Sorbonne Université
Master ANDROIDE

Apprentissage conditionné par des buts en BBRL

UE de projet M1

Roxane Cellier – Yi Qin – Zhenyue Fu

2023

Table des matières

Chapitre 1

Introduction

L'objectif de notre projet a été d'étudier différentes méthodes d'apprentissage par renforcement. En commençant par des méthodes tabulaires simples (Q-Learning par états-actions), nous avons dû étendre nos codes et algorithmes pour nous permettre d'implémenter et d'étudier des techniques plus poussées. Le principal algorithme autour duquel nous avons centré nos efforts se nomme Hindsight Expérience Replay, que nous avons tenté d'implémenter suivant différents modèles. Notre étude s'est divisée en deux parties : d'abord dans un environnement de type labyrinthe par la recherche par un agent d'un chemin vers un (ou plusieurs) but(s), puis dans un environnement de type CartPole pour chercher la meilleure stratégie permettant à l'agent d'équilibrer le pendule inversé.

Tout au long du projet, nous avons cherché à surveiller de près les performances de l'algorithme HER dans différents environnements et autour de différents objectifs, et la manière dont il aide les agents à trouver la meilleure politique d'action. Nous avons travaillé en étroite collaboration avec notre mentor Olivier Sigaud, qui nous a permis de comprendre et de nous plonger au cur des techniques l'apprentissage par renforcement, au fur et à mesure de l'avancement de ce projet. Nos implémentations et études sont présentées dans ce dépôt github.

Une grande partie des algorithmes sur lesquels nous avons dû nous attarder ne nous ont jamais été présentés pendant notre parcours. Pour nous permettre une bonne implémentation et utilisation de ces différents algorithmes, nous avons dû en premier lieu les étudier attentivement. Nous avons pour ce faire lu avec attention un certain nombre d'articles qui nous étaient fournis ou que nous avons dû trouver par nous-mêmes. La plupart de ces articles ont par ailleurs été présentés dans notre carnet de bord, et sont cités tout au long de ce rapport.

Une autre étude indispensable a été faite sur de multiples librairies de Python. En effet, nous avons utilisé à de multiples reprises des modules proposés par Python et d'autres développeurs externes, que nous devons comprendre pour nous assurer une utilisation correcte. Les principales librairies sont NumPy pour le travail sur tableau et PyTorch pour l'utilisation de réseaux de neurones, mais aussi Gym et BBRL, présentées plus loin dans le rapport.

Chapitre 2

État de l'art

Dans cette partie, nous cherchons à réintroduire les notions de base de l'apprentissage par renforcement, en expliquer les principes et les concepts, pour ensuite approfondir sur les méthodes et algorithmes utilisés pour notre implémentation. Nous commençons donc par un état de l'art rapide et général de l'apprentissage par renforcement, avant de passer au Q-Learning, pour finir sur l'algorithme de Hindsight Experience Replay. Sans entrer dans tout le détail des théories probabilistes, nous essayons tout de même de présenter les algorithmes et leurs paramètres.

2.1 Apprentissage par Renforcement

L'apprentissage par renforcement [1] est une méthode d'apprentissage automatique qui permet à un agent d'apprendre à force d'essais et de récompenses, basée sur le formalisme des processus décisionnels de Markov. Son principal objectif est d'apprendre seul une politique d'actions pour optimiser la prise de décision dans un environnement.

Considérons donc un agent évoluant dans un environnement. Cet agent peut se trouver dans différents états s parmi l'ensemble des états possibles de $\mathcal{S}$, et effectuer un certain nombre d'actions a parmi son espace des actions $\mathcal{A}(s)$. Il utilise ainsi une politique π pour déterminer son plan d'action selon son état, en cherchant généralement à maximiser les récompenses reçues. Lorsque l'agent effectue une action, l'environnement lui transmet une récompense, dépendante de l'état et l'action effectuée.

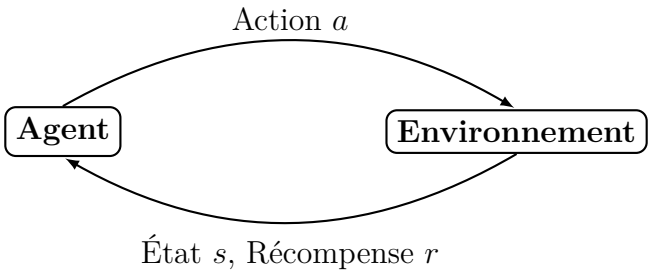

FIGURE 2.1 – Schéma du principe d'apprentissage par renforcement

L'agent apprend alors au fil de ses essais dans l'environnement les meilleures actions à effectuer sur chacun des états. Généralement, l'agent commence par une phase dite

d'exploration suivie d'une phase d'exploitation. Dans sa phase d'exploration, il agit plus ou moins aléatoirement, autorisant des actions qui ne sont pas forcément optimales en récompense. Cela lui permet de tester un large éventail de possibilités et de connaître les récompenses d'un grand nombre de situations. Dans sa phase d'exploitation, il agit dans le but d'obtenir la meilleure récompense possible. Cet apprentissage cependant peut se retrouver long et fastidieux, lorsque l'environnement est grand et que le champ d'action l'est également.

De nombreux domaines utilisent de l'apprentissage par renforcement, qui peut s'adapter à différents types de problèmes, notamment d'optimisation comme par exemple en robotique, gestion de ressources, planification, etc. L'apprentissage par renforcement a également permis de créer des programmes capables de résoudre de multiples jeux, comme différents jeux de la gamme Atari [2, 3], ou le jeu de Go [4], tous deux proposés par Google DeepMind. Plus récemment encore, c'est entre autres de l'apprentissage par renforcement qui a été utilisé par OpenAI pour l'entraînement de plusieurs modèles génératifs, comme GPT-3 [5], utilisé principalement par son agent conversationnel Chat-GPT, sorti fin 2022. De par ses multiples applications possibles, ce domaine se situe au cœur de la recherche actuelle en apprentissage automatique. Il y a donc régulièrement des nouveautés et avancées notamment concernant les algorithmes et les optimisations.

2.2 Q-Learning

Le Q-learning [6] est un algorithme d'apprentissage par renforcement, qui tire son nom de la fonction Q utilisée pour le calcul de récompense d'une action a exécutée dans un état s donné. Cette fonction calcule donc le gain potentiel de cette action sur le long terme, et permet de supposer à chaque instant quelle est la meilleure action à prendre pour ainsi maximiser la récompense totale. Ces gains potentiels sont stockés dans ce qu'on appelle la Q-table. C'est donc un algorithme dit tabulaire, c'est-à-dire adapté à un environnement et un espace des actions finis.

L'algorithme de la méthode Q-Learning peut être présenté de cette manière :

Entrées *Facteur d'apprentissage* $\alpha \in [0,1]$, *Paramètre de politique* $\epsilon, \tau \in [0,1]$, *Nombre d'épisodes* N ;
Résultat : Table Q
Initialiser $Q(s,a)$ arbitrairement $\forall s \in S, a \in A(s)$, et $Q(but, \cdot) = 0$;
$e \leftarrow 1$;
while $e \leq N$ **do**
    $s \leftarrow$ état initial choisi dans S ;
    while $s \neq but$ **do**
        $a \leftarrow$ action choisie dans $A(s)$ suivant la politique ($\epsilon - greedy$ ou $softmax$) ;
        Exécuter action a ;
        $r, s' \leftarrow$ récompense observée, nouvel état ;
        $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma max_{a'} Q(s',a') - Q(s,a)]$;
        $s \leftarrow s'$;
    end
    $e \leftarrow e + 1$;
end
return Q ;

Algorithme 1 : Algorithme de Q-Learning

Un état est donc défini comme objectif global de la procédure, et cette méthode étant à objectif unique, l'agent n'aura que ce but à résoudre. L'état de départ de l'agent est généré de manière aléatoire, et l'agent se déplace alors jusqu'à la fin de son épisode d'exploration.

Lors d'un mouvement d'un état s à un autre s', l'agent reçoit le gain potentiel r déterminé par la Q-table. Cette récompense est alors rétro-propagée sur létat s, en fonction de deux valeurs : le facteur d'apprentissage α et le facteur d'actualisation γ. Le premier facteur détermine à quel point la nouvelle information calculée va remplacer la précédente. Si α vaut 0 alors l'agent n'apprendra rien, et au contraire s'il vaut 1, l'agent ne considérera que la dernière information dans son calcul. Le second facteur détermine l'importance des récompenses à venir dans le calcul de la récompense actuelle. Si γ vaut 0, alors l'agent prends en compte uniquement la récompense immédiate. S'il vaut 1, toutes les récompenses futures deviennent aussi importantes que la récompense immédiate.

Comme on peut le constater dans le détail de l'algorithme, le Q-Learning est un algorithme dit off-policy. C'est-à-dire que l'on considère deux politiques différentes selon l'étape de l'algorithme. Lors de la phase d'apprentissage, ou d'exploration, nous utilisons une stratégie fixe de choix de l'action optimale. Nous en considérons ici deux en particulier : la stratégie $\epsilon - greedy$ et la stratégie $softmax$. La première permet, dans un état donné, de choisir avec une forte probabilité $(1 - \epsilon)$ la meilleure action proposée par la Q-table, ou de tirer une action de façon aléatoire avec une très faible probabilité (ϵ). La seconde permet de choisir parmi les actions selon des probabilités proportionnelles à leurs gains potentiels. Ainsi, une action avec un gain moyen aura plus de chance dêtre tirée qu'une action avec un faible gain, mais aussi moins de chance qu'une action avec un gain plus élevé. Ensuite, lors de la phase de mise à jour de la Q-Table, la valeur du gain potentiel est estimée en fonction de la meilleure action possible dans létat suivant s'.

De nombreuses techniques dérivant du Q-Learning existent, comme le double Q-learning [7], qui met en place une mise à jour croisée entre deux fonctions Q d'évaluations s'entraînant sur deux ensembles différents. Cette méthode est entre autres utilisée pour éviter la

surestimation et accélérer le temps d'apprentissage dans des environnements bruités.

Une autre technique est le Deep Q-Learning (ou Deep Q-Network, DQN), qui applique le principe de l'algorithme tabulaire sur un réseau profond de neurones. Cela permet de passer d'une représentation tabulaire à une représentation continue, et ainsi ne plus être limité par des espaces d'états finis. C'est cet algorithme qui a été utilisé par Google DeepMind pour son programme capable de résoudre les jeux de la gamme Atari, présenté plus haut [2].

