ECOLE ROYALE MILITAIRE

172ième Promotion POL

Cpt Maxime Sévrin

Année académique 2021 – 2022

2ième Master

La communication entre les agents dans l'apprentissage par renforcement

Par Sous Lieutenant candidat officier de carrière

Yemen RHOUMA

Mémoire de fin d'études du département CISS
présenté en vue de l'obtention du grade académique
de master en sciences de l'ingénieur
sous la direction Capitaine Commandant Koen BOECKX, ir
Bruxelles, 2022

ECOLE ROYALE MILITAIRE
Département CISS

Version 1.0
25/4/2022

La communication entre les agents dans l'apprentissage par renforcement

Yemen RHOUMA

RMA
Royal Military Academy

Avant-propos

Table des matières

Liste des graphiques et figures

Liste des tableaux

1. Introduction

1.1. Contexte

Cette thèse fait partie d'un grand projet "Intelligent Recognition Information System (IRIS)" qui est pris en charge par le département CISS de l'école royale militaire. Le projet a débuté en janvier 2019 et a pour de développer des outils afin d'aider l'équipage des véhicules blindés à exécuter leurs tâches (engager ou non un ennemi potentiel, faire de la reconnaissance ...). Le projet se compose de trois grandes étapes :

- La détection et classification d'objets au sol grâce à des capteurs situés à l'avant du véhicule. Cela conduit à la création d'une grande collection de données sur les objets rencontrés sur le terrain et leurs liens avec le véhicule militaire (position, distance, etc.). à citer SAHARA Semi-Automatic Help for Aerial Region Analysis
- La détection des différentes menaces potentielles afin de créer une carte de ce qui est connu sur le terrain.
- La création d'une stratégie d'attaque à partir de la carte de situation afin de traiter les différentes menaces.

Dans le cadre du troisième point présenté ci-dessus, une partie a été faite par le Cdt Koen BOECKX, ir et qui sera reprise et étudiée en détail. Son travail consiste en :

- Premièrement, le développement d'un modèle algorithmique représentant le terrain. Les acteurs tels que les forces amies et ennemies peuvent interagir avec l'environnement, par exemple en tirant, en se déplaçant, en visant... . L'environnement contient des obstacles qui, par leur présence, peuvent restreindre la visibilité et la mobilité des agents.
- Deuxièmement, voir s'il est possible de créer une stratégie d'attaque basée sur des algorithmes de multi-agents basées sur l'intelligence artificielle et des théorie comme "l'apprentissage approfondi (Deep Learning)" et "l'apprentissage par Renfoncement (Reinforcement Learning)" .

1.2. Analyse de la littérature

1.2.1. L'Apprentissage par Renforcement

L'idée derrière l'apprentissage par renforcement est d'apprendre en interagissant avec l'environnement. Ce type d'apprentissage est celui que les humains expérimentent dès la naissance. Un enfant après la naissance n'a pas d'enseignant explicite. Il dispose de ses différents sens qui lui permettent d'obtenir des informations sur son environnement.

Le domaine de "l'apprentissage automatique" peut être divisé en deux grandes catégories, dont il faut comprendre la différence. Ces deux catégories sont "l'Apprentissage Supervisé (Supervised Learning)" et "l'apprentissage par renforcement".

L'apprentissage supervisé consiste à construire une fonction qui associe une certaine entrée à une certaine sortie sur la base d'exemples. Ces exemples sont étiquetés en fonction de la sortie souhaitée. L'objectif est d'interpréter correctement les nouveaux exemples.

Contrairement au dernier type d'apprentissage, dans l'apprentissage par renforcement, l'agent s'entraîne par l'expérience, non pas en lui donnant une série d'exemples mais en le mettant dans l'environnement. Plus de détails vont suivre dans le paragraphe suivante.

Les éléments de l'apprentissage par renforcement [6]

Il existe deux entités majeures dans l'apprentissage par le renforcement (figure 1.2). Il s'agit de l'**environnement** et de l'**agent**. L'agent est une entité physique ou virtuelle (en simulation) qui est capable d'agir dans un environnement et de le percevoir (de manière limitée). Il est animé par un ensemble de tendances sous forme d'objectifs (fonction d'optimisation) lui permettant de rechercher son but, voire sa survie.

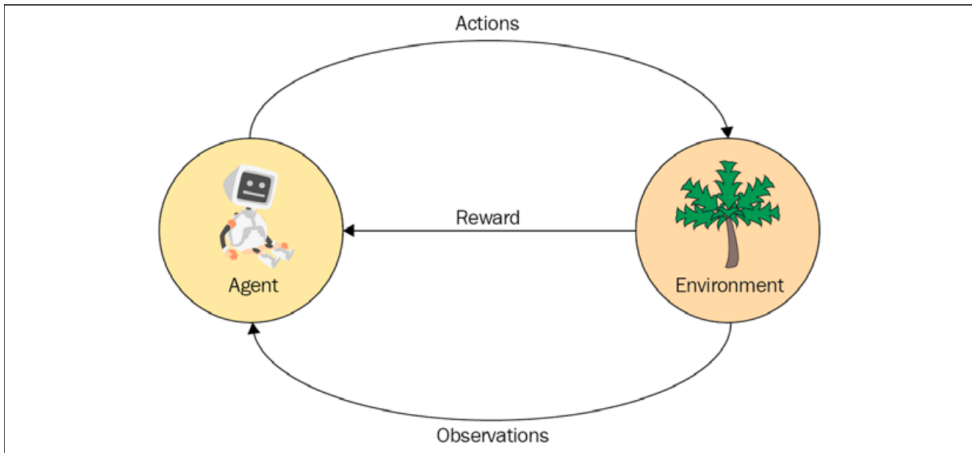

Fig. 1.1. Les entités de l'apprentissage par renforcement [6]

