoints, avec une récompense aux extrémités opposées. Le but de ce labyrinthe est de simuler une tâche très utilisée dans les études de remémorations des souvenirs par l'hippocampe : l'agent doit effectuer des allers-retours pour collecter des récompenses aux extrémités. L'agent commence à l'état (0,0). Après avoir reçu une récompense à (0,9), il est transporté vers le deuxième segment à l'emplacement (2,9). De manière analogue, en atteignant la récompense à (2,0), il sera ramené à l'état (0,0) et ainsi de suite. Les segments, bien que similaires, servent à représenter la distinction de l'espace d'état en termes d'emplacements ainsi que de directions de déplacements, simulant la façon dont les cellules de lieu hippocampiques différencieraient les états dans ce type d'environnement.

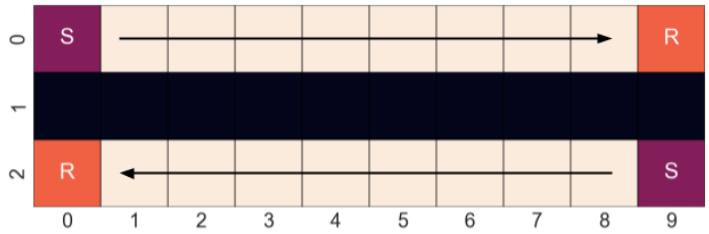


Figure 2: Représentation de Linear Track d'après Matter et Daw les états de départ sont notés "S" et les récompenses "R

Le second environnement, appelé *Open Field* (figure 3), est un labyrinthe de taille 6 par 9 contenant des obstacles. Contrairement à l'environnement précédent, l'état de départ d'*Open Field* est choisi aléatoirement après chaque épisode. Le but de l'agent est de naviguer dans le labyrinthe afin de trouver une récompense placée à un endroit fixe (0,8). Cette tâche est largement utilisée dans les études d'apprentissage par renforcement utilisant le cadre DYNA qui correspond à la simulation *random replay* (figure 6 et figure 7).

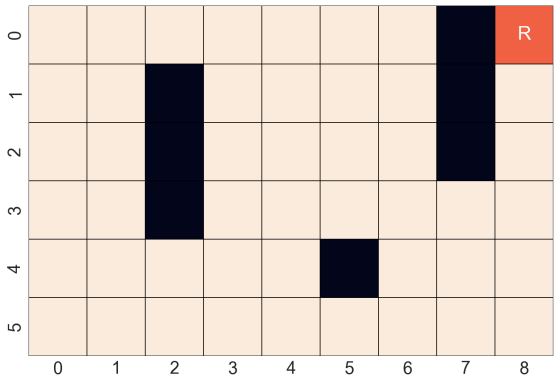


Figure 3: Représentation du labyrinthe Open Field

Nous avons commencé par implémenter en *python* le programme des auteurs écrit en *matlab*. Nous avons essayé de le restructurer en identifiant les différents objets, puis en divisant la simulation en plusieurs fonctions distinctes.

