Rapport projet COCOMA

Étudiants :
Baris KAFTANCIOGLU : 28711733
Paul-Tiberiu IORDACHE : 28706827

Encadré par :
Vincent CORRUBLE

10 novembre 2024

Table des matières

1 Introduction

Le but de ce rapport est d'implémenter un algorithme d'apprentissage par renforcement profond dans un environnement multi-agent. Nous tenons à préciser que l'algorithme choisi est le DQN de la librairie SB3 (Stable Baselines 3) et que l'environnement multi-agent que nous avons sélectionné est KAZ (Knights Archers Zombies) de la librairie PettingZoo.

Dans notre étude, nous optons pour l'algorithme DQN en raison de sa capacité à traiter efficacement les environnements de renforcement multi-agents, tout en bénéficiant de la généralisation des expériences à travers un réseau de neurones profond. Nous allons comparer nos résultats avec ceux obtenus par l'algorithme PPO présenté dans l'article Revisiting Parameter Sharing in Multi-Agent Deep Reinforcement Learning de J. K. Terry et al.. Il est également pertinent de noter que DQN et PPO partagent certaines équivalences, notamment dans leur utilisation de techniques de mise à jour de politique et d'optimisation, bien qu'ils diffèrent fondamentalement dans leur approche.

Avant de commencer l'étude, nous tenons à préciser que le code est disponible sur notre dépôt GitHub : https://github.com/PaulTiberiu/MultiAgentDeepRL.

L'environnement KAZ est un environnement où les agents (Knights et Archers) doivent collaborer pour éliminer les zombies. Chaque fois qu'un agent élimine un zombie, soit avec une flèche tirée par l'archer, soit avec un coup de sabre donné par le knight, les agents reçoivent une récompense de 1. Si tous les agents sont tués par un zombie ou si les zombies arrivent en bas de l'environnement, la simulation est terminé.

2 Première approche

En ce qui concerne notre première approche, au lieu de commencer par tester une stratégie aléatoire pour nos agents, nous avons choisi d'utiliser une approche simple d'apprentissage par renforcement multi-agents, c'est-à-dire appliquer un algorithme d'apprentissage à chaque agent de l'environnement. Ainsi, dans cette première approche, nous avons utilisé 2 archers et 2 knights, chacun appliquant l'algorithme DQN. De plus, un total de 4 zombies a été utilisé lors de ces premières expériences.

Nous initialisons 4 agents DQN dans l'environnement KAZ et implémentons une fonction d'entraînement sur 1000 timesteps et 1000 épisodes. À chaque étape, les agents s'entraînent dans l'environnement, puis chaque agent effectue une prédiction pour sa prochaine action. Nous visualisons également la performance des agents pendant chaque épisode.

Au début, nous avons utilisé les hyperparamètres proposés par la librairie SB3, que l'on peut trouver dans le code DQN sur le GitHub de SB3 : `https://github.com/DLR-RM/stable-baselines3/blob/master/stable_baselines3/dqn/dqn.py` :

- `learning_rate` : `Union[float, Schedule]` = 1e-4,
- `buffer_size` : `int` = 1000000, # 1e6
- `learning_starts` : `int` = 100,
- `batch_size` : `int` = 32,
- `tau` : `float` = 1.0,
- `gamma` : `float` = 0.99,

— `train_freq` : `Union[int, Tuple[int, str]]` = 4,
— `gradient_steps` : `int` = 1,
— `target_update_interval` : `int` = 10000,
— `exploration_fraction` : `float` = 0.1,
— `exploration_initial_eps` : `float` = 1.0,
— `exploration_final_eps` : `float` = 0.05,
— `max_grad_norm` : `float` = 10,
— `stats_window_size` : `int` = 100,
— `net_arch` : `[int, int]` = [64, 64]