2.3 Goal-Conditioned Reinforcement Learning

GCRL (Goal-Conditioned Reinforcement Learning) [8] est une méthode d'apprentissage par renforcement axée sur les objectifs. Elle vise à résoudre les problèmes auxquels sont confrontés les algorithmes traditionnels d'apprentissage par renforcement lorsqu'il y a plusieurs objectifs différents dans une tâche. Dans GCRL, chaque tâche a un objectif clair, et la politique de l'agent ainsi que les fonctions de récompense sont liées à cet objectif.

Dans GCRL, nous devons étendre le Processus de Décision Markovien (MDP) traditionnel en définissant un tuple étendu qui comprend l'espace des états ($\mathcal{S}$), l'espace des actions ($\mathcal{A}$), la fonction de transition d'état ($\mathcal{P}$), l'espace des objectifs ($\mathcal{G}$) et l'application (ϕ) de l'état actuel à l'espace des objectifs. Dans GCRL, la fonction de récompense est liée aux états, aux actions et aux objectifs. Par exemple, nous pouvons définir une fonction de récompense $r_g(s_t, a_t, s_{t+1})$ qui attribue une récompense de 1 lorsque l'agent est proche de son objectif, et de 0 sinon.

2.4 Hindsight Experience Replay

Les récompenses dans GCRL sont souvent très éparses. En raison des difficultés de l'agent à atteindre l'objectif au début de l'apprentissage et ne recevoir qu'une récompense nulle, cela ralentit considérablement l'ensemble du processus d'apprentissage. Existe-t-il une méthode pour exploiter efficacement ces expériences "ratées" ? C'est dans cette optique que l'algorithme de Hindsight Experience Replay (HER) [9], a été proposé lors de la conférence Neural Information Processing Systems (NeurIPS) en 2017, devenant ainsi l'une des méthodes classiques de GCRL.

Hindsight Experience Replay est un algorithme ayant pour objectif d'accélérer la découverte d'une politique optimale dans un contexte multi-objectifs. Le principe de base est simple : lorsque l'on ne parvient pas à atteindre le but défini à l'origine, on considère alors les états traversés comme s'ils étaient nos objectifs, et l'on recalcule alors nos récompenses en considérant ces nouveaux objectifs. Les récompenses ne sont donc plus calculées pendant la phase de marche, mais une fois l'épisode fini. Voici l'algorithme appliqué à une fonction d'apprentissage Q :

Entrées *Facteur d'apprentissage* $\alpha \in [0,1]$*, Paramètre de politique* $\epsilon, \tau \in [0,1]$*, Nombre d'épisodes* N*, Nombre maximal d'étapes* M*, Nombre de mise à jour* R ;
Résultat : Table Q
Initialiser $Q(s, but, a) = 0$, $\forall s, but \in S, a \in A(s)$;
$e \leftarrow 1$;
while $e \leq N$ **do**
  $s_1 \leftarrow$ état initial choisi dans S ;
  $but \leftarrow$ état but choisi dans S ;
  $RB \leftarrow$ initialisation du ReplayBuffer ;
  $t \leftarrow 1$;
  while $t \leq M$ ***AND*** $s_t \neq but$ **do**
    $a_t \leftarrow$ action choisie dans $A(s)$ suivant la politique ($\epsilon - greedy$ ou $softmax$) ;
    Exécuter action a_t ;
    $r_t, s_{t+1} \leftarrow$ récompense observée considérant l'état visé *but*, nouvel état ;
    Stocker dans RB la transition $(s_t, a_t, r_t, s_{t+1}, but)$;
    $s_t \leftarrow s_{t+1}$;
    $t \leftarrow t + 1$;
  end
  foreach *étape* t **do**
    Tirage aléatoire d'un ensemble G parmi les états traversés ;
    foreach $g \in G$ **do**
      $r'_t, s'_{t+1} \leftarrow$ récompense observée considérant l'état visé g, nouvel état ;
      Stocker dans RB la transition $(s_t, a_t, r'_t, s'_{t+1}, g)$;
    end
  end
  for $i = 1, R$ **do**
    Tirer aléatoirement un mini-batch B du ReplayBuffer RB ;
    foreach $(s, a, r, s', goal) \in B$ **do**
      $Q(s, goal, a) \leftarrow Q(s, goal, a) + \alpha[r + \gamma max_{a'} Q(s', goal, a') - Q(s, goal, a)]$;
    end
  end
  $e \leftarrow e + 1$;
end
return Q ;

Algorithme 2 : Algorithme de Hindsight Experience Replay appliqué à Q-Learning

Plus formellement, supposons que nous utilisions la politique π pour explorer l'environnement avec l'objectif g et que nous obtenions une trajectoire $s_1, s_2, ..., s_T$ avec $g \neq s_1, s_2, ..., s_T$. Cela signifie que sur toute la trajectoire, les récompenses obtenues sont toutes nulles, ce qui offre peu d'aide pour notre apprentissage. Alors, que se passerait-il si nous examinions cette trajectoire sous un nouvel objectif g' ? En d'autres termes, bien que nous n'ayons pas atteint l'objectif initial g, la politique a accompli les objectifs correspondant à $s_1, s_2, ..., s_T$, c'est-à-dire $\phi(s_1), \phi(s_2), ..., \phi(s_T)$. Si nous utilisons ces nouveaux objectifs pour remplacer l'objectif initial g par le nouvel objectif g' et re-calculons les récompenses le long de la trajectoire, cela permet à la politique d'obtenir des informations utiles pour l'apprentissage à partir de ces expériences dites "ratées".

Pour nous permettre de calculer ces nouvelles valeurs à posteriori, il faut stocker les transitions effectuées tout au long de la marche. Nous utilisons donc pour cela un Replay

Buffer, dans lequel nous stockons à chaque pas les données nous permettant le calcul (état de départ, action effectuée, état d'arrivée, récompense, et but considéré). Aux transitions stockées de cette manière, nous ajoutons comme expliqué plus haut de nouvelles transitions à partir de ces dernières, auxquelles nous modifions simplement le but considéré pour le remplacer par l'état atteint. Nous re-traitons ensuite ces étapes pour ainsi calculer leurs nouvelles récompenses associées. Le tout est stocké dans ce buffer, et nous permet ainsi, pour une marche et un but précis, de considérer une multitude de pas pour différents objectifs.

Cet algorithme (et ceux suivant le principe de l'Experience Replay) est largement utilisé dans diverses implémentations de Q-Learning [10]. C'est en effet un pilier de nombre de recherches actuelles dans le domaine de l'apprentissage par renforcement. Une autre application, ici dans l'objectif d'une utilisation en environnement continu, utilise l'algorithme DQN couplé à un algorithme d'Experience Replay [11]. Cela à pour but d'augmenter remarquablement la vitesse d'apprentissage.

Chapitre 3

Contribution

3.1 Environnements

Pour nous permettre de comprendre et visualiser le fonctionnement des algorithmes, et pour pouvoir comparer nos différents résultats et implémentations, nous avons utilisé tout au long du développement différents types d'environnements. Nous allons donc commencer par une courte description de nos environnements.

Dans un premier temps, nos tests se sont fait par la recherche par un agent d'un chemin vers un (ou plusieurs) but(s) dans un labyrinthe. L'environnement utilisé pour ces tests est l'environnement MazeMDPEnv de la librairie bbrl_gym, développé par notre encadrant Olivier SIGAUD et Benjamin PIWOWARSKI. Inspiré des environnement de Gym [12], ce module permet de générer de façon aléatoire un labyrinthe, représenté sous forme de quadrillage, où chaque case correspond à un état. Dans chaque état, quatre actions sont possibles, aller vers le haut, vers le bas, vers la droite et vers la gauche. Il est possible également de placer des murs sur certains états, pour ainsi bloquer et complexifier la tâche de notre agent. Son objectif dans cet environnement est d'apprendre la meilleure politique de déplacement jusqu'à un but. Les paramètres de génération sont la largeur et la hauteur du labyrinthe, ainsi que le ratio de case correspondant à des murs.

Dans un second temps, nous avons effectué nos tests sur l'environnement CartPole, proposé par la librairie Gym développée par OpenAI. Il présente le problème du pendule inversé. L'état de l'agent est représenté par un quadruple représentant la position du kart, sa vélocité, l'angle du pendule, et sa vélocité. Ces valeurs étant continues, nous ne pouvons donc pas les tester avec un algorithme tabulaire. Ses actions possibles sont d'aller à droite ou à gauche. L'objectif de base d'un agent dans cet environnement est de maintenir le pôle en équilibre aussi longtemps que possible. Nous avons utilisé cet environnement au travers de la librairie bbrl, développée par Olivier SIGAUD, Olivier SERRIS et Benjamin PIWOWARSKI, et inspirée de la librairie SaLina [13].

3.2 Q-Learning en 3 dimensions

Notre premier objectif au sein de ce projet a été de prendre en main l'algorithme de Q-Learning, pour en comprendre le sens et les objectifs, et ainsi nous permettre de l'implémenter correctement autour d'une évolution de l'algorithme et d'étudier son compor-

tement.

3.2.1 Implémentation

L'idée était d'adapter l'algorithme du Q-Learning présenté en Section 2.2 pour le transformer en un algorithme d'apprentissage par renforcement conditionné par les buts (GC-RL), et de le mettre en place dans un contexte multi-objectifs. Ainsi, plutôt que de représenter la fonction comme un tableau en deux dimensions, c'est-à-dire dépendant de l'état s et de l'action considérée a, nous avons considéré un tableau en trois dimensions. Le gain dépend alors de l'état s, de l'action a, et de l'état objectif *but* que l'agent cherche à atteindre. De cette manière, chaque état peut être considéré comme un potentiel but. L'objectif de l'agent devient alors de déterminer la meilleure politique de déplacement pour chacun des buts possibles.