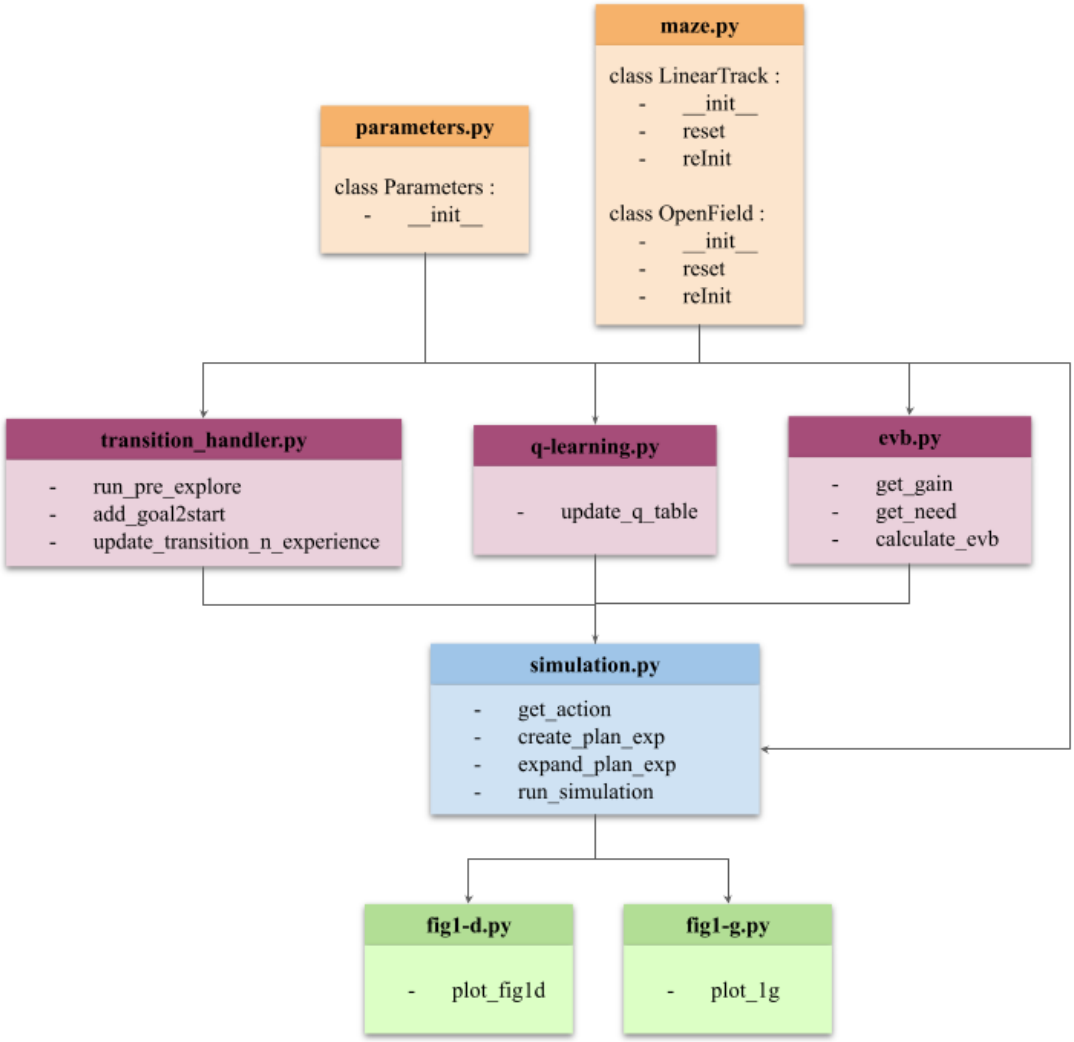


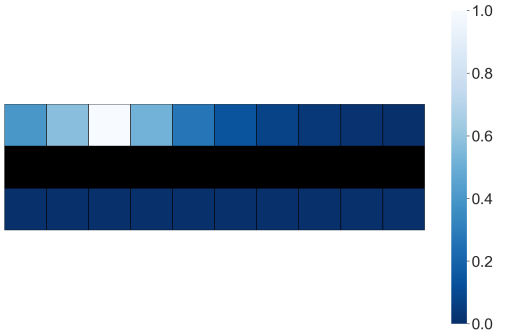
Figure 4: Représentation graphique des fichiers du code

3.4 Figures reproduites

Nous avons affiché ci-dessous les figures que nous avons reproduites jusqu'à maintenant.