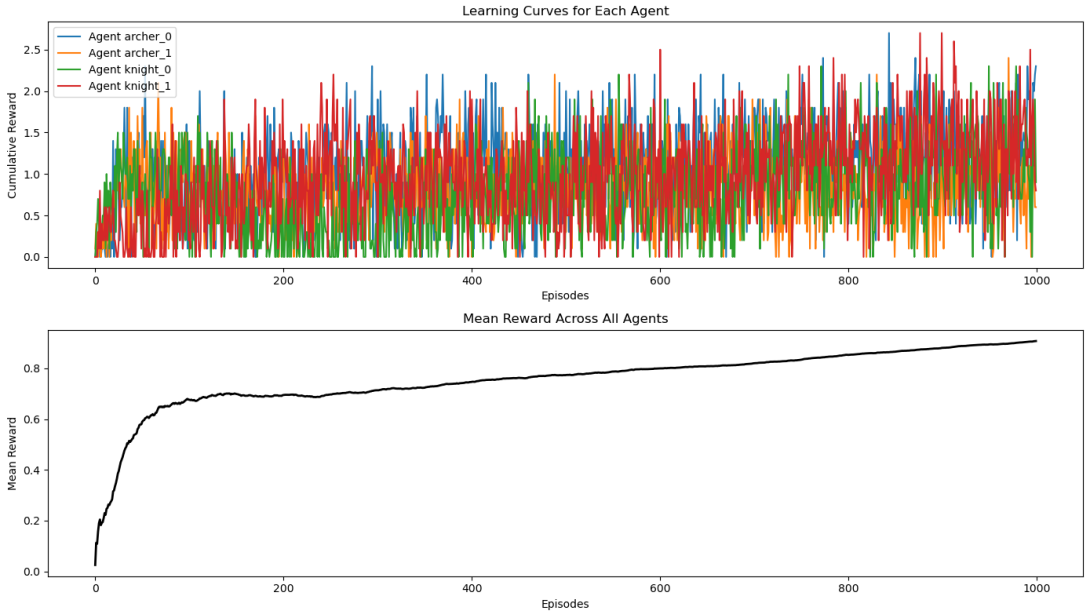

FIGURE 1 – Courbes d'apprentissage en SB3 dans l'environnement KAZ avec les hyperparamètres proposés par la librairie SB3.

Comme nous pouvons le voir dans la figure 1, après 1000 épisodes, notre récompense moyenne sur les 4 agents atteint une valeur de 0,8. Cela indique que l'apprentissage a réussi, bien que ce ne soit pas encore parfait, car nous nous attendions à une valeur moyenne de 1 comme récompense, ce qui signifierait que tous les zombies sont éliminés.

Cela signifie que les hyperparamètres proposés par la librairie SB3 ne sont pas suffisamment adaptés au cas d'un environnement multi-agents, plus précisément celui de KAZ. Donc, pour obtenir de meilleurs résultats, nous avons choisi de tuner nos hyperparamètres en utilisant une librairie dédiée à cet effet : Optuna. Les plages d'hyperparamètres sur lesquelles nous lançons l'optimisation se trouvent dans le tableau 1.

Hyperparameter	**Range**
Gamma (Discount Factor)	0.9, 0.95, 0.98, 0.99, 0.995, 0.999, 0.9999
Learning Rate	[1e-5, 1]
Batch Size	16, 32, 64, 100, 128, 256, 512
Buffer Size	10^4, 5×10^4, 10^5, 10^6
Exploration Final Epsilon	[0, 0.2]
Exploration Fraction	[0, 0.5]
Target Update Interval	1, 1000, 5000, 10000, 15000, 20000
Learning Starts	0, 1000, 5000, 10000, 20000
Train Frequency	1, 4, 8, 16, 128, 256, 1000
Subsample Steps	1, 2, 4, 8
Gradient Steps	max(train_freq/subsample_steps, 1)
Network Architecture Type	tiny, small, medium
Network Architecture	{tiny : [64], small : [64, 64], medium : [256, 256]}

TABLE 1 – Plages d'hyperparamètres pour l'étude d'optimisation

Après avoir lancé Optuna, nous avons trouvé le jeu d'hyperparamètres suivant :

— `gamma` : 0.99
— `learning_rate` : 0.0001774658772353578
— `batch_size` : 32
— `buffer_size` : 10000
— `exploration_final_eps` : 0.08646173611211427
— `exploration_fraction` : 0.32951412388518647
— `target_update_interval` : 1000
— `learning_starts` : 0
— `train_freq` : 128
— `subsample_steps` : 4
— `net_arch` : [64]

Comme le montre la figure 2, après 1000 épisodes, notre récompense moyenne semble converger vers la valeur optimale de 1. De plus, en comparant avec la figure 1, nous observons que la convergence se produit plus rapidement, autour du 220^e épisode. Cependant, nous remarquons également qu'à partir du 600^e épisode, la courbe commence à descendre légèrement, tout en restant très proche de la valeur de 1. Ce phénomène nous paraît normal, car après avoir convergé et bien appris, l'algorithme DQN a tendance à privilégier l'exploration, et cette légère baisse peut être expliquée par un équilibre naturel entre exploration et exploitation.