L'agent peut effectuer des **actions** en fonction de l'état actuel qui est défini par l'état de l'environnement et l'état propre de l'agent. Cela conduit à un nouvel état, chaque action entreprise est liée à une récompense ou un coût immédiat (**Reward**). En conclusion, l'objectif de l'agent est d'apprendre quelle action choisir dans chaque état (**observation**) afin d'atteindre son objectif final.

Processus de décision Markovien

Avant d'approfondir la théorie de l'apprentissage par renforcement, il est d'abord nécessaire de définir quelques concepts. Considérons un système dynamique qui est uniquement observable. Les différentes observations forment l'"espace d'état". Pour appeler ce processus un processus de Markov, la propriété de Markov doit être satisfaite. Cette propriété est la suivante : l'observation future d'un système $S(t+1)$ ne dépend que de son état actuel $S(t)$. En bref, seul l'état actuel modélise le système et non toutes les actions précédentes. En mathématiques, un processus de Markov est une séquence de variables aléatoires $(x_0, x_1, x_2..x_n; n >= 0)$ qui satisfont à la condition suivante 1.1 :

$$P(X_{n+1} = x|X_0, X_1, X_2, ..., X_n) = P(X_{n+1}|X_n) \tag{1.1}$$

Cette notion de processus de markov sera étendue par l'introduction de la notion de récompense. La récompense peut être une valeur positive ou négative et peut être grande ou petite. Elle sera ajoutée lors de la transition d'un état à un autre. En outre, la notion d'action est ajoutée au modèle pour modéliser pleinement le problème de l'apprentissage par renforcement.

En conclusion, un processus de décision de Markov est constitué de :

- un ensemble d'états (**States**) , avec un état initial s_0
- un ensemble d'**actions** possibles pour chaque état
- un modèle de transition $P(s'|s, a)$

- une fonction qui détermine la récompense lors du passage d'un état à un autre $R(s)$

La question suivante à se poser est de savoir comment trouver la solution au problème. La solution doit spécifier ce que l'agent est censé faire dans chaque état pour atteindre son objectif. Une telle solution est définie comme la politique de l'agent notée $\pi(s)$.

L'objectif de l'agent est de déterminer la meilleure politique qui conduit à un gain total maximal 1.2 :

$$G_t = R_0 + R_1 + ... + R_n = \sum_{t=0}^{n} R_t \tag{1.2}$$

Dans les processus de décision markoviens qui n'ont pas de but final, la somme des récompenses n'a pas de valeur finie. Pour surmonter ce problème, on introduit le facteur de dévaluation γ. Ce facteur est une valeur comprise entre 0 et 1 afin d'avoir un gain total qui converge. Cela implique que plus les récompenses s'éloignent, moins elles sont importantes. le gain final qui sera maximisé aura l'expression suivante 1.3 :

$$G_t = R_0 + \gamma \cdot R_1 + ... + \gamma^n \cdot R_n = \sum_{t=0}^{n} \gamma^t \cdot R_t \tag{1.3}$$

La valeur de l'état, la valeur de l'action, l'équation de Bellman

La valeur de l'état V(s)
La grande idée derrière l'apprentissage par renforcement est bâtie sur la valeur de l'état V et comment l'approximer à l'aide de l'équation de Bellman. la valeur d'un état s peut être écrite sous cette forme :

$$V(s) = E[\sum_{t=0}^{\infty} r_t * \gamma^t]$$

La valeur de l'action Q(s,a)

La valeur de l'action (Q value) est définie comme la récompense immédiate que l'agent a eu dans un état s plus la récompense qu'il va avoir à long terme jusqu'à avoir atteint son but 1.4.

$$Q(s,a) = E_{s'\ s}[r(s,a) + \gamma \cdot V(s')] \tag{1.4}$$

ceci devient 1.5 :

$$Q(s,a) = r(s,a) + \gamma \cdot max_{a' \in A} Q(s',a') \tag{1.5}$$

Le lien entre V(s) et Q(s,a) est simplet directe 1.6 :

$$V(s) = \max_{a \in A} Q(s,a) \tag{1.6}$$

Les types de l'apprentissage par renforcement

1.2.2. Q Learning [9]

Dans cette section, une technique d'apprentissage par renforcement sera expliquée. Elle permet d'apprendre une stratégie qui indique quelle action entreprendre dans chaque état du système. L'idée est d'apprendre la fonction susmentionnée Q(s,a) qui représente le gain potentiel.
Pour ce faire, on crée un tableau qui détermine le gain pour chaque état du système et pour chaque action possible. Le tableau sera initialisé au début de la simulation en prenant un tableau rempli de zéros ou en prenant des valeurs aléatoires.