L'algorithme que nous avons utilisé se décrit donc comme suit :

Entrées *Facteur d'apprentissage* $\alpha \in [0, 1]$, *Paramètre de politique* $\epsilon, \tau \in [0, 1]$, *Nombre d'épisodes* N, *Nombre maximal d'étapes* M ;
Résultat : Table Q
Initialiser $Q(s, but, a) = 0$, $\forall s, but \in S, a \in A(s)$;
$e \leftarrow 1$;
while $e \leq N$ **do**
    $s \leftarrow$ état initial choisi dans S ;
    $but \leftarrow$ état but choisi dans S ;
    $t \leftarrow 1$;
    while $t \leq M$ ***AND*** $s \neq but$ **do**
        $a \leftarrow$ action choisie dans $A(s)$ suivant la politique ($\epsilon - greedy$ ou $softmax$) ;
        Exécuter action a ;
        $r, s' \leftarrow$ récompense observée considérant l'état visé *but*, nouvel état ;
        $Q(s, but, a) \leftarrow Q(s, but, a) + \alpha[r + \gamma max_{a'} Q(s', but, a') - Q(s, but, a)]$;
        $s \leftarrow s'$;
        $t \leftarrow t + 1$;
    end
    $e \leftarrow e + 1$;
end
return Q ;

Algorithme 3 : Algorithme de Q-Learning conditionné par les buts

Dans son principe, nous pouvons remarquer que l'algorithme ne change pas beaucoup de sa version initiale. La formule mathématiques utilisée est identique, de même que l'alternance entre une phase d'exploration et de mise à jour de la Q-table. Quelques différences cependant sont notables, comme présentées ci-dessous.

Tout d'abord, l'ajout du choix d'un but, qui a lieu en même temps que le choix de l'état de départ de l'agent. Ce choix est utilisé lors du calcul du gain par la fonction Q. En effet, dans un état s, l'action a ne rapporte pas forcément la même récompense pour deux objectifs différents. Il était donc important de séparer les tables de gains pour permettre à l'agent de déterminer une politique différente pour chacun des états possibles.

Ensuite, pour encadrer notre temps d'exécution, nous n'avons pas utilisé à l'identique l'algorithme présenté précédemment. En effet, même si l'agent finit le plus souvent par

atteindre son objectif, il est cependant possible que le chemin soit obstrué par des murs ou que l'agent passe longtemps autour de son objectif sans l'atteindre. Pour éviter des temps de calcul trop longs, nous avons instauré une limite M du nombre de pas que peut effectuer notre agent dans l'environnement sur toute la durée d'un épisode. Si cette limite est atteinte alors que l'agent n'a toujours pas rejoint son objectif, l'épisode s'arrête et le suivant est lancé.

En terme de code supplémentaire, il nous a fallu implémenter des méthodes pour nous permettre de modifier l'environnement. En effet, par défaut dans MazeMDPEnv, l'état but est la dernière case non mur du labyrinthe (dans le coin inférieur droit). C'est également suivant cette indication que le calcul de la récompense est fait. Pour permettre à l'environnement de fonctionner correctement dans cette nouvelle configuration, nous avons donc dû modifier les paramètres de l'environnement à chaque tirage d'un nouveau but, ré-écrivant l'état final, les transitions et le calcul de la récompense. Pour minimiser le nombre de calculs effectués, nous initialisons dès le démarrage de l'algorithme des matrices contenant pour chaque but les bonnes données. Ainsi, à chaque changement de but, nous appelons la fonction suivante :

```
r""" modifie les paramètres de la MDP du MazeMDPEnv en fonction du but à
    atteindre """
def maj_goal(mdp: MazeMDPEnv, but, r_list, P_list):
    # modification de la récompense
    mdp.mdp.r = r_list[but]

    # modification des transisitons
    mdp.P = P_list[but]
    mdp.mdp.P = P_list[but]

    # modifie le plotter
    mdp.mdp.plotter.terminal_states = [but]
```

À noter aussi que dans cette implémentation, les valeurs de la Q-table ne sont pas initialisées arbitrairement. Nous les initialisons toutes à zéro pour laisser l'agent progresser par lui-même. Pour la récompense, nous considérons une récompense brute de 1 à l'état objectif, et de 0 partout ailleurs. Les gains potentiels ne sont donc calculés que par rétropropagation de la récompense originale au fur et à mesure des épisodes.

3.2.2 Expérimentations

Les premières expérimentations nous permettent de vérifier le bon fonctionnement de notre algorithme. Nous avons considéré pour ce cas un labyrinthe de taille 4×4, avec un ratio de mur de 0.25. Nous ferons donc nos tests sur le labyrinthe suivant :

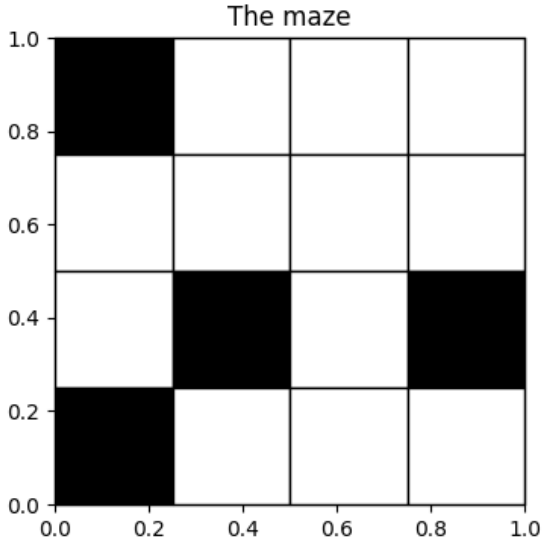

FIGURE 3.1 – Labyrinthe

De par l'ajout d'une dimension à notre problème, il est évident de remarquer que l'apprentissage est plus long qu'avec un algorithme de Q-Learning mono-objectif. Cependant il y a plusieurs aspects sur lesquels nous pouvons nous questionner. Nous avons cherché à étudier l'efficacité de la politique choisie de la phase d'exploration, ainsi que l'influence des différents paramètres sur nos résultats et sur la vitesse d'apprentissage. Nous voulions ainsi tenter de déterminer la meilleure manière d'optimiser cette adaptation de Q-Learning.

En premier lieu, nous nous sommes posé la question de s'il existait une stratégie d'apprentissage plus efficace que l'autre. Nous nous sommes donc demandé s'il était pertinent d'apprendre nos buts les uns à la suite des autres plutôt qu'en les tirant aléatoirement à chaque épisode. Pour faire ce test, nous avons fixé comme stratégie de politique la stratégie $\epsilon - greedy$, avec $\epsilon = 0.1$. Nous avons choisi comme taille maximale d'épisode 50, et 200 épisodes par but possible, pour un total de 2400 épisodes. L'exécution nous renvoie le graphique suivant :

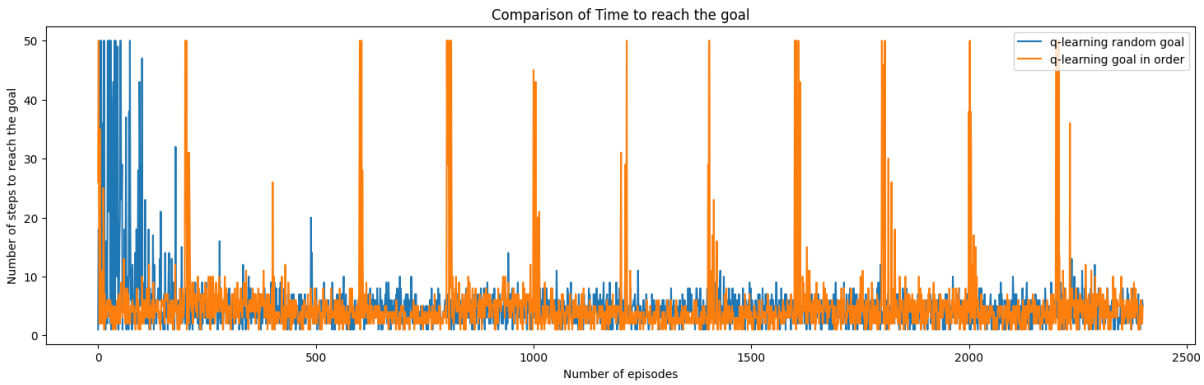

FIGURE 3.2 – Comparatif entre tirage aléatoire et apprentissage dans l'ordre

Il semblerait d'après ce graphique qu'une stratégie de tirage aléatoire soit plus efficace

que de chercher à apprendre les politiques but par but. Le tirage aléatoire permet en effet de faire progresser l'agent en même temps sur l'ensemble des objectifs. Ainsi, le nombre de pas à parcourir pour atteindre l'objectif initialisé au début de l'épisode chute drastiquement lors des 200 premiers épisodes. L'apprentissage but par but en revanche permet d'apprendre rapidement une politique optimale pour un but donné. Chaque pic visible sur le graphe correspond au changement de but. Cette stratégie peut être intéressante selon la problématique. Cependant, si le nombre d'objectifs est trop important, cette stratégie peut se retourner contre l'agent, puisqu'il est alors obligé de faire sa phase d'apprentissage sur l'ensemble des buts avant de s'être construit une politique fiable. De plus, nous pourrions discuter du fait qu'un tirage aléatoire ressemble plus au mode d'apprentissage naturel des êtres vivants. Dans la suite de nos expérimentations, nous avons considéré la stratégie de tirage aléatoire au début de chaque épisode.

Un autre question que nous nous sommes posée a été de trouver quel algorithme de choix d'action permet à Q-Learning d'apprendre le plus efficacement. Pour ce test nous avons considéré les valeurs 0.1 pour le paramètre de $\epsilon - greedy$ et le paramètre de $softmax$, et 500 épisodes au total.

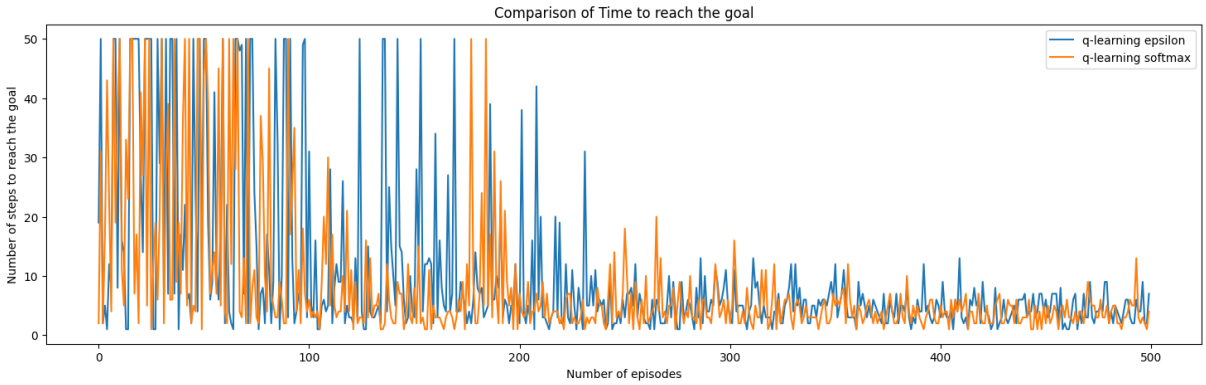

FIGURE 3.3 – Comparatif entre $\epsilon - greedy$ et $softmax$

Il n'est pas évident à partir de ce graphique de réellement déterminer la stratégie de choix d'action. À première vue, l'algorithme du $softmax$ semble faire décroître le nombre d'étapes par épisode plus rapidement, en effet la courbe d'epsilon continue à proposer de nombreux pics alors que le nombre d'étape par $softmax$ semble de plus en plus faible. Mais la stabilisation globale se fait pour les deux algorithmes autour du 250^e épisode, et les deux oscillations se croisent régulièrement par la suite.