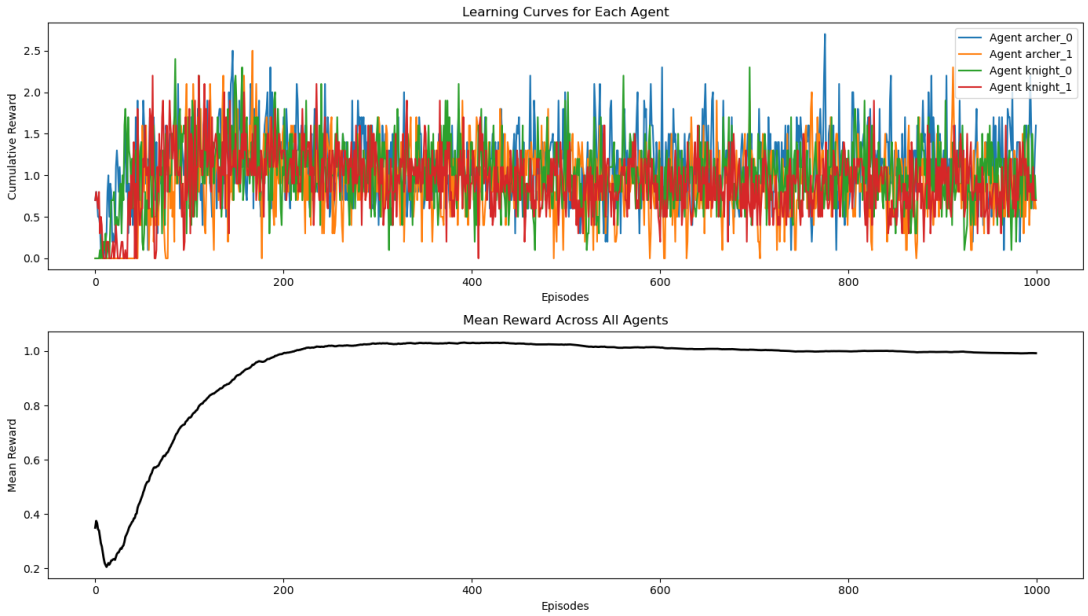

FIGURE 2 – Courbes d'apprentissage en SB3 dans l'environnement KAZ avec les hyperparamètres proposés par Optuna.

3 Deuxième approche

Pour affiner l'approche simple de l'apprentissage par renforcement multi-agents, nous avons décidé de pénaliser les agents lorsqu'ils effectuent des actions inappropriées. De plus, nous avons réfléchi à plusieurs méthodes, notamment l'ajout de communication entre les agents. Cependant, dans le cas de l'environnement KAZ, il nous semble que cette communication ne serait pas très utile, car une connaissance globale de l'environnement peut être obtenue en consultant les observations fournies par env.last(). La seule situation où la communication pourrait s'avérer utile serait que chaque agent puisse signaler le zombie qu'il choisit d'attaquer (donc sa propre action) en ajoutant un champ dans les observables. Par exemple, cela permettrait de donner la priorité aux archers pour cibler les zombies les plus éloignés, tandis que les chevaliers s'occuperaient de ceux qui sont plus proches.

Dans notre méthode, nous appliquons une pénalité de -1 à la récompense des agents, dans la fonction step de l'environnement, lorsque les zombies se rapprochent dangereusement (au-delà d'un certain seuil) de la partie inférieure de l'environnement (cas où les zombies gagnent) et lorsqu'ils menacent de tuer un de nos agents (également à un certain seuil). Pour évaluer si cette approche améliore les résultats, nous avons choisi de tester d'abord avec les mêmes hyperparamètres que pour les figures 1 et 2, avant d'effectuer un ajustement des hyperparamètres avec Optuna.