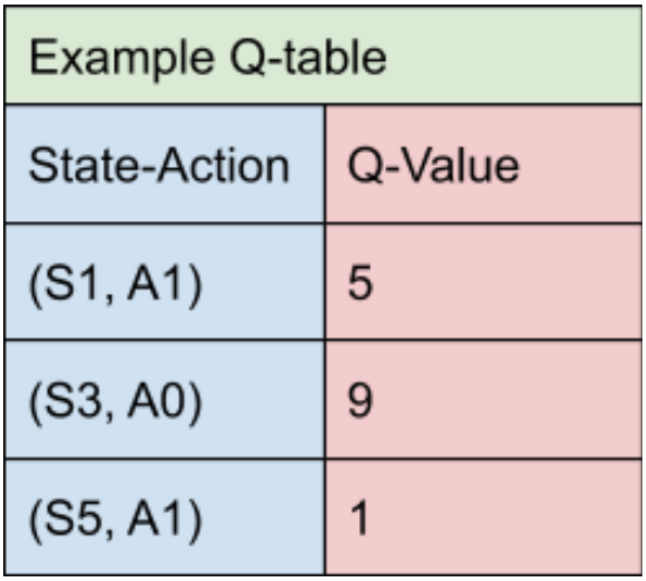

Fig. 1.2. Q table[6]

Afin de créer la politique que l'agent va suivre pour atteindre son objectif, l'agent va soit exploiter la table, soit explorer l'environnement.l'agent interagit avec l'environnement de deux manières. Premièrement, l'agent peut utiliser le tableau comme référence pour voir toutes les actions possibles dans l'état dans lequel il se trouve, il choisira donc l'action qui correspond à la valeur maximale de Q (la future récompense) . Cette façon de choisir l'action revient à l'exploitation de la table "Q_table".

La deuxième méthode consiste à agir de manière aléatoire. Au lieu de prendre une action en fonction de la récompense future, l'action sera prise au hasard. On dit alors que l'agent explore l'environnement. Ceci est très important, car cela permet à l'agent de découvrir de nouveaux états et par conséquent d'explorer d'autres chemins afin d'atteindre le but recherché.

Un bon équilibre entre exploration et exploitation est nécessaire. On peut y parvenir en choisissant une valeur de référence ϵ. Cette valeur détermine la durée d'exploration et la durée d'exploitation. Un exemple est présenté ci-dessous.

- Choisir une valeur ϵ
- choisir une valeur r aléatoire entre 0 et 1
- si r < ϵ alors **EXPLORATION**
- sinon **EXPLOITATION**

La valeur ne restera pas inchangée pendant le processus d'apprentissage. De préférence, cette valeur commence à une valeur très élevée et cette valeur est incrémentée au fil du temps. Cela se traduit par une exploitation complète au début du processus d'apprentissage et, plus tard, par une exploitation complète de la politique créée à l'aide du tableau. Cette variante d'epsilon est appelée **Epsilon Greedy** et peut prendre différentes formes (exponentielle, linéaire ...) . Un exemple est présenté dans la figure ci-dessous 1.3.

Pendant le processus d'apprentissage, le tableau contenant la politique est mise à jour à chaque fois sur la base d'une équation similaire à l'équation 1.5.

$$Q(s,a) = Q(s,a) + lr \cdot (reward + \gamma \cdot \max_{a \in A} Q(s,a) - Q(s,a)) \tag{1.7}$$

Dans l'équation ci-dessus, la valeur de Q est ajustée en fonction de la différence entre la nouvelle et

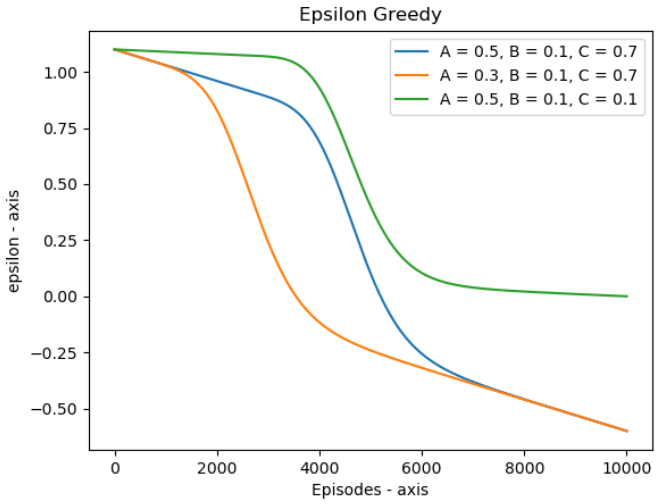

Fig. 1.3. Epsilon Greedy

l'ancienne valeur de q. Une variable de cette équation nécessite une explication.

Le taux d'apprentissage "Learning Rate" (lr or α) il représente la quantité de la nouvelle valeur à accepter.

1.2.3. Deep Q Learning [4]

Dans la section précédente, une explication du q-learning a été donnée. Il existe deux types d'environnements : un environnement avec des états et actions infinis, qui ne pose aucun problème lors de la création de la table. Le deuxième type d'environnement a un domaine d'état infini. Cela implique un échantillonnage dans le domaine d'état afin de créer la table.