Notre dernière étude de Q-Learning sest portée sur l'influence des paramètres des différentes stratégies de choix d'action. Pour chaque stratégie, nous avons observé la vitesse d'apprentissage pour différentes valeurs de leurs paramètres.

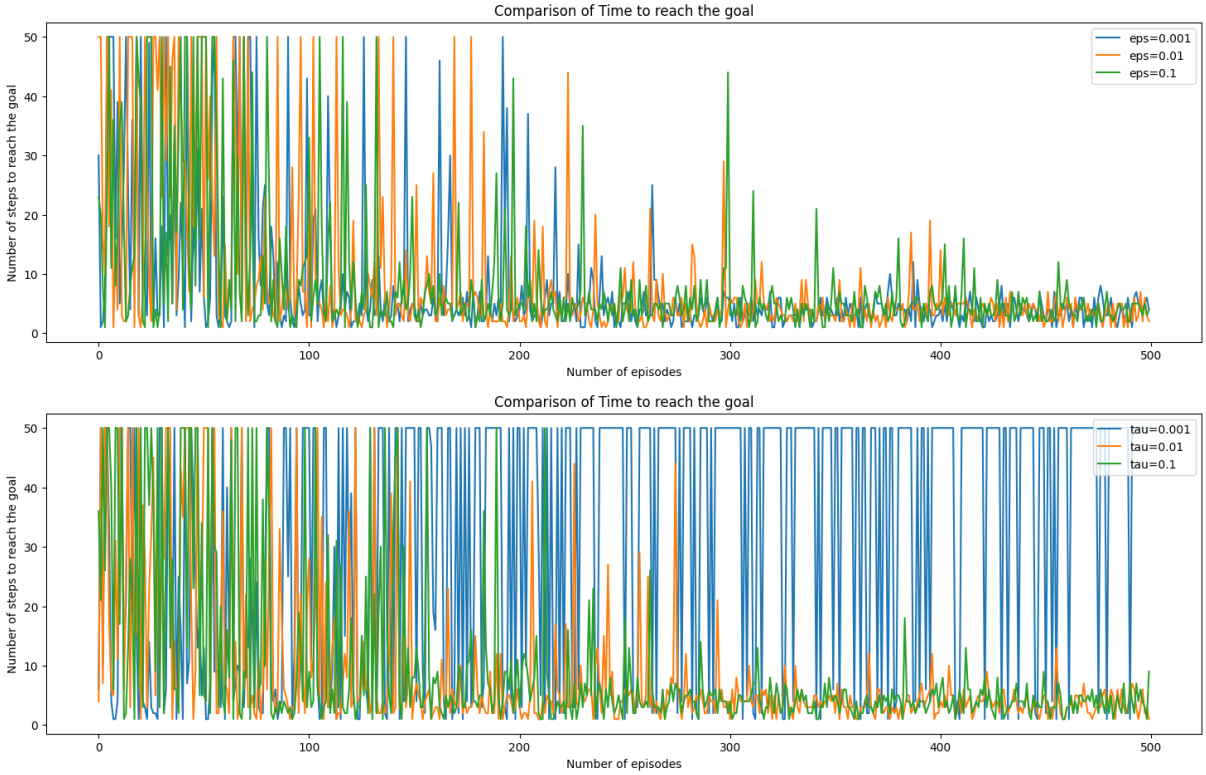

FIGURE 3.4 – Comparatifs des différentes valeurs pour ϵ et τ

Sur ces graphiques, il semblerait dans les deux cas qu'une valeur de 0.1 pour les paramètres accélérerait la vitesse d'apprentissage de Q-Learning en multi-objectifs. Cependant, de la même manière que précédemment, il est difficile d'affirmer qu'une des valeurs des paramètres est fondamentalement meilleure que les deux autres dans un contexte de tirage aléatoire de l'objectif. Les courbes ne sont jamais les mêmes, et l'ordre peut ainsi changer d'un test à l'autre. Nous pouvons voir cependant qu'un τ trop faible ne permet pas un apprentissage correct de la part de l'algorithme.

Nous clôturons cette partie sur l'implémentation de Q-Learning 3D en présentant des politiques calculées par notre algorithme sur le labyrinthe présenté plus haut. Les paramètres des stratégies ont été initialisés à 0.1, et l'agent s'est entraîné sur 500 épisodes. Il est important de noter, que l'agent se déplaçant au départ suivant une marche aléatoire (les valeurs de la Q-table sont à 0), ces résultats ne sont pas comparables et ne sont pas forcément représentatifs de l'efficacité des différentes stratégies.

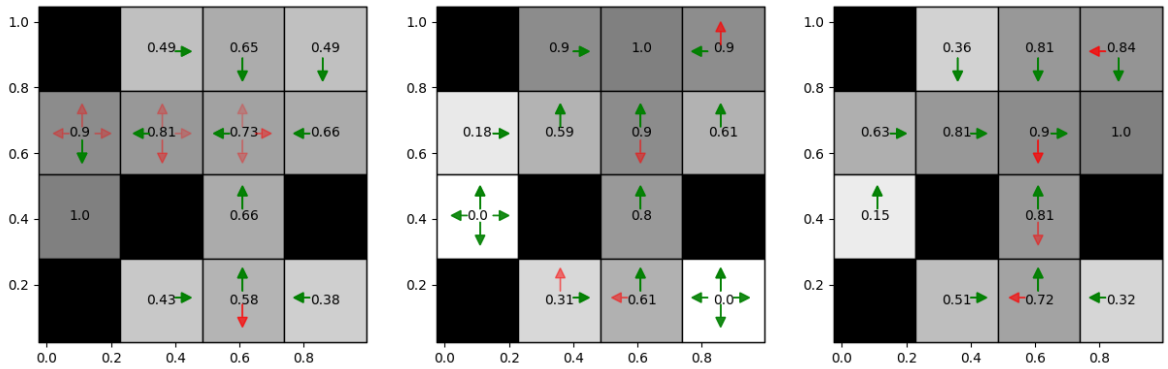

FIGURE 3.5 – Politiques apprises pour buts 1 (gauche), 5 (milieu), 10 (droite) par $\epsilon - greedy$

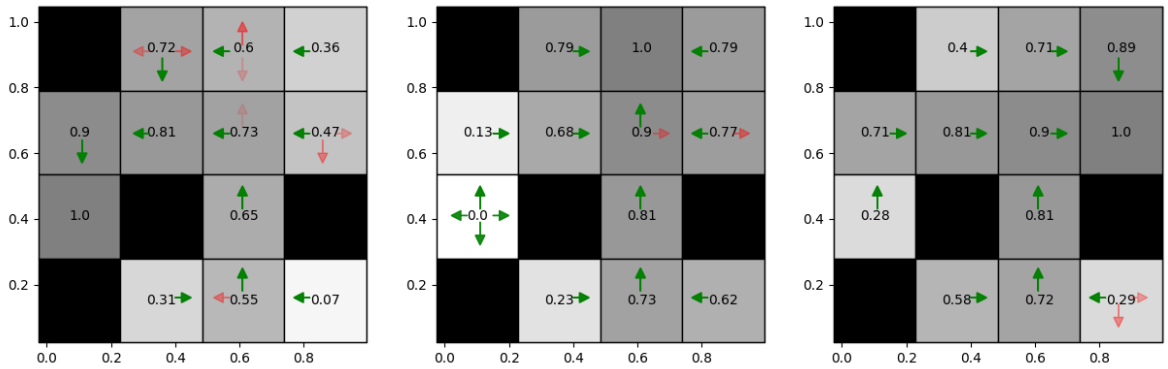

FIGURE 3.6 – Politiques apprises pour buts 1 (gauche), 5 (milieu), 10 (droite) par $softmax$

3.3 HER tabulaire

Notre deuxième objectif a donc été d'implémenter la méthode HER dans notre algorithme de Q-learning en multi-objectifs. Le but étant de tester des implémentations pour ensuite effectuer des comparatifs entre Q-Learning avec et sans HER.

3.3.1 Implémentation

L'objectif de cette partie était donc d'implémenter l'algorithme de Hindsight Experience Replay tel qu'il a été décrit en Section 2.4, pour le rendre utilisable avec le Q-Learning. Nous devions donc définir notre Replay Buffer, et implémenter les différentes boucles permettant le bon déroulé de l'algorithme.

