Rapport Projet MARL

COCOMA

Natacha Rivière, 28706745
Léa Movsessian, 28624266

Sorbonne Université

2024-2025

1 Introduction

1.1 Environnement

Nous avons choisi l'environnement Multiwalker, faisant partie des environnements SISL.

Plusieurs agents portent un paquet représenté par une barre sur la tête et doivent l'amener, en coopérant, vers la droite de l'écran.

Les agents se déplacent en appliquant une force dans leurs "jambes". L'espace des actions, continu, est donc représenté par un vecteur de 4 éléments. L'espace des observations est également continues et est représenté par un vecteur de 31 éléments.

Pour ce projet, nous avons choisi de garder le nombre d'agents par défaut, c'est-à-dire trois.

1.2 Objectifs

Notre but est de faire apprendre aux agents à transporter le paquet jusqu'à la droite de l'écran. Pour cela, étant donné que l'espace des actions est continu, nous avons choisi l'algorithme DDPG.

Notre objectif a été de tester plusieurs implémentations différentes de l'algorithme, de changer des paramètres, pour analyser les effets sur l'apprentissage des agents.

2 Implémentation

2.1 Librairies

Pour les algorithmes d'apprentissage par renforcement, nous utilisons la librairie AgileRL. La liste des paquets nécessaires, ainsi que leur version, est disponible dans le README associé au projet. L'organisation des fichiers est également disponible dans le README.

2.2 DDPG

En premier lieu, nous avons transformé le problème de multi-agents en un problème mono-agent. Nous avons donc entraîné un unique agent avec l'algorithme DDPG. Cet agent correspond à un seul walker, mais sa politique est dupliquée sur tous les walkers. En conséquence, tous les walkers font la même action à chaque pas. Comme les agents sont identiques et que le terrain est principalement plat, cette approche nous a paru pertinente.

En effet, si tous les agents font les mêmes actions, on peut supposer qu'ils avanceront de manière synchronisée si l'agent DDPG apprend correctement.

2.3 MADDPG

Nous avons ensuite travaillé avec l'algorithme MADDPG. Pour ce faire, nous avons utilisé plusieurs agents DDPG, un pour chaque walker. Nous avons testé deux approches pour le replay buffer, le mettre en commun, les agents apprennent donc tous grâce aux mêmes

échantillons de transitions; et un replay buffer par agent, où les agents sélectionnent leurs propres échantillons de transitions. D'après la documentation, utiliser un replay buffer commun permet de stabiliser l'apprentissage.

3 Résultats et Analyses

3.1 DDPG

Avec cette approche, nous obtenons généralement le toujours le même résultat, notre agent apprend à ne pas tomber, souvent en mettant un genou à terre, et reste statique. Il est possible d'observer ce comportement dans la vidéo `multiwalker_duplicate_multi`.

Cet apprentissage est problématique car si les agents restent immobiles sans tomber, la simulation continue longtemps sans changement notable. On a donc beaucoup de pas inutiles, les agents ne bougent pas, n'obtiennent pas de récompense, mais pas non plus de pénalité. On a donc un temps de calcul très long pour chaque simulation, sans obtenir de résultats intéressants.

Nous avons aussi déduit un autre problème de ce comportement, comme les agents font beaucoup de pas inutiles, qui consiste à rester sur place, ils saturent assez vite le replay buffer de transitions ce qui ne permet pas l'apprentissage. Par conséquent, les calculs sont trop long et en plus ne sont plus utiles.

Nous avons essayé de corriger ce comportement de plusieurs manières :

- Variation de la récompense pour avancer
- Enchaînement de DDPG et MADDPG

Pour le premier cas, nous avons tenter de mettre la récompense pour avancer à 100, pour compenser la pénalité de tomber, par défaut à -100. Notre hypothèse était que les agents n'essayaient pas d'avancer parce que la pénalité était trop forte ou la récompense trop basse.

Pour le second cas, nous avons essayé de fusionner les deux méthodes. En effet, comme dans l'algorithme "mono-agent" les agents apprennent à ne pas tomber et que des comportements intéressants et différents ont pu être observés avec MADDPG, nous avons essayé d'entraîner un modèle qui ne tombe pas, pour ensuite reprendre ce modèle avec l'algorithme MADDPG pour que les agents développent ces comportements sans tomber.