Le sur-échantillonnage implique un processus d'apprentissage relativement lent. Cela nécessite une mémoire supplémentaire pour stocker et mettre à jour les états ainsi que les actions disponibles et aussi les valeurs Q. Le sous-échantillonnage peut faire que l'agent n'apprenne pas correctement.

Une solution à ce problème est d'approximer ces valeurs Q avec un modèle d'apprentissage automatique tel qu'un réseau neuronal [1]. La figure 1.4 ci-dessous représente le passage d'un tableau vers un réseau neuronal.

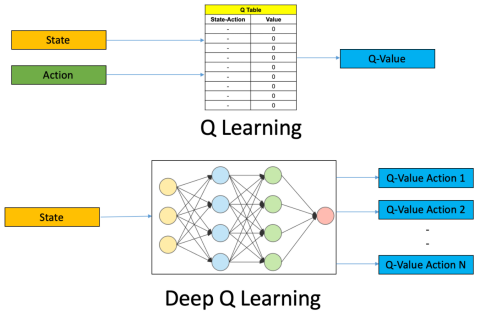

Fig. 1.4. Le passage d'un tableau de valeurs à un réseau neuronal [1]

L'utilisation d'un exemple est nécessaire pour comprendre le fonctionnement de cet algorithme. Pour ce faire, l'exemple de CartPole sera utilisé. CartPole est un environnement qui fait partie de l'interface Openai Gym. il représente un bâton tenu verticalement sur une base mobile. la position du bâton est

initialisée au centre et à un angle par rapport à la verticale égal à 0°. Le but est de maintenir ce bâton vertical en contrôlant uniquement la base mobile en la déplaçant vers la gauche et vers la droite à l'angle de 0°.

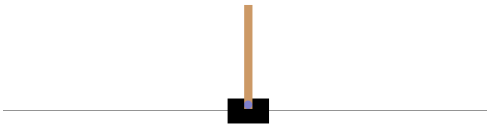

Fig. 1.5. Illustration de l'environnement CartPole

Deux entités majeures sont nécessaires pour résoudre ce problème.

- L'ensemble des états est infini pour cet environnement. L'état est présenté par 4 variables : La position de la base et sa vitesse, l'angle que forme le bâton avec l'axe vertical et la vitesse angulaire.
- Les seules actions possible sont la mouvement de la base à droite ou à gauche.

Le réseau neuronal remplaçant le tableau des valeurs de q aura la forme suivante (figure 1.6) [10]

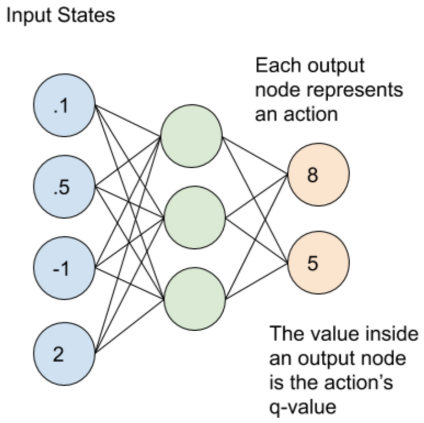

Fig. 1.6. Le réseau neuronal transformant les état en valeur Q

Le fonctionnement de ce réseau neuronal est simple, l'entrée de cette fonction est l'état complet du système. Cet état sera transmis sous la forme d'une valeur Q correspondant à chaque action. Comme expliqué précédemment, l'opération consistera à choisir l'action qui correspond à la valeur Q la plus élevée. Le mécanisme reste comme dans le Q learning :

- Initialiser le réseau neuronal
- Choisir une action et l'appliquer sur l'environnement
- Mettre à jours les poids du réseau neuronal

Remarque Une différence majeure avec le Q Learning est l'utilisation de deux réseaux neuronaux dans cet algorithme ("réseau principal" et "réseau cible"). Ces deux réseaux ont la même architecture mais n'ont pas les mêmes poids pendant l'apprentissage. Cette façon de travailler conduit à un processus d'apprentissage stable et à une grande efficacité.

2. l'apprentissage par renforcement multi-agents

2.1. Introduction

Dans cette partie du travail de recherche, nous allons expliquer l'apprentissage par renforcement dans le cas de multi-agents. Considérons un environnement où les observations et les actions sont distribuées sur n agents. L'environnement peut être totalement observable ou partiellement observable. Dans le cas d'un environnement totalement observable, un agent dispose non seulement d'informations sur lui-même mais aussi sur les autres agents. Contrairement au cas précédent, un agent dans un environnement partiellement observable a des informations limitées sur l'ensemble du système.

chaque agent dans l'environnement a sa propre observation et est responsable de l'action qu'il applique à l'environnement. la récompense donnée à cet agent peut être individuelle ou collective. Un exemple de récompense collective est lorsque l'agent travaille dans un groupe afin d'atteindre un objectif final.