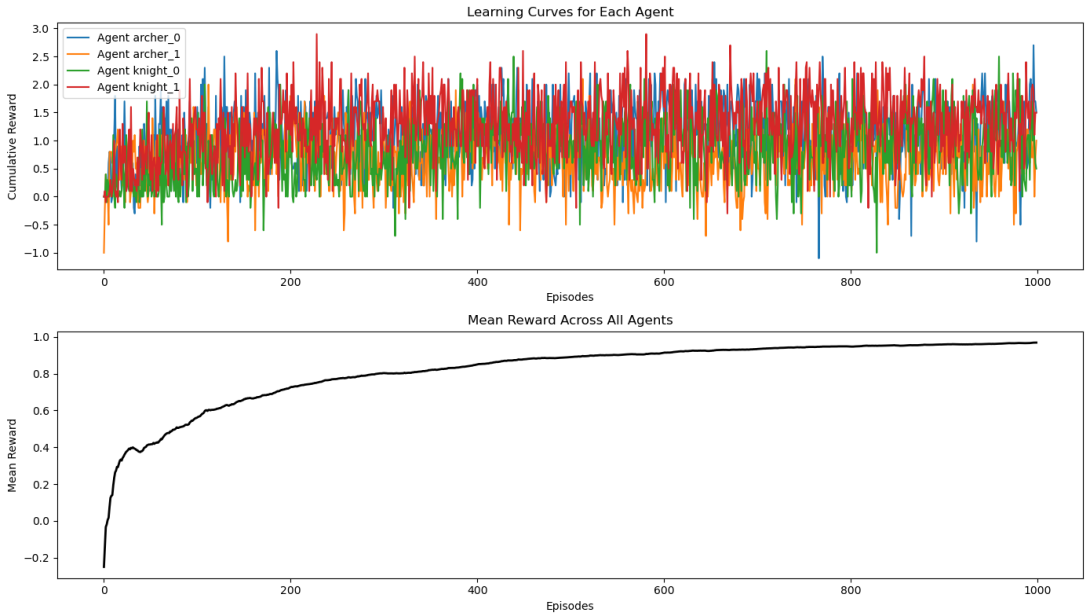

FIGURE 3 – Courbes d'apprentissage avec l'approche affiné en SB3 dans l'environnement KAZ avec les hyperparamètres proposés par la librairie SB3.

Par rapport à la figure 1, nous constatons que, même si nous observons quelques valeurs négatives au début (ce qui est normal en phase initiale d'apprentissage), nous atteignons, vers l'épisode 1000, une convergence très proche de 1, supérieure à celle obtenue dans la figure 1, où nous avions atteint 0,8. De plus, nous remarquons qu'avec cette nouvelle version, l'algorithme converge globalement plus rapidement.

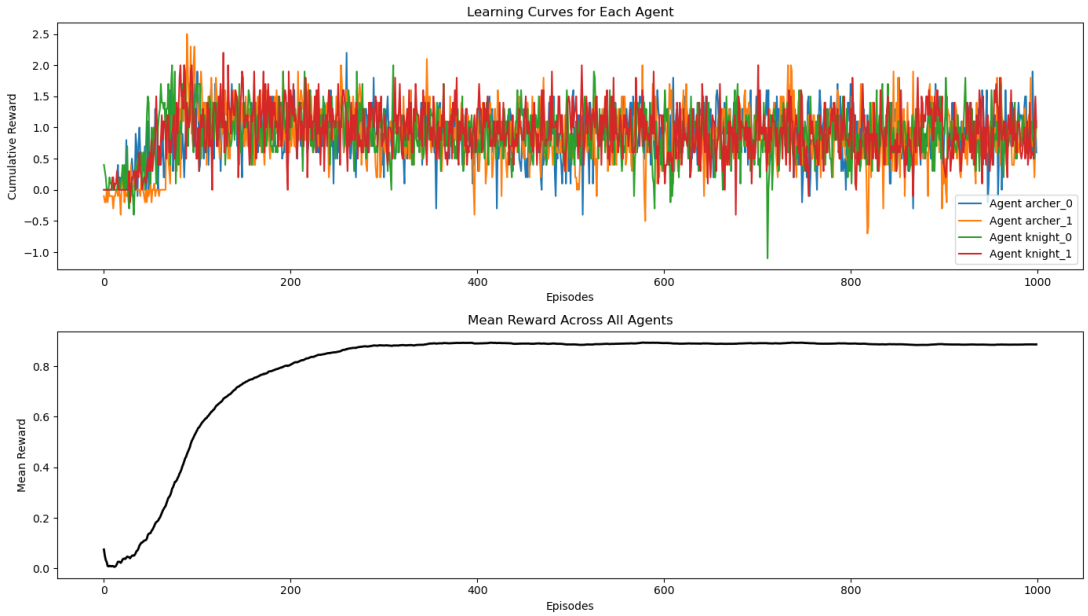

FIGURE 4 – Courbes d'apprentissage avec l'approche affiné en SB3 dans l'environnement KAZ avec les mêmes hyperparamètres que la figure 2.