Dans les deux cas, le comportement était le même ou pire. Les agents n'ont donc pas réussi à avancer ou l'apprentissage était bien trop long à cause des problèmes cités précédemment.

3.2 MADDPG

Figure 1: Evolution de la récompense avec MADDPG en utilisant un replay buffer par agent.

Comme on peut l'observer sur la figure 1, avec un replay buffer par agent, on obtient une récompense très instable, soit autour de 0, soit autour de -100. C'est l'exploration plus que l'apprentissage qui permet d'obtenir des récompenses autour de zéro, c'est-à-dire des itérations où les agents arrivent à ne pas tomber et à rester plus ou moins stables.

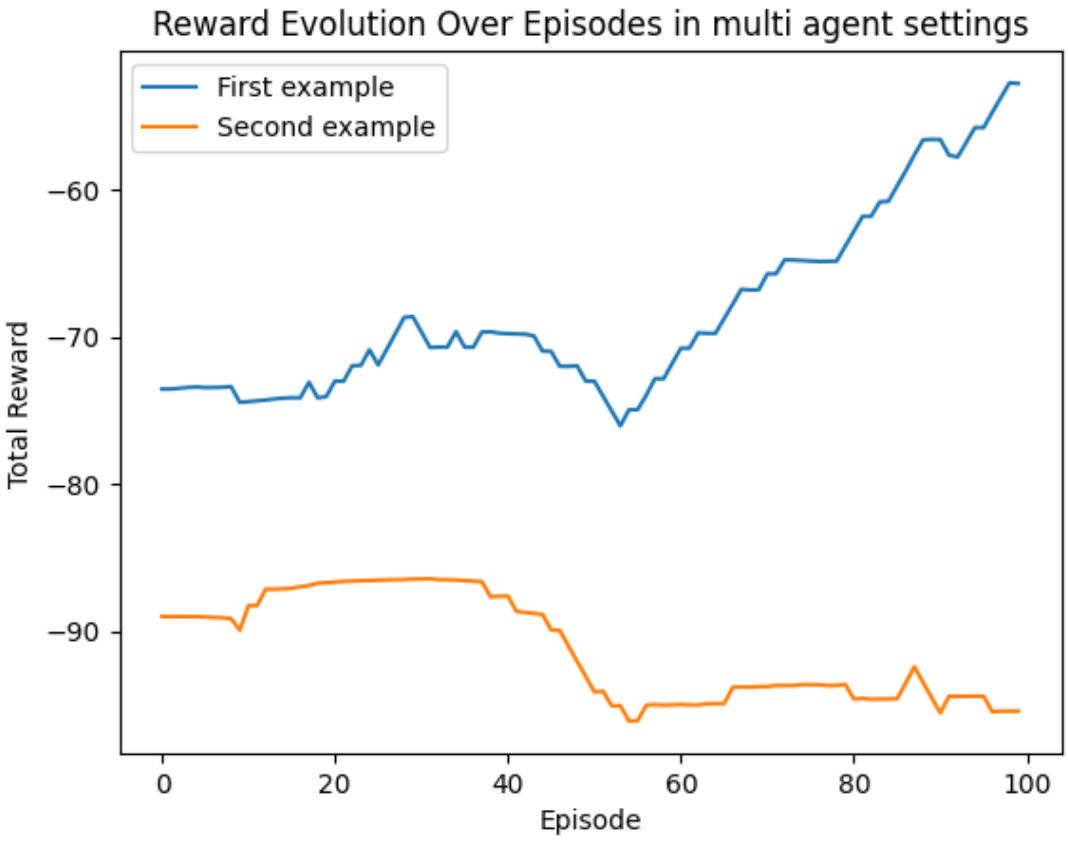

Figure 2: Evolution de la récompense avec MADDPG en utilisant un replay buffer par agent (lissé)

La figure 2 nous confirme que la méthode telle quelle n'est pas concluante. Même si on note une amélioration pour la première courbe, ce n'est pas le cas pour la seconde ni dans le cas général. Cela repose donc plus sur la chance que sur un véritable apprentissage.