2.2. Apprentissage / Exécution (centralisé vs. décentralisé)

L'apprentissage et l'exécution dans l'apprentissage par renforcement dans le cas d'agents multiples se fait de manière centralisée ou décentralisée. Ces deux termes méritent une explication détaillée afin de comprendre la suite du travail. L'apprentissage centralisé consiste à considérer les agents comme une seule entité lors de l'apprentissage. Cela n'implique pas une exécution centralisée. En revanche, une approche différente consiste à laisser chaque agent apprendre seul en se basant uniquement sur ses propres observations. Le même raisonnement s'applique à l'exécution. Des exemples d'algorithmes sont expliqués dans les sections suivantes.

2.3. Independant Q Learning [3]

Une approche pour résoudre un problème multi-agents en utilisant l'apprentissage par renforcement consiste à traiter chaque agent indépendamment. L'idée est que chaque agent agit seul et considère les autres agents comme une simple partie de l'environnement. Dans ce cas, l'agent exécute un algorithme comme le DQN mentionné dans la section précédente et agit comme s'il était le seul dans l'environnement (figure 2.1).

Malgré sa simplicité, IQL reste un algorithme puissant et efficace pour certains problèmes pour les raisons suivantes :

- Il ne présente pas les problèmes d'extensibilité et de communication qui se posent dans d'autres algorithmes lorsque le nombre d'agents augmente.
- Chaque agent n'a besoin que de l'historique de ses propres observations pour apprendre.

Remarque D'autres extensions de IQL existent comme DQL (Distributed Q-learning) proposé par Lauer et Riedmiller [3]. Cela nécessite un environnement totalement observable dans lequel chaque agent a une idée de l'état complet du système mais ne connaît pas les actions entreprises par les autres agents.

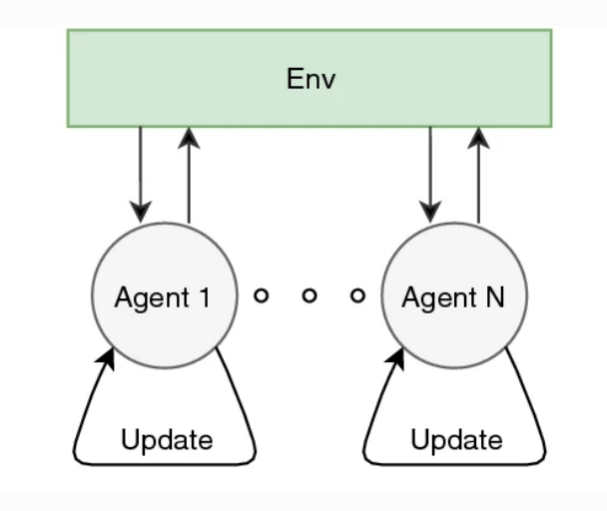

Fig. 2.1. Chaque agent met à jour sa propre politique [3]

2.4. Value Decomposition Networks [8]

Il s'agit d'un algorithme à apprentissage centralisé et à exécution décentralisée. Nous considérons un problème d'apprentissage par renforcement multi-agent dont le but est d'optimiser une récompense collective représentant tous les agents. Chaque agent a accès à sa propre observation (locale ou globale) et peut prendre une action pour agir sur l'environnement.

L'architecture du VDN est illustrée dans la figure ci-dessous à droite 3.1. Elle représente les deux réseaux neuronaux représentant deux agents qui traduisent les observations locales en valeurs Q. Ensuite, ces valeurs sont regroupées par une fonction Q.

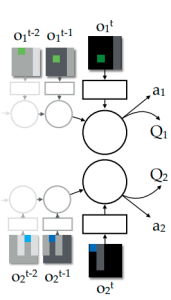

Figure 1: Independent agents architecture showing how local observations enter the networks of two agents over time (three steps shown), pass through the low-level linear layer to the recurrent layer, and then a dueling layer produces individual Q-values.

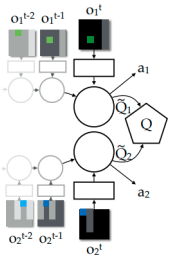

Figure 2: Value-decomposition individual architecture showing how local observations enter the networks of two agents over time (three steps shown), pass through the low-level linear layer to the recurrent layer, and then a dueling layer produces individual "values" that are summed to a joint Q-function for training, while actions are produced independently from the individual outputs.

Fig. 2.2. Gauche : Illustration du modèle de IQL (Independant Q-learning) | Droite : Illustration de VDN (Value Decomposition Networks) [8]

Remarque : dans cette figure, les réseaux de neurones utilisés sont des réseaux de neurones récurrents. La particularité de ces derniers est la prise en compte des anciennes observations dans le calcul de la valeur Q. Ceci est pratique dans le cas d'un environnement partiellement observable.