Cette première boucle représente le déroulé d'un épisode. Elle permet à l'agent de se déplacer et de choisir ses prochaines actions. Dans le code présenté, c'est la stratégie $\epsilon - greedy$ qui a été implémentée. C'est dans cette boucle que l'on stocke chacune des transitions dans le replay buffer rb.

```
### boucle de l'épisode déroulée de manière normale
while not done:
    # ajout de l'état actuel au Buffer du chemin
```

```
    sb.append(s)

    # Draw an action using an epsilon-greedy policy
    a = egreedy(Q[:, but, :], s, epsilon)

    # ajout de l'action effectuée au Buffer des actions
    ab.append(a)

    # Perform a step of the MDP
    [s_prime, r, done, _] = mdp.step(a)

    # Replay Buffer pour enregistrer les transitions : tuple (s, a, r ,
       s', but)
    rb.append( (s, a, r, s_prime, but) )

    # Update the agent position
    s = s_prime
```

C'est dans cette deuxième boucle que l'on effectue l'action de relabelling. Cette action représente le fait de modifier le but de l'épisode par un état atteint lors de la marche. Les buts considérés sont tirés au hasard, et l'action est effectuée de nouveau pour pouvoir ainsi récupérer la nouvelle récompense.

```
### boucle de l'expérience replay sur chaque step
for t in range(mdp.mdp.timestep -1):
    # choix des buts annexes parmi le chemin parcouru
    nb_buts = min(NB_BUTS_MAX, len(sb))
    goal = random.sample(sb, nb_buts)

    # récupération des états et de l'action de la transition au temps t
    s = sb[t]
    a = ab[t]

    # pour chaque but annexe, ou ajoute la transition au Replay Buffer
    for g in goal:
        maj_goal(mdp, g, r_list, P_list)    # maj du but

        mdp.mdp.current_state = s     # placement de l'agent dans l'état s
        [s_prime, r, _, _] = mdp.step(a)      # calcul de la nouvelle
           récompense

        rb.append( (s, a, r, s_prime, g) )      # ajout au Buffer
```

Dans cette dernière boucle, nous tirons le mini-batch de transitions, et c'est sur ces transition que nous allons effectuer les calculs permettant la mise à jour de la Q-Table. Nous constatons, comme expliqué dans l'état de l'art, que la Q-Table n'est modifiée qu'à ce moment de l'algorithme. Ce n'est pas le chemin parcouru par l'agent sur l'instant qui compte, mais les potentielles récompenses de chacun de ses pas indépendamment de son objectif de base.

```
### boucle du calcul des nouvelles valeurs de Q par HER
for _ in range(NB_REPLAY):
    # tirage du mini batch parmi les transitions du Replay Buffer
    taille_batch = min(TAILLE_BATCH_MAX, len(rb))
```

```
rb_batch = random.sample(rb, taille_batch)

# pour chaque transiton du mini batch
for  i in range(taille_batch):
    # récupération des données de la transition
    s, a, r, s_prime, but = rb_batch[i]

    # calcul des nouvelles valeurs de Q
    delta = r + mdp.gamma * np.max(Q[s_prime, but]) - Q[s, but, a]
    Q[s, but, a] = Q[s, but, a] + alpha * delta
```

3.3.2 Expérimentations

L'environnement dans lequel se déplace notre agent est le même que celui présenté dans la Figure 3.1. Assez naturellement, un épisode effectué par HER est plus long encore que par Q-Learning multi-objectifs. En effet, en plus du temps de marche habituel, la répétition d'expérience passée augmente le nombre de pas, et donc ralenti le processus. De la même manière que dans la section précédente, nous avons étudié l'influence de la stratégie de choix d'action et de leurs paramètres. Les hyper-paramètres montrés dans la sous-section précédente sont fixés pour le moment, avec 5 le nombre de replay effectué par épisode, 15 la taille du mini-batch tiré parmi les transitions du replay buffer, et 10 la taille du tirage des états traversés.

Nous avons regardé dans un premier temps :

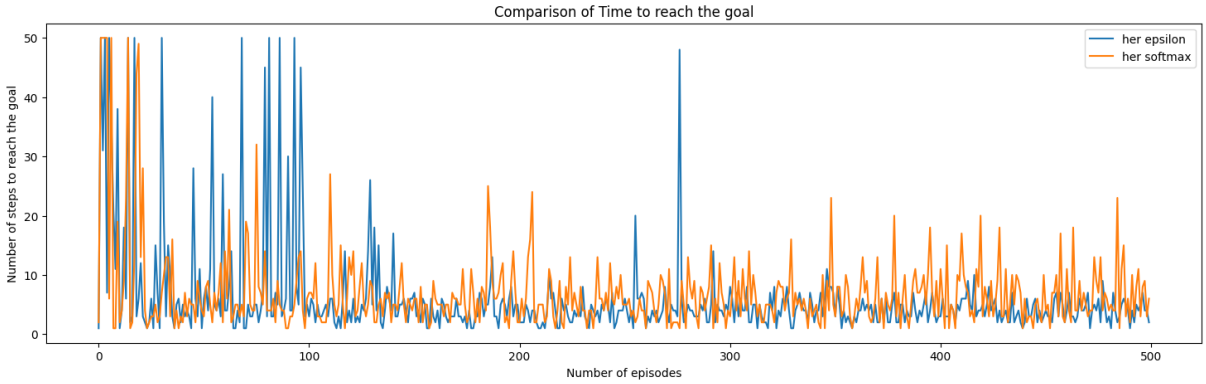

FIGURE 3.7 – Comparatif entre $\epsilon - greedy$ et $softmax$

Contrairement aux tests de Q-Learning simple pour lesquels il était difficile de déterminer, nous pouvons ici constater que $\epsilon - greedy$ semble être la meilleure option pour optimiser l'apprentissage par HER. En effet, les oscillations de $softmax$ restent très prononcées malgré l'avancé du nombre d'épisodes. $\epsilon - greedy$ au contraire varie beaucoup moins à partir du 150^e épisode. Cela signifie qu'une politique se dessine et propose des chemins plus courts. La différence entre les hauteurs d'oscillations étant relativement faible, il serait bon de ne pas en dégager une généralité.

Nous avons ensuite comparé l'apprentissage suivant les différentes stratégies en fonction de leurs paramètres, et obtenu ainsi les graphes suivants :

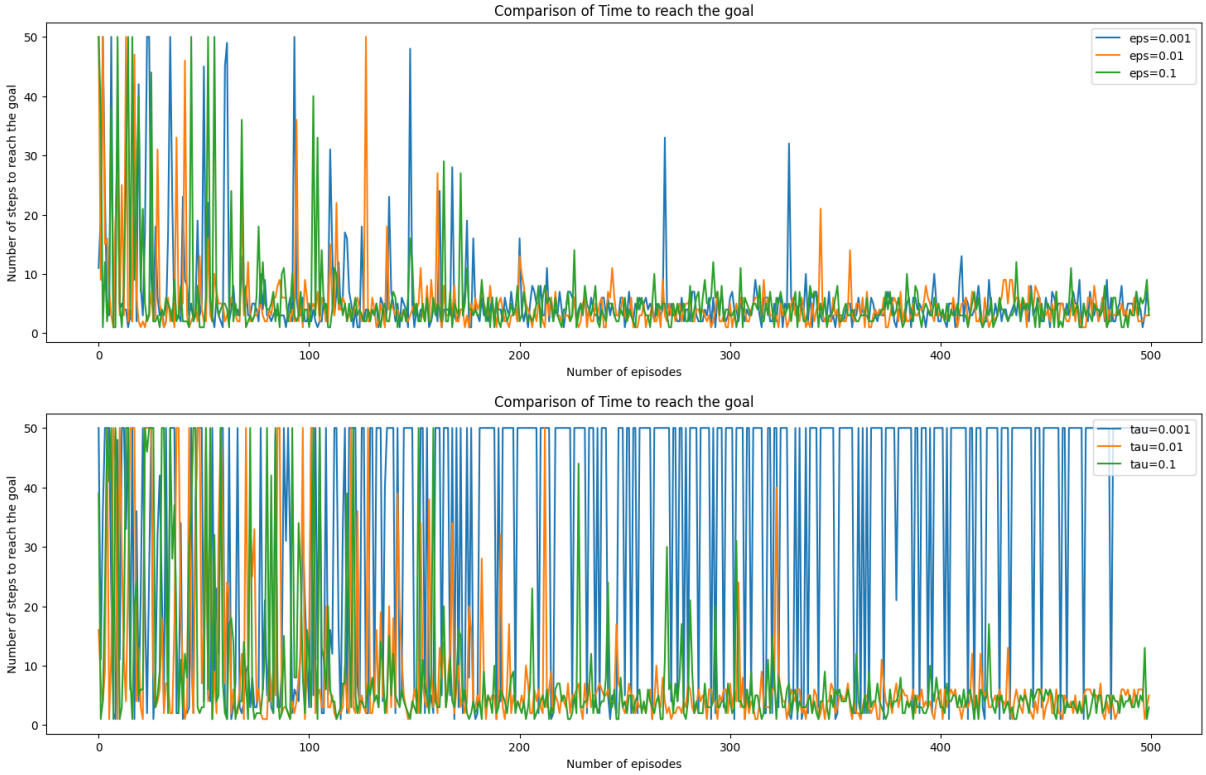

FIGURE 3.8 – Comparatifs des différentes valeurs pour ϵ et τ

Nous constatons de manière assez rapide que $\tau = 0.001$ empêche un apprentissage correct, de la même manière qu'avec Q-Learning. Pour ce qui est des deux autres valeurs de la stratégie $softmax$, il est difficile de les départager par ce graphique. Il semblerait que la courbe verte se stabilise plus rapidement que la orange, mais nous ne pouvons pas distinguer ici de différence notable. Concernant les valeurs de ϵ, il est compliqué de les départager. Même si la valeur la plus élevée semble montrer de plus petites oscillations plus rapidement, ce n'est pas suffisant pour démontrer la préférence d'une valeur sur les autres.

Pour terminer, nous présentons encore une fois une partie des politiques calculées par HER. Nous voyons que malgré l'efficacité de l'algorithme, l'agent n'est pas parvenu à exprimer une politique complète pour chacun des objectifs possibles. Cela s'explique par le tirage aléatoire des buts et des transitions du Replay Buffer. Un état par lequel l'agent est peu passé n'aura pas beaucoup de représentation possible, et il est alors difficile d'exprimer une politique adéquate.