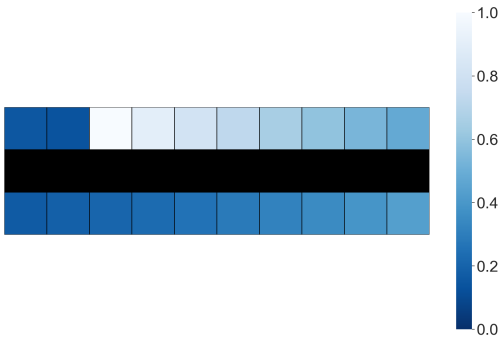
La figure 5 est la représentation graphique du *need* dans les labyrinthes *Linear Track* et *Open Field* à différents moments de la simulation. Nous avons normalisé les résultats pour mieux les visualiser. Dans ces figures, nous avons attribué aux murs la couleur noire.

Les graphiques suivants représentent le nombre de pas par épisodes nécessaire à l'agent pour atteindre la récompense en fonction du mode de remémoration des souvenirs utilisés. Nous avons produit le graphique de gauche avec notre code actuel. Le graphique de droite est une copie de la figure correspondante dans l'article dont nous essayons de reproduire les résultats.



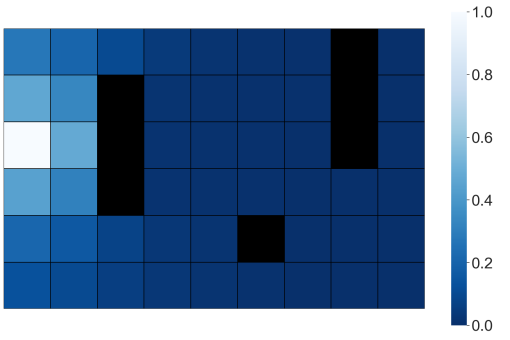
(a) Notre figure : Linear Track, représentation du need avant l'apprentissage

(b) Figure des auteurs : Linear Track, représentation du need avant l'apprentissage

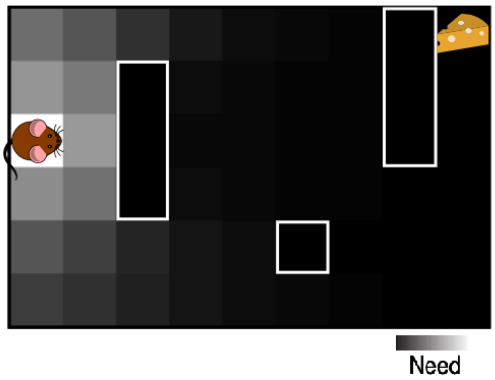


(a) Notre figure : Linear Track, représentation du need après l'apprentissage

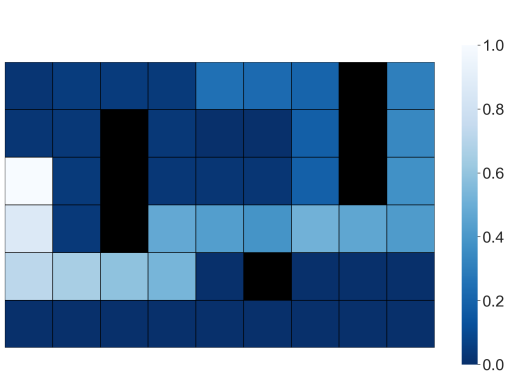
(b) Figure des auteurs : Linear Track, représentation du need après l'apprentissage



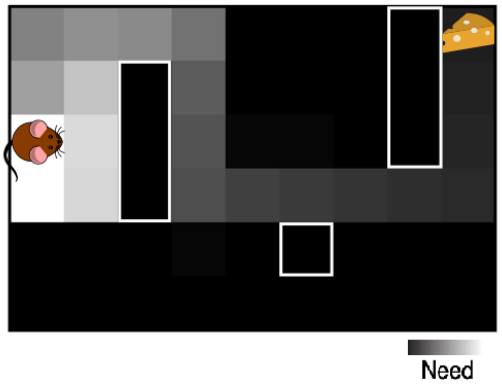
(a) Notre figure : Open Field, représentation du need avant l'apprentissage



(b) Figure des auteurs : Open Field, représentation du need avant l'apprentissage