L'hypothèse principale sur laquelle repose le VDN est de considérer une valeur commune à tous les agents qui est égale à la somme des valeurs Q de chaque agent. La valeur commune est donnée par l'équation 2.1 :

$$Q((O_1, O_2, ..., O_n), (a_1, a_2, ..., a_n)) = \sum_{i=1}^{n} Q_i(O_i, a_i) \tag{2.1}$$

Où Q_i représente la valeur Q de l'agent i ayant une observation O_i et ayant effectué une action a_i. L'apprentissage Q est effectué par rétro-propagation des gradients issue de la règle expliquée dans l'apprentissage Q en utilisant la valeur commune. En d'autres termes, chaque agent observe l'environnement et obtient la valeur Q pour chaque action. Puis il entreprend une action en exploitant son réseau neuronal ou par exploration (action aléatoire). La somme des valeurs Q donne la valeur commune Q_{tot}. En utilisant la récompense commune et Qtot, on calcule la perte ainsi que les gradients qui seront ensuite rétro-propagés dans les réseaux de neurones des agents.

la paire observation-action doit satisfaire la condition IGM (Individual Global Max) représentée par l'équation suivante 2.2 :

$$\underset{a}{\text{argmax}} Qtot(O, a) = (argmax_{a_1} Q_1(O_1, a_1), ..., argmax_{a_2} Q_2(O_2, a_2)) \tag{2.2}$$

Les expériences [8] montrent que les approches utilisées précédemment, telles que IQL, donnent des résultats insatisfaisants. En revanche, VDN montre de bonnes performances pour un large éventail de problèmes.

2.5. QMIX [7]

L'approche QMIX est basée sur le même problème que VDN et propose une amélioration de cet algorithme. il partage donc les valeurs Q pendant l'apprentissage (apprentissage centralisé, exécution décentralisée). Pour pouvoir additionner les valeurs Q de tous les agents VDN, il faut remplir la condition IGM mentionnée ci-dessus. A cette condition, QMIX ajoute une autre condition (L'inéquation 2.3) afin de pourvoir respecter l'équation 2.2. Elle renforce des poids positives dans le réseau de mixage (expliqué plus loin dans le texte).

$$\frac{\delta Q_{tot}}{\delta Q_i} \geq 0, \vee i \tag{2.3}$$

Contrairement à VDN qui utilise la factorisation totale, cet algorithme a juste besoin de s'assurer que l'opération "argmax" performée sur Qtot résulte en la même opération appliquée à chaque agent. Dans ce modèle, chaque agent possède son propre réseau neuronal et fait partie d'un réseau neuronal général qui fournit la valeur Qtot. Notre algorithme va mixer les valeurs Q des agents de manière non linéaire. En plus de cela, il ajoutera des informations provenant de l'état global du système. Le modèle est le suivant :

- le réseau neuronal de l'agent : Il reçoit l'observation et renvoie la valeur Q. Dans cette figure 2.3 , l'agent utilise un réseau neuronal récursif pour prendre en compte non seulement l'observation mais aussi la dernière action effectuée par l'agent. Selon certaines études, cela facilite l'apprentissage et accélère la convergence.

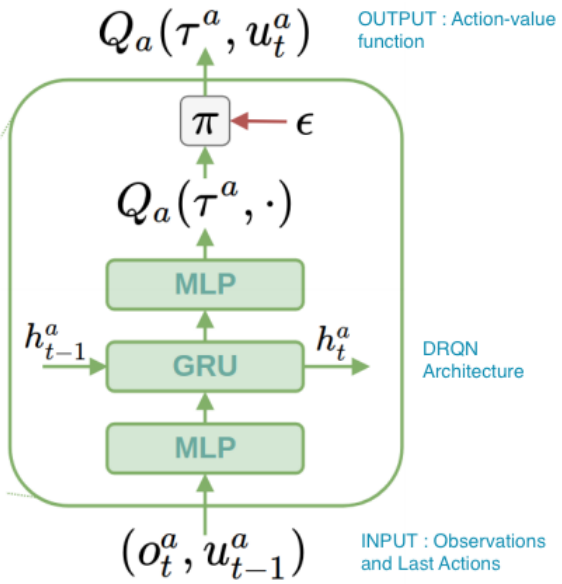

Fig. 2.3. le réseau neuronal de l'agent [2]

- Le réseau de mixage : Il prend les valeurs Q de chaque agent et donne en sortie la valeur Q du système global. Ce qui est intéressant dans ce réseau, c'est que les poids du réseau neuronal sont générés par ce qu'on appelle un "hyper-réseau" (marquée en rouge sur la figure ci-dessous 2.5). Il s'agit d'un réseau neuronal dont l'entrée est l'état du système.

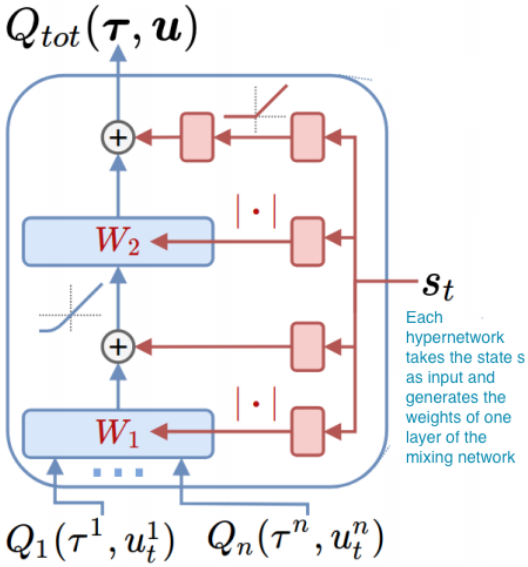

Fig. 2.4. Le réseau de mixage [2]

Remarque : Pour le réseau de mixage, les pondérations sont forcées d'être positives, contrairement aux biais.