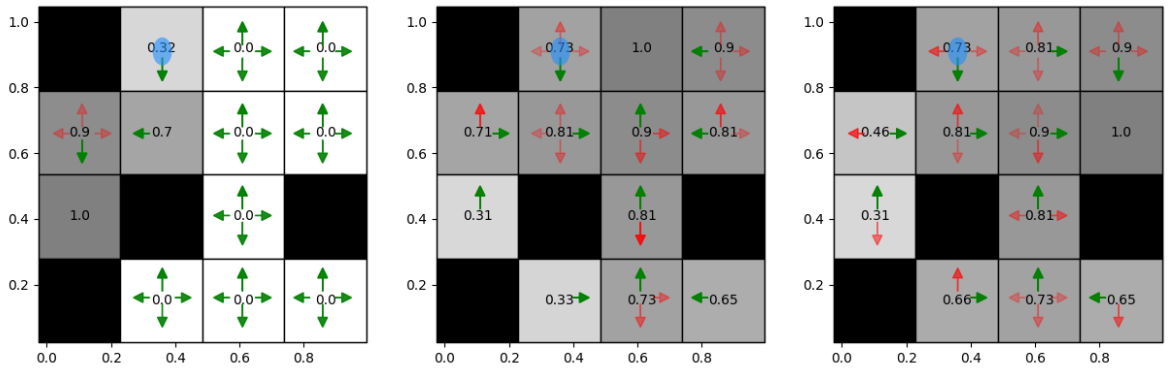

FIGURE 3.9 – Politiques apprises pour buts 1 (gauche), 5 (milieu), 10 (droite) par $\epsilon - greedy$

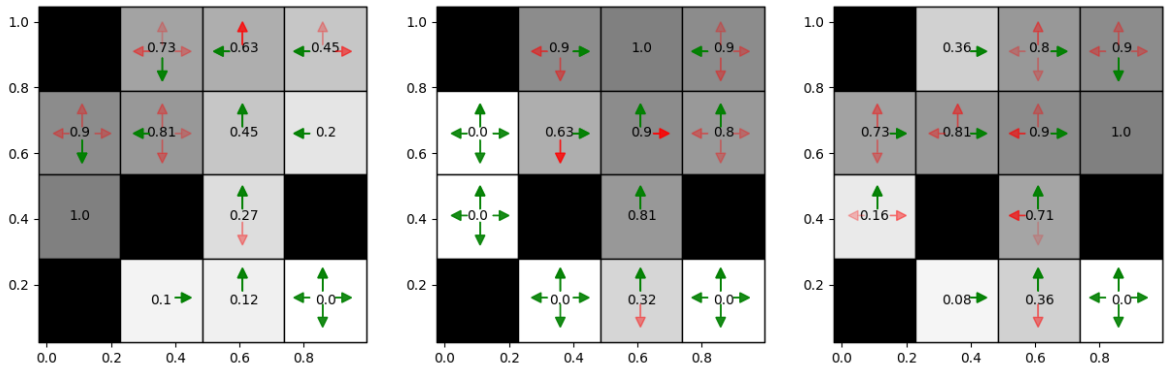

FIGURE 3.10 – Politiques apprises pour buts 1 (gauche), 5 (milieu), 10 (droite) par $softmax$

3.4 Comparaisons Q-Learning/HER

Après avoir étudié et vérifié que nos implémentations de Q-Learning multi-objectifs et de Hindsight Experience Replay sont fonctionnelles et efficaces, nous avons cherché à étudier les différences de vitesse d'apprentissages. Pour ce faire, nous avons comparé les résultats de ces deux algorithmes d'apprentissage sur $\epsilon - greedy$ et $softmax$. Nous fixons ainsi les paramètres de ces stratégies à 0.1, et nous laissons l'agent apprendre sur une durée de 500 épisodes.

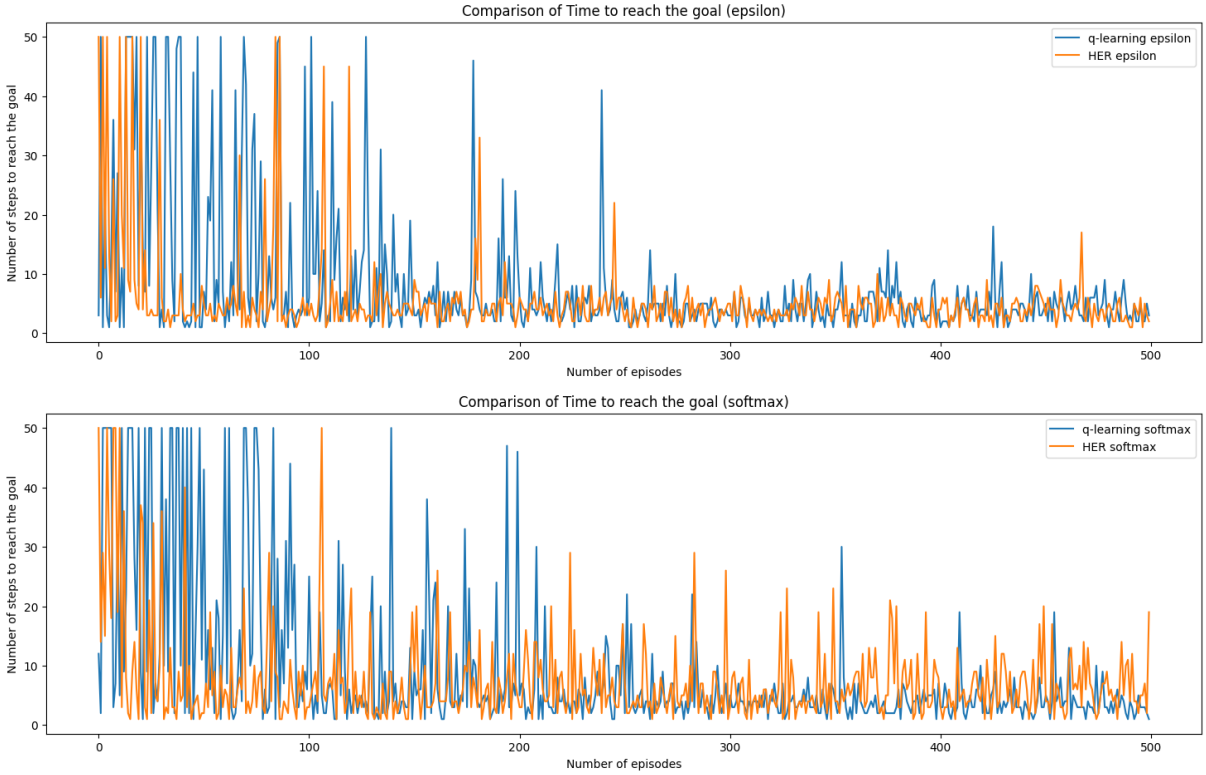

FIGURE 3.11 – Comparatifs Q-Learning/HER par stratégie de choix d'actions

Bien que les courbes et oscillations soient proches dans les deux cas, nous pouvons constater un écart pour la stratégie $softmax$. Il semblerait en effet que Q-Learning offre de meilleurs résultats avec les paramètres que nous avons fixés. Même si la courbe de HER se stabilise, la politique qui découle de l'apprentissage fait prendre à l'agent des chemins plus long. Il pourrait être intéressant d'étudier cette évolution lorsque l'on modifie le nombre de transitions répétées ou le nombre de buts ajoutés dans le replay buffer. Concernant la stratégie $\epsilon - greedy$, HER semble être sensiblement meilleur, mais la différence n'est pas vraiment exploitable dans notre analyse.

3.5 HER dans BBRL

Notre troisième objectif est d'implémenter la méthode HER (Hindsight Experience Replay) au problème CartPole. Dans ce problème, l'agent intelligent doit apprendre à équilibrer la barre supérieure en déplaçant le chariot inférieur. Il s'agit d'un problème classique d'apprentissage par renforcement utilisé pour tester et démontrer l'efficacité de divers algorithmes d'apprentissage par renforcement. Pour adapter l'environnement CartPole à GCRL, nous avons redéfini l'objectif de l'environnement de manière à maintenir la barre en équilibre sur une coordonnée x spécifique. Uniquement lorsque le chariot est à proximité de l'objectif, il reçoit une récompense de 1.

3.5.1 Application de HER à CartPole

La méthode HER (Hindsight Experience Replay) est utilisée pour résoudre les problèmes de récompenses éparses. Dans cette méthode, si le chariot ne parvient pas à atteindre la coordonnée x spécifiée, il reçoit continuellement une récompense nulle, ce qui n'aide pas à

l'entraînement de l'agent. Nous avons donc changé l'objectif et recalculeé les récompenses le long de la trajectoire, de manière à ce que la politique puisse tirer des informations utiles de ces expériences d'échec.

Dans ce code, nous avons créé quatre classes d'agents spécifiques : **GoalAgent**, **RewardAgent**, **HerAgent**, chacune héritant de la classe **Agent** de base. Parmi ces classes :

- GoalAgent est chargé de génère un objectif à chaque début de épisode et l'écrit dans le workspace.
- RewardAgent compare l'état actuel avec l'objectif, calcule la récompense et l'écrit dans le workspace. Dans l'environnement CartPole, nous vérifions si la coordonnée x du chariot est proche de l'objectif.
- HerAgent est une classe de base qui définit comment choisir de nouveaux objectifs dans la méthode HER. Dans la pratique, nous n'utilisons pas directement cette classe, mais plutôt ses sous-classes spécifiques.
    - HerFinal est une classe héritant de HerAgent. Dans HerFinal, nous modifions l'objectif pour chaque trajectoire en utilisant le dernier état de la trajectoire.

La classe **GCQAgent** est un modèle de réseau neuronal pour l'estimation de la fonction de valeur état-action, qui contient un perceptron multicouche (MLP).

3.5.2 Procédure Expérimentale

Au début de l'expérience, l'environnement est créé et les différents agents ainsi que leurs paramètres associés ont été initialisés en fonction de l'environnement. Ensuite, la boucle d'apprentissage se lance et, à chaque étape de l'apprentissage, une série d'actions sont effectuées dans l'environnement à l'aide des agents disponibles. Les résultats sont calculés et enregistrés. Ces résultats sont stockés dans une mémoire tampon de relecture de l'expérience (Replay Buffer) en vue d'une utilisation lors d'une formation ultérieure. Un lot d'expériences est alors tiré de la mémoire tampon, et l'agent est formé avec ces expériences, mettant à jour sa fonction de valeur d'état-action prédite. Ce processus se poursuit jusqu'à ce que certaines conditions d'arrêt soient remplies.

En outre, cette procédure comporte une composante d'évaluation, dans laquelle l'agent est exécuté dans un environnement distinct à intervalles réguliers et ses performances sont enregistrées.

3.5.3 Résultats et Analyse

Dans notre évaluation, nous utilisons quatre environnements et calculons la moyenne des récompenses cumulatives. La figure 3.12 montre les résultats obtenus avec l'algorithme HER, tandis que la figure 3.13 montre les résultats sans l'utilisation de l'algorithme HER. Par le biais de cette expérience comparative, nous pouvons observer une amélioration significative des performances lors de l'utilisation de l'algorithme HER. L'avantage principal de l'algorithme HER réside dans le fait qu'il rend les signaux de récompense plus denses en ré-attribuant les objectifs des trajectoires contenues dans l'historiques. Ainsi, il permet d'apprendre à partir des données qui étaient initialement des échecs afin de réussir les "nouvelles tâches", ce qui améliore la stabilité de l'apprentissage et l'efficacité des échantillons.