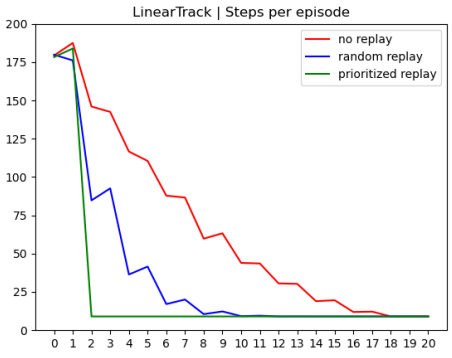


(a) Notre figure : Open Field, représentation du need après l'apprentissage

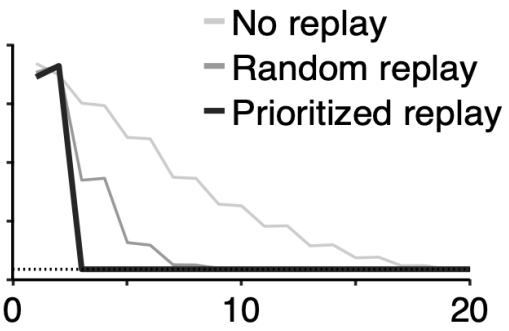


(b) Figure des auteurs : Open Field, représentation du need après l'apprentissage

Figure 5: Représentations graphiques du need

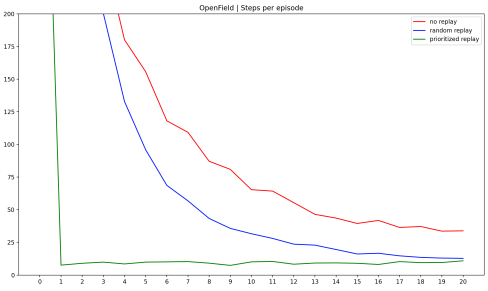


(a) Notre figure : Nombre de pas jusqu'à la récompense par épisode

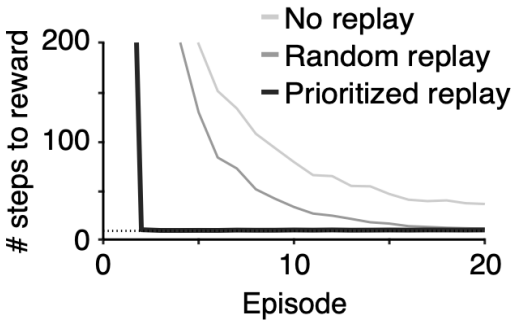


(b) Figure des auteurs : Nombre de pas jusqu'à la récompense par épisode

Figure 6: Performance d'un agent dans Linear Track



(a) Notre figure : Nombre de pas jusqu'à la récompense par épisode



(b) Figure des auteurs : Nombre de pas jusqu'à la récompense par épisode

Figure 7: Performance d'un agent dans Open Field

4 Conclusion

Pour conclure, notre objectif est de reproduire toutes les figures créées par Mattar et Daw. Nous continuons à étudier le code des auteurs. Aussi, nous découvrons au fur et à mesure des détails qui ne sont pas expliqués clairement dans l'article mais qui sont présents dans leur programme.

Actuellement nous avons des difficultés à comprendre comment l'algorithme choisit de faire un *forward replay* ou un *backward replay*. Il s'agit d'un résultat important de l'article et nous ne saisissons toujours pas où cela intervient dans le code.

En outre, la majorité des figures que nous devons encore reproduire concerne le *forward* et *backward replay*. Nous devons donc impérativement comprendre cette partie pour poursuivre notre projet.

En fonction du temps qu'il nous reste après la reproduction des figures de l'article, nous envisageons de modifier certains paramètres pour voir comment les résultats évoluent. Par exemple, nous pensons changer le mode de perception du rat : au lieu de connaître la matrice de transition, il pourrait ne percevoir que les cases qui l'entourent et donc la construire au fur et à mesure.