L'architecture globale de ce modèle est la suivante :

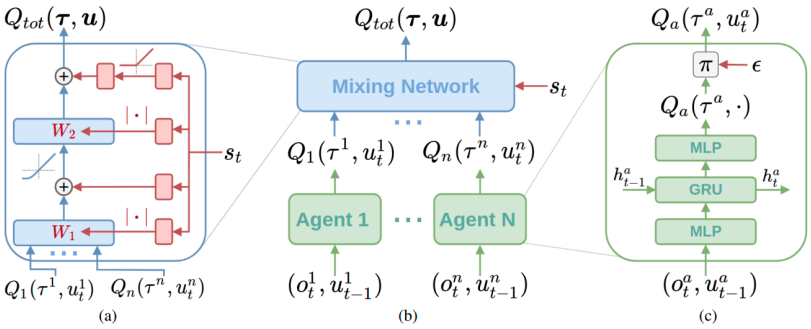

Figure 2. (a) Mixing network structure. In red are the hypernetworks that produce the weights and biases for mixing network layers shown in blue. (b) The overall QMIX architecture. (c) Agent network structure. Best viewed in colour.

Fig. 2.5. L'architecture générale de QMIX[7]

l'algorithme fonctionne comme suit :

- initialiser les réseaux neuronaux des agents et du mixeur (principal et cible)
- prendre l'état du système pour chaque agent
- choisir une action suivant Epsilon Greedy pour chaque agent
- stocker toutes les informations nécessaires à l'entraînement (observation, action effectuée, nouvelle observation et récompense) et communiquer les valeurs de Q avec le réseau de mélange
- En utilisant les poids calculés avec les hyper-réseaux, calculer la valeur Qtot du système
- Calculer la perte en utilisant Qtot et la récompense communiquée par l'environnement
- Calculer les gradients et les rétro-propager dans les réseaux des agents.

En conclusion, ce modèle basé sur un apprentissage centralisé et une exécution décentralisée a montré son efficacité par rapport aux autres modèles d'apprentissage par renforcement. Il est important de faire attention aux contraintes imposées pour valider l'utilisation du modèle.

Ceci résume la théorie nécessaire et les bases qu'il faut maîtriser pour comprendre le reste du travail. Dans les prochains chapitres, nous nous concentrerons sur la tâche et l'objectif de ce sujet.

3. Étude de l'environnement

3.1. PettingZoo [5]

Il existe différents environnements qui ont été développés dans le domaine de l'apprentissage par renforcement. L'API OpenAI gym est considérée comme la norme pour le cas d'un agent simple. Elle est ensuite devenue une base pour le développement d'environnements à multiple agents. MARL (Multi Agent Reinforcement Learning) n'a pas de modèle mental et mathématique fixe comme le cas de l'agent simple. Cela a conduit à l'existence de plusieurs implémentations qui ne fonctionnent pas de la même manière. L'absence de normalisation et de standardisation implique un temps de recherche énorme pour adapter les algorithmes aux environnements et bloque le développement de bibliothèques utilisables partout.

C'est là que PettingZoo entre en jeu. Il s'agit d'une bibliothèque et d'une API qui rassemble les points positifs des API qui existent déjà dans le monde. Premièrement, PettingZoo est un environnement simple à utiliser comme Gym, avec une bonne interface et un grand nombre d'environnements dans un seul paquet. Deuxièmement, cette bibliothèque est développée pour MARL. Elle peut donc supporter un grand nombre d'agents. Les agents, comme dans le cas de ce travail, peuvent mourir et donc disparaître de l'environnement. Les agents peuvent être sélectionnés pour participer à un épisode.

PettingZoo est basé sur ce que l'on appelle le "cycle agent-environnement" (AEC) [5]. L'idée est d'itérer entre les agents. L'agent effectue une action et l'agent suivant est alors sélectionné. La figure ci-dessous décrit comment cela fonctionne :

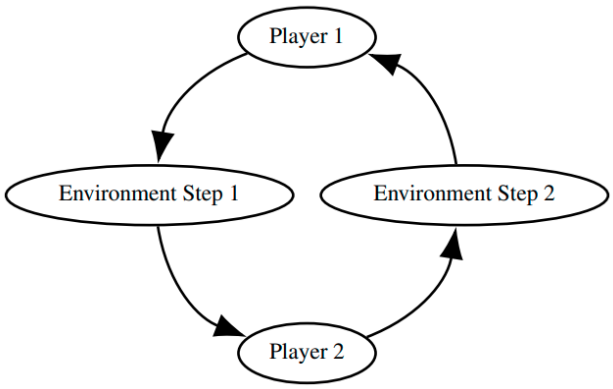

Fig. 3.1. Le cycle AEC) [5]

Étant donné l'importance de l'apprentissage par renforcement multi-agent, PettingZoo peut être considéré comme un standard pour MARL, comme l'est Gym pour "Simple Agent". Il facilite la recherche académique et la familiarisation avec les MARL. Un point négatif à mentionner est le dysfonctionnement de l'environnement lorsque le nombre d'agents est très élevé ($\geq 10,000$ agents) .