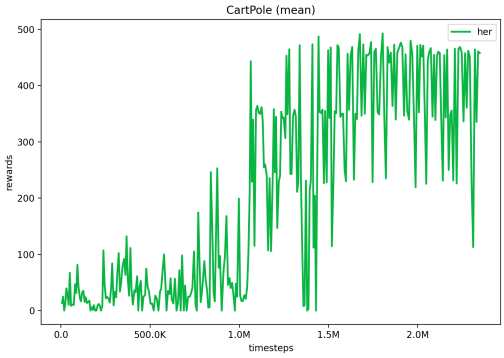

FIGURE 3.12 – HER

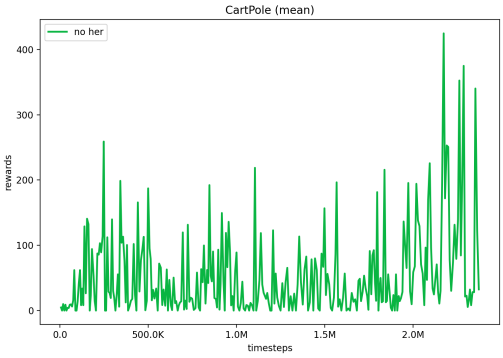

FIGURE 3.13 – Sans HER

3.6 Difficultés

L'implémentation de l'algorithme de Hindsight Experience Replay a été laborieuse et nous a amené plusieurs difficultés, notamment au niveau l'intégration à la bibliothèque BBRL. Cette librairie étant dense, nous avons dû passer plusieurs jours à étudier en détail le fonctionnement, les enjeux et l'architecture de cette librairie, pour nous permettre de nous en inspirer et d'en utiliser correctement les concepts. Initialement notre phase d'expérimentation devait se faire uniquement dans un environnement de type labyrinthe, modifié pour correspondre à un environnement continu. Cependant l'astuce faisant passer l'environnement d'un espace d'états finis à un espace d'états continus, ainsi que l'architecture des Workspaces ont rendu difficile les modifications ajoutées par notre phase de développement.

Nous nous sommes donc approchés de l'environnement CartPole de Gym, prévu dès l'origine avec un espace des états continus, et s'adaptant plus facilement à nos objectifs. Nous avons malgré tout eu plusieurs difficultés dans l'implémentation de nos agents, ainsi que dans le choix de l'utilisation des paramètres du Workspace.

Chapitre 4

Conclusion

Dans ce projet, nous avons présenté et implémenté en détail l'algorithme de Hindsight Experience Replay (HER). Nous avons réalisé de multiples expériences dans plusieurs environnements, pour évaluer les performances de cette méthode dans différentes conditions, et avons effectué une comparaison et une analyse détaillées de nos résultats.

Grâce à notre travail, nous avons également appris comment mettre en œuvre le Q-Learning dans des environnements avec plusieurs objectifs. De plus, nous avons examiné en détail les différences de performances entre le Q-Learning et le HER dans différentes configurations de stratégies et d'environnements. Nos résultats expérimentaux indiquent que le HER performe souvent mieux que le Q-Learning seul lorsqu'il s'agit de résoudre des tâches difficiles. Cependant même si nos résultats sur CartPole appuient cette idée pour un espace d'états continu, il est difficile d'en déduire en réelle généralité dans un espace fini avec nos seules expériences.

Cependant, notre travail présente également certaines limitations. Tout d'abord, nos expériences reposent principalement sur des environnements spécifiques et des paramètres spécifiques. De ce fait, notre implémentation peut ne pas être applicable à tous les problèmes et environnements. De plus, bien que le HER ait montré des performances supérieures dans certains cas, des ressources de calcul plus importantes et un temps d'entraînement plus long peuvent être nécessaires dans des environnements complexes.

Pour l'avenir, nous pensons qu'il y a encore de nombreuses pistes de recherche à explorer. Tout d'abord, nous pourrions ajouter davantage de stratégies pour la sélection de nouveaux objectifs dans le HER. Ensuite, il serait intéressant d'envisager une combinaison du HER avec d'autres algorithmes d'apprentissage par renforcement off-policy tels que les méthodes Actor-Critic. De plus, pour le Q-Learning, nous pouvons étudier des stratégies plus avancées telles que l'ajustement dynamique de ϵ pour une meilleure exploration et exploitation. Enfin, il serait bénéfique de tester ces algorithmes dans des environnements plus variés et des tâches plus complexes afin d'évaluer davantage leurs performances et leur adaptabilité.

Bibliographie

[1] Richard S Sutton et Andrew G Barto. *Reinforcement learning : An introduction*. MIT press, 2018 (page 2).

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski et al. « Human-level control through deep reinforcement learning ». In : *nature* 518.7540 (2015), p. 529-533 (pages 3, 5).

[3] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar et David Silver. « Rainbow : Combining improvements in deep reinforcement learning ». In : *Proceedings of the AAAI conference on artificial intelligence*. T. 32. 1. 2018 (page 3).

[4] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton et al. « Mastering the game of go without human knowledge ». In : *nature* 550.7676 (2017), p. 354-359 (page 3).

[5] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever et al. « Improving language understanding by generative pre-training ». In : (2018) (page 3).

[6] Christopher John Cornish Hellaby Watkins. « Learning from delayed rewards ». In : (1989) (page 3).

[7] Hado Hasselt. « Double Q-learning ». In : *Advances in neural information processing systems* 23 (2010) (page 4).

[8] Minghuan Liu, Menghui Zhu et Weinan Zhang. « Goal-conditioned reinforcement learning : Problems and solutions ». In : *arXiv preprint arXiv :2201.08299* (2022) (page 5).

[9] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel et Wojciech Zaremba. « Hindsight experience replay ». In : *Advances in neural information processing systems* 30 (2017) (page 5).

[10] Shangtong Zhang et Richard S Sutton. « A deeper look at experience replay ». In : *arXiv preprint arXiv :1712.01275* (2017) (page 7).

[11] Melrose Roderick, James MacGlashan et Stefanie Tellex. « Implementing the deep q-network ». In : *arXiv preprint arXiv :1711.07478* (2017) (page 7).

[12] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang et Wojciech Zaremba. « Openai gym ». In : *arXiv preprint arXiv :1606.01540* (2016) (page 8).

[13] Ludovic Denoyer, Alfredo De la Fuente, Song Duong, Jean-Baptiste Gaya, Pierre-Alexandre Kamienny et Daniel H Thompson. « Salina : Sequential learning of agents ». In : *arXiv preprint arXiv :2110.07910* (2021) (page 8).

Annexe A

Cahier des charges

A.1 Contexte et Objectifs

L'objectif de notre projet est d'étudier différentes méthodes d'apprentissage par renforcement. En commençant par des méthodes tabulaires simples (Q-Learning par états-actions), nous devons étendre nos codes et algorithmes pour nous permettre d'implémenter et d'étudier des techniques plus poussées. Le principal algorithme autour duquel nous allons centrer nos efforts se nomme Hindsight Expérience Replay, que nous tenterons d'implémenter suivant différents modèles. Les tests de nos programmes et de nos algorithmes se feront par la recherche par un agent d'un chemin vers un (ou plusieurs) but(s) dans un labyrinthe. Nos labyrinthes seront générés aléatoirement par un module externe, et seront représentés sous forme de quadrillages. Ils pourront également contenir des murs, pour bloquer et complexifier les chemins menant à lobjectif. Nous cherchons ainsi à apprendre la meilleure politique de déplacement pour notre agent au sein de cet environnement.

A.2 Besoins

Une grande partie des algorithmes sur lesquels nous devons nous attarder ne nous ont jamais été présentés pendant notre parcours. Pour nous permettre une bonne implémentation et utilisation de ces différents algorithmes, nous devons en premier lieu les étudier attentivement. Nous avons pour ce faire lu avec attention un certain nombre d'articles qui nous étaient fournis ou que nous avons dû trouver par nous-mêmes. La plupart de ces articles ont par ailleurs été présentés dans notre carnet de bord.

Une autre étude indispensable a été faite sur de multiples librairies de Python. En effet, nous utiliserons à de multiples reprises des modules proposés par Python et d'autres développeurs externes, que nous devons comprendre pour nous assurer une utilisation correcte. Les principales librairies sont numpy pour le travail sur tableau et PyTorch pour l'utilisation de réseaux de neurones, mais aussi Gym proposée par OpenAI ainsi que BBRL développée par des chercheurs de l'université, et sur laquelle nous devrons nous appuyer pour nos futures implémentations.

A.3 Approches

A.3.1 Q-Learning

Le Q-learning est une méthode dapprentissage par renforcement, qui tire son nom de la fonction Q utilisée pour le calcul de récompense dune action exécutée dans un état donné. Cette fonction calcule donc le gain potentiel de cette action, et permet de supposer à chaque instant quelle est la meilleure action à prendre pour maximiser la récompense totale. Ce gain potentiel est stocké dans une Q-table. La phase de calcul s'effectue de manière assez simple. On considère tout d'abord une récompense de valeur 1 sur l'état but, et de 0 partout ailleurs. Lorsque ce but est atteint, la récompense est rétro-propagée sur le chemin, en fonction de deux valeurs : le facteur d'apprentissage et le facteur d'actualisation. Le premier facteur détermine à quel point la nouvelle information calculée va remplacer la précédente. Le second détermine l'importance des récompenses à venir dans le calcul de la récompense actuelle. C'est également ce gain qui nous aide à choisir l'action à effectuer dans chaque état.

Le choix de laction optimale peut se faire par plusieurs stratégies, et nous en considérerons deux en particulier : $\epsilon - greedy$ et $softmax$. La première permet de choisir avec une forte probabilité la meilleure action proposée par la Q-table, ou parmi les autres actions de façon équitable (mais avec une très faible probabilité). La seconde permet de choisir parmi les actions selon des probabilités proportionnelles à leurs gains potentiels. Ainsi, une action avec un gain moyen aura plus de chance dêtre tirée quune action avec un faible gain, mais moins de chance quune action avec un gain plus élevé. Le choix de laction dépendant des probabilités, il est intéressant de noter que ces méthodes convergent sur le long terme vers une politique de déplacement optimale.

Dans notre cas, nous devons considérer deux versions du Q-Learning. Dans un premier temps, nous devrons étudier une représentation en 2 dimensions, pour le cas dun seul objectif à atteindre. Notre Q-table doit donc prendre en entrée deux valeurs, létat courant et laction envisagée, pour en calculer la récompense. Dans un second temps, nous devrons nous pencher sur une représentation en 3 dimensions, dans un contexte multi-objectifs. La récompense ne dépend donc plus que de létat et de laction, mais également du but à atteindre dans l'environnement. Nous considérons dans ce cas là chaque état atteignable comme un potentiel but. Le temps de travail pour lalgorithme avant de trouver une politique convenable étant décuplé sous cette représentation, nous travaillerons sur de plus petits labyrinthes.