Dans la vidéo `multiwalker_buffer_individuel`, on peut voir que c'est l'agent le plus à gauche qui tombe et arrête la simulation. Celui le plus à droite arrive à se stabiliser comme pour la version mono-agent. Un des agents arrivent donc à apprendre un comportement stable, mais les autres en sont incapables dans l'état actuel.

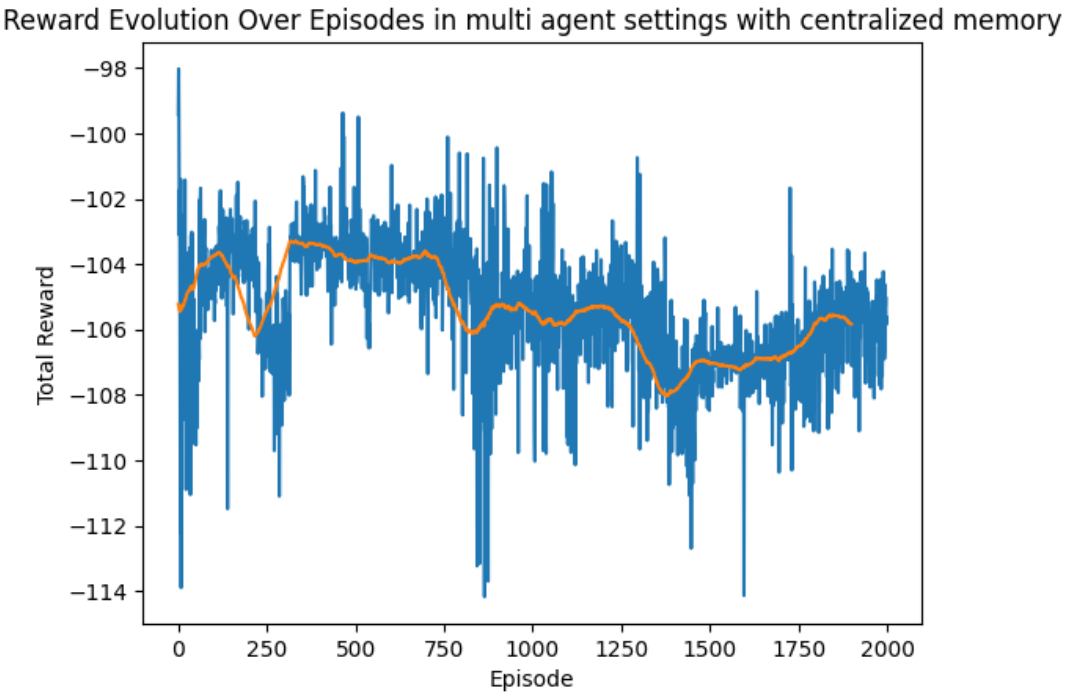

Figure 3: Evolution de la récompense avec MADDPG en utilisant un replay buffer commun à tous les agents

Comme on peut le voir sur la figure 3, dans le cas d'un seul replay buffer pour tous les agents, les agents n'apprennent pas. Ils continuent de tomber, le manque d'exploration se fait immédiatement voir.

4 Conclusion

Les essais avec DDPG et MADDPG ne sont pas concluants. Nous n'avons pas réussi à faire avancer les agents et donc à coopérer pour atteindre le but.

Néanmoins, il est possible qu'avec assez d'épisodes, on puisse observer un certain apprentissage. Il nous est malheureusement impossible de confirmer ou d'infirmer cette théorie vu que cela prend un temps trop conséquent pour faire tourner les algorithmes.

La technique mélangeant les deux méthodes restent une possibilité, étant donné que nous n'avons pas pu entièrement l'essayer à cause, encore une fois, d'un apprentissage beaucoup trop long.

Cependant, grâce à la première méthode, nous avons toutefois réussi à obtenir un état stable, où les agents ne tombent pas et donc, ne meurent pas. A défaut d'avoir appris à marcher, ils ont appris à survivre, ce qui reste une réussite en soit.