Par rapport à la figure 2, nous observons qu'en utilisant les mêmes hyperparamètres, la convergence est atteinte vers le même épisode. En revanche, dans ce cas, la convergence

n'est pas parfaite, contrairement à la figure 2 où elle atteignait une valeur de 1 ; ici, elle oscille plutôt autour de 0,9.

Maintenant, nous avons lancé une optimisation des hyperparamètres en utilisant les mêmes valeurs que celles du tableau 1. Les résultats sont les suivants :

— `gamma` : 0.95
— `learning_rate` : 0.0005001046217977603
— `batch_size` : 16
— `buffer_size` : 100000
— `exploration_final_eps` : 0.09536343411663083
— `exploration_fraction` : 0.1821964330021213
— `target_update_interval` : 1
— `learning_starts` : 0
— `train_freq` : 128
— `net_arch` : [64]

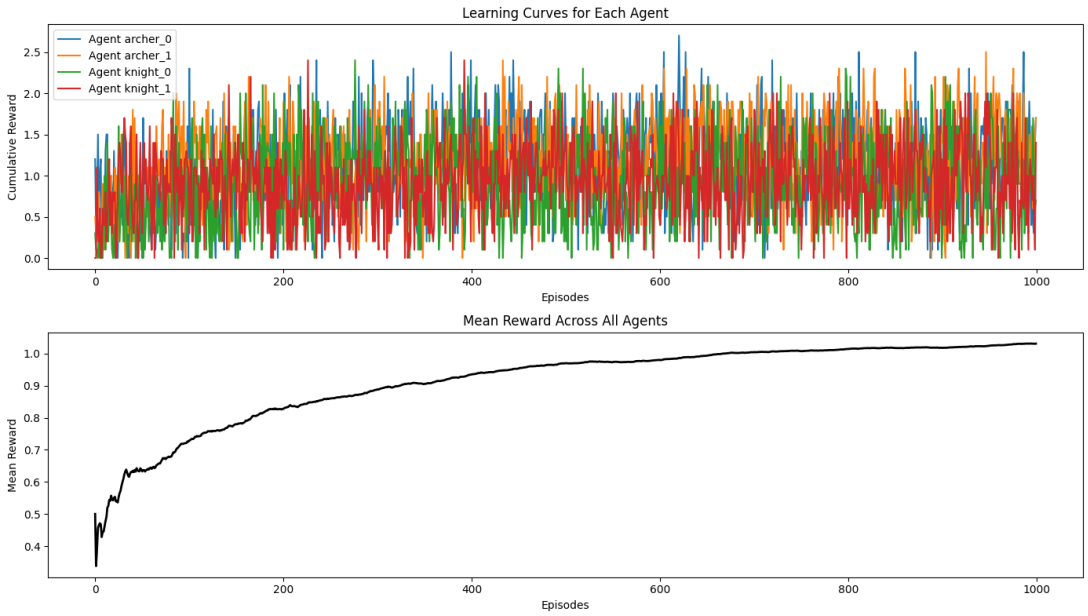

FIGURE 5 – Courbes d'apprentissage avec l'approche affiné en SB3 dans l'environnement KAZ avec les hyperparamètres donnés par Optuna.

Dans le cas de la figure 5, nous pouvons constater une convergence vers la valeur maximale : 1. Cependant, dans ce cas, nous avons eu besoin de 700 épisodes pour atteindre cette convergence, ce qui est moins rapide avec la convergence de la figure 2 utilisant l'approche simple. Toutefois, il y a quelques différences d'implémentation entre notre code et celui de la librairie PettingZoo que nous allons détailler par la suite.