3.2. defense-v0

3.2.1. Introduction

L'environnement sur lequel les travaux ont été effectués, a été développé par le Cdt BOECX basé sur AEC. Il s'agit d'un environnement entièrement observable qui décrit un champ de bataille. deux équipes sont présentes lors de la création de l'environnement, l'équipe bleue et l'équipe rouge. A chaque itération, les agents de chaque groupe doivent collaborer et travailler ensemble pour éliminer les agents de l'autre groupe. Chaque agent peut à chaque itération aller à gauche, à droite, en haut, en bas ou n'exécuter aucune action. Afin d'éliminer un agent adverse, un agent peut viser son adversaire et tirer. L'ennemi est considéré comme mort s'il se trouve dans le rayon d'action de l'agent.

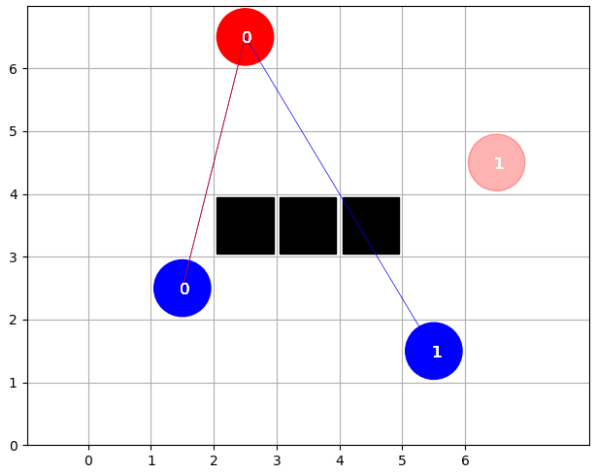

Fig. 3.2. Exemple de mise en place de l'environnement

La figure ci-dessus 3.2 représente un champ de bataille avec deux agents dans chaque groupe. Les lignes pointillées représentent l'action "viser" et la couleur de la ligne représente l'agent qui effectue cette action. Les carrés noirs indiquent la présence d'obstacles à ses coordonnées. Ceux-ci vont limiter le mouvement des agents et les bloquer si la ligne de tir les traverse.

Le terrain est généré par un fichier .ter où les agents sont représentés par 1 ou 2, les obstacles par des x et le reste de l'environnement par des points.

2.....2
.....
.....
..xxx..
.....
.....
1.....1

3.2.2. Fonctionnement

Comme defense-v0 est basé sur AEC, la navigation dans les agents est également facile à faire en utilisant l'exemple de code ci-dessous :

```
env = defense_v0.env(terrain='central_5x5', max_cycles = MAX_CYCLES)
env.reset()
for agent in env.agent_iter():
    obs, reward, done, info = env.last()
    action = policy(obs) if not done else None
    env.step(action)
```

Avec la fonction env.last, l'agent obtient des informations intéressantes pour choisir son action. Ces informations vont être détaillées :
L'observation contient deux vecteurs. Le premier indique l'état dans lequel se trouve l'agent en donnant sa position (x,y), son statut mort ou vivant (0,1), ce qu'il lui reste en nombre de munitions et l'agent sur lequel il tire.Le second vecteur ne contient que des valeurs booléennes donnant les actions qui peuvent être utilisées dans cet état.
En plus de cela, l'agent reçoit la récompense pour avoir effectué sa dernière action et reçoit son statut de vivant ou de mort.

Remarque : Les vecteurs peuvent avoir des tailles différentes en fonction du nombre d'agents. Par exemple, l'action "tirer" sera constituée de trois actions consécutives, chacune correspondant à un adversaire. En outre, un agent qui est mort au cours d'un épisode est retiré de la liste des agents de l'environnement.

4. Communication entre les agents

5. Conclusion

A. Information supplémentaire

Bibliographie

[1] Ankit Choudhary. A hands-on introduction to deep q-learning using openai gym in python.

[2] gema.parreno.piqueras. Qmix paper ripped : Monotonic value function factorization for deep multi-agent reinforcement learning in starcraft ii.

[3] Sven Gronauer and Klaus Diepold. Multi-agent deep reinforcement learning : a survey. *Artificial Intelligence Review*, 2021.

[4] Yanhua Huang. Playing atari with deep reinforcement learning 2020.

[5] Benjamin Black ... J. K. Terry. Pettingzoo : Gym for multi-agent reinforcement learning. 2020.

[6] Maxim Lapan. *Deep Reinforcement Learning Hands-On*. Packt Publishing, Birmingham, UK, 2018.

[7] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix : Monotonic value function factorisation for deep multi-agent reinforcement learning. volume 10, 2018.

[8] P. Sunehag, W.M. Czarnecki, M. Lanctot, G. Lever, V. Zambaldi, N. Sonnerat, A. Gruslys, M. Jaderberg, J.Z. Leibo, K. Tuyls, and T. Graepel. Value-decomposition networks for cooperative multi-agent learning, 2017.

[9] Andre Violante. Simple reinforcement learning : Q-learning.

[10] Mike Wang. Deep q-learning tutorial : mindqn.