Nous tenons à préciser que nos meilleures valeurs pour les seuils sont : 0,98 (max 1) pour la position y des zombies et 0,02 pour la distance entre le zombie et l'agent courant. Comme le montrent les figures 3 et 4, les résultats ne convergent pas vers la valeur maximale, donc 1. Cependant, en examinant les courbes individuelles de chaque agent, nous

observons que même vers les derniers épisodes de l'apprentissage, certaines valeurs restent négatives, ce qui est inattendu, surtout après une optimisation des hyperparamètres.

Ce phénomène est dû au fait que, dans l'environnement KAZ de PettingZoo, les collisions entre agents et zombies sont gérées comme des collisions de sprites (spritecollider). Notre seuil de 0,02 ne représente donc pas forcément une collision réelle, mais simplement une valeur que nous avons fixée. De même, un problème similaire survient pour les sorties de l'environnement par les zombies : dans le code KAZ, la sortie d'un zombie au bas de l'écran est indiquée par zombie.rect.y > const.SCREEN_HEIGHT - const.ZOMBIE_Y_SPEED. Cependant, n'ayant pas accès à ces données directement et ne pouvant utiliser que les observations normalisées entre 0 et 1, nous avons fixé cette valeur à 0,98. (un cas similaire se trouve dans la vidéo trained_agents_Reward_punishment, où le chevalier se situe très près, sous le seuil, d'un zombie avant de le tuer)

Ainsi, des cas peuvent se produire où un zombie s'approche à une distance inférieure à 0,02 de l'agent et le spridecollider n'est pas encore produit, mais l'agent le tue ensuite. Dans ce cas, la récompense reste nulle (-1 parce que le zombie est proche de nous, +1 parce que nous avons tué le zombie), ce qui ne reflète pas correctement le scénario. Le même problème survient lorsque les zombies sortent de l'environnement : nous pénalisons les agents lorsqu'un zombie dépasse la position 0,98 en y, mais il peut arriver que certains agents tuent le zombie avant que la condition de sortie dans l'environnement KAZ soit remplie.

4 Comparaisons et conclusions

Comme le montre la figure 6, les résultats obtenus dans l'article mentionné, en utilisant différentes méthodes, se situent également autour de 0,8 (avec quelques variations). Ainsi, nos résultats semblent être en accord avec ceux d'autres sources, ce qui valide notre approche.

Agent indication method	Avg. Reward
Inversion	0.87 ± 0.29
Geometric	0.82 ± 0.3
Inversion	0.67 ± 0.22
Inversion with Replacement	0.66 ± 0.18
Inversion	0.65 ± 0.26
Inversion with Replacement	0.58 ± 0.11
Inversion with Replacement	0.53 ± 0.06
Geometric	0.53 ± 0.08
Geometric	0.51 ± 0.07
Inversion with Replacement	0.5 ± 0.08

Figure 6 – Résultats de l'article Revisiting Parameter Sharing in Multi-Agent Deep Reinforcement Learning de J. K. Terry et al. sur l'environnement KAZ, utilisant l'algorithme PPO pour différentes méthodes d'indication des agents.

Pour conclure, nous observons que notre deuxième méthode converge mieux lorsque les hyperparamètres ne sont pas bien réglés. En revanche, une fois les hyperparamètres optimisés, les deux méthodes convergent vers la valeur maximale. Ainsi, nous pouvons affirmer que la deuxième approche est moins sensible aux choix des hyperparamètres. Cependant, elle ne converge pas toujours exactement à 1, mais à des valeurs très proches,

en raison de différences dans la manière dont les collisions sont traitées dans le code source de PettingZoo et dans notre code.

5 Visualisation de nos agents

Nous précisons que des vidéos de nos agents entraînés avec les deux approches sont disponibles sur notre GitHub : https://github.com/PaulTiberiu/MultiAgentDeepRL, sous les noms trained_agent_dumb_approach.mp4 et trained_agents_Reward_punishment.mp4. Ces résultats ont été obtenus avec les hyperparamètres optimisés via Optuna (correspondant aux vidéos des figures 2 et 5). En outre, nous avons également ajouté une vidéo présentant des agents non entraînés. Comme nous pouvons le constater, les agents entraînés (en particulier l'un des deux archers) ont bien appris à viser les zombies. Cependant, ils ne sont pas encore parfaits et de meilleures optimisations pourraient être réalisées pour améliorer leur apprentissage.

6 Références

[1] J. K. Terry et al., Revisiting Parameter Sharing in Multi-Agent Deep Reinforcement Learning, https://arxiv.org/pdf/2005.13625