A.3.2 HER

Hindsight Experience Replay est un algorithme ayant pour objectif d'accélérer la découverte d'une politique optimale dans un contexte multi-objectifs. Le principe de base est simple : lorsque l'on ne parvient pas à atteindre le but défini à l'origine, on considère alors l'état final comme s'il était notre objectif, et l'on recalcule alors nos récompenses en considérant ce nouvel objectif. Les récompenses ne sont donc plus calculées pendant la phase de marche, mais une fois l'épisode fini.

Pour nous permettre de calculer ces nouvelles valeurs à posteriori, il faut stocker les transitions de la marche. Nous utilisons donc pour cela un Replay Buffer, dans lequel nous stockons à chaque pas les données nous permettant le calcul (état de départ, ac-

tion effectuée, état d'arrivée, récompense, et but considéré). Aux transitions stockées de cette manière, nous allons ajouter de nouvelles transitions à partir de ces dernières, auxquelles nous allons modifier le but considéré pour le remplacer par l'état atteint. Nous re-traitons ensuite ces étapes pour ainsi calculer leurs nouvelles récompenses associées. Ainsi, n'importe quelle séquence d'actions dans l'environnement permet d'améliorer la politique de déplacement. C'est cette approche que nous allons tenter d'implémenter en premier lieu.

Une version plus complète de cet algorithme consiste, en plus de considérer l'état final, à étudier également les états traversés pendant la marche. De façon aléatoire, nous sélectionnons un ensemble d'états que nous utiliserons pour remplacer sur chacune des transitions le but considéré. Ensuite, nous tirons au hasard des transitions parmi l'entièreté des valeurs stockées, et c'est sur ce sous-ensemble que nous recalculons les récompenses. Cela permet d'améliorer la politique pour les états traversés par la marche. Ainsi, une seule marche peut permettre de calculer des gains potentiels pour plusieurs états, actions, et buts considérés. Nous devrions ainsi avoir besoin de moins de trajets pour obtenir une table des récompenses convenable. Le nombre de calculs effectués risque cependant d'être plus long et peut en effet rallonger la durée d'exécution de notre algorithme. Une fois une bonne compréhension du principe de HER, c'est cette version que nous tenterons d'implémenter.

A.3.3 Comparaisons

Notre objectif dans cette première partie de projet est de nous permettre de comparer entre elles les différentes approches grâce à notre implémentation. Comme annoncé en introduction, nous testerons notre algorithme dans l'optique de recherche de politique de déplacement conditionné par les buts, dans un environnement de type labyrinthe. Cet environnement sera dans un premier temps traité comme un environnement discret. C'est-à-dire que chaque état sera représenté par une case numérotée, de même que les actions discréditées selon les quatre points cardinaux.

Notre application se fera méthodiquement. Nous utiliserons plusieurs hyper-paramètres pour nous permettre d'étudier nos algorithmes. Dans un premier temps nous pourrions modifier la topologie de notre labyrinthe. Un paramètre intéressant est le ratio du nombre de murs, c'est-à-dire de cases infranchissables. Il nous est ainsi possible de créer des labyrinthes imparfaits ou possédant des entonnoirs, ou d'autres topologies spécifiques intéressantes pour l'évaluation de nos algorithmes.

Dans le gros de notre étude cependant, nous tenterons d'observer l'influence des paramètres des algorithmes en eux-mêmes. Le nombres d'épisodes correspond aux nombres de marches différentes effectuées dans le labyrinthe. Le nombre de steps correspond au nombre de pas maximum effectués au sein d'un épisode. En effet l'épisode continuerait tant que le but n'est pas atteint, pour ne pas prendre le risque d'un épisode infini, nous définissons une valeur limite. Nous pouvons également nous intéresser à la taille de l'échantillon considéré pour le recalcul des récompenses, choisis à partir des transitions du Replay Buffer.

A.4 Ouverture

Notre premier objectif est donc de comprendre et développer ces approches et algorithmes, pour nous permettre de les comparer et d'étudier leurs différences tant par leur exécution que par leurs résultats. Une fois cette première étape terminée et maîtrisée, une ouverture nous est proposée.

En nous basant sur BBRL, une librairie python existante et développée par des enseignants-chercheurs du laboratoire de l'ISIR, nous devrons implémenter les algorithmes présentés précédemment dans un environnement cette fois continu. Pour ce faire, nous n'utiliserons plus un tableau à trois entrées, mais un réseau de neurones prenant en entrée l'état actuel et le but espéré, et nous renvoyant les gains potentiels des différentes actions, nous permettant ensuite de choisir l'action adaptée. Dans un premier temps nous devrons donc prendre en main la librairie pour une assurer par la suite une correcte utilisation et implémentation des algorithmes.

Annexe B

Manuel utilisateur

B.1 Q-Learning en 3 dimensions

Le fichier `QL_HER_3D.ipynb` explore la mise en uvre et l'application de Q-Learning en trois dimensions, y compris la configuration d'un environnement spécifique, l'importation des modules nécessaires, la création d'un environnement de labyrinthe et l'utilisation d'une série de fonctions utilitaires. Deux versions de Q-Learning et Hindsight Experience Replay (HER) sont mises en uvre, à savoir les stratégies $\epsilon - greedy$ et $softmax$. Les résultats de ces méthodes sont visualisés à l'aide de courbes d'apprentissage, montrant le nombre d'étapes nécessaires pour atteindre l'objectif au cours du processus de formation. Enfin, elle compare les performances de Q-Learning et HER, fournissant des outils et des méthodes pour comprendre et optimiser ces deux stratégies.

B.1.1 Dépendances

Suivez les étapes ci-dessous pour installer et configurer le projet :

1. Installez les versions spécifiques de setuptools et wheel :
   `pip install setuptools==66 wheel==0.38.4`
2. Installez le dépôt bbrl_gym :
   `pip install git+https://github.com/osigaud/bbrl_gym`
3. Si l'installation de Box2D échoue, procédez à l'installation manuelle de SWIG en utilisant les instructions appropriées.
   `pip install swig`

B.1.2 Import

Ensuite, utilisez le code de la section import pour importer les modules nécessaire.

B.1.3 Création du Labyrinthe

On crée un environnement de labyrinthe en utilisant l'environnement MazeMDP. Les paramètres de l'environnement sont définis dans les paramètres de configuration :

```
MAZE_LARGEUR = 4
MAZE_HAUTEUR = 4
RATIO = 0.25
env = gym.make("MazeMDP-v0", kwargs={"width": MAZE_LARGEUR, "height":
    MAZE_HAUTEUR, "ratio": RATIO})
env.reset()
env.init_draw("The maze")
```

Ces paramètres définissent la largeur et la hauteur du labyrinthe, ainsi que le ratio d'obstacles.

B.1.4 Utils

Les fonctions utilitaires incluent :

1. `get_policy_from_q(Q: np.ndarray, but: int) : np.ndarray` : Cette fonction renvoie la meilleure politique pour un but donné.
2. `calcul_transition(mdp: MazeMDPEnv, but)` : Cette fonction calcule la matrice de transition adéquate pour un but donné.
3. `calcul_goal(mdp: MazeMDPEnv, but: int)` : Cette fonction calcule et stocke les matrices de récompenses et de transitions pour un but donné.
4. `maj_goal(mdp: MazeMDPEnv, but, r_list, P_list)` : Cette fonction modifie les paramètres du MazeMDP en fonction du goal.

B.1.5 Q-Learning 3D

Il y a deux versions du Q-Learning 3D : l'une utilisant $\epsilon - greedy$ et $softmax$.

1. ϵ-greedy : utilise une stratégie $\epsilon - greedy$ pour l'exploration. L'implémentation est effectuée par la fonction `ql_3d_eps()`.
2. softmax : utilise une stratégie $softmax$ pour l'exploration. L'implémentation est effectuée par la fonction `ql_3d_soft()`.

En plus des deux versions décrites ci-dessus, il existe une fonction supplémentaire appelée `q_learning_eps_order()`. Cette fonction implémente une version ordonnée de l'algorithme Q-Learning avec une stratégie $\epsilon - greedy$.

B.1.6 Hindsight Experience Replay 3D

Comme pour Q-Learning 3D, il y a deux versions de HER Q-Learning 3D : $\epsilon - greedy$ et $softmax$.

1. ϵ-greedy : utilise une stratégie $\epsilon - greedy$ pour l'exploration. L'implémentation est effectuée par la fonction `her_ql_3d_eps()`.
2. softmax : utilise une stratégie $softmax$ pour l'exploration. L'implémentation est effectuée par la fonction `her_ql_3d_soft()`.

B.1.7 Visualisation

Pour visualiser les résultats des algorithmes Q-Learning et HER, vous pouvez utiliser le code suivant :

```
    print("état but : ", but)
    env.mdp.plotter.terminal_states = [but]
    env.draw_v_pi(Q[:,but,:], get_policy_from_q(Q, but), title="Q-learning
        e-greedy"+str(but))
```

Ce code affiche l'état objectif, définit cet état comme l'état final pour l'environnement du labyrinthe et affiche ensuite sur chaque case du labyrinthe la fonction de valeur Q et la politique correspondante pour cet état objectif. Le titre du graphique sera "Q-learning e-greedy"+str(but), où but est le numéro de l'état objectif.

B.1.8 Courbe d'apprentissage

Le script fournit plusieurs fonctions pour tracer les courbes d'apprentissage, qui montrent le nombre d'étapes nécessaires pour atteindre l'objectif en fonction du nombre d'épisodes pour différents paramètres et algorithmes.

1. `plot_ql_3d_eps()` : Cette fonction trace les courbes d'apprentissage pour l'algorithme Q-Learning avec une stratégie $\epsilon - greedy$. Différentes valeurs de ϵ peuvent être fournies en entrée pour comparer leurs performances.
2. `plot_ql_3d_tau()` : Cette fonction trace les courbes d'apprentissage pour l'algorithme Q-Learning avec une stratégie $softmax$. Différentes valeurs de τ peuvent être fournies en entrée pour comparer leurs performances.
3. `plot_ql_3d_goal()` : Cette fonction trace les courbes d'apprentissage pour l'algorithme Q-Learning avec les stratégies $\epsilon - greedy$ et $softmax$.
4. `plot_her_3d_eps()` : Cette fonction trace les courbes d'apprentissage pour l'algorithme HER avec une stratégie $\epsilon - greedy$. Différentes valeurs de ϵ peuvent être fournies en entrée pour comparer leurs performances.
5. `plot_her_3d_tau()` : Cette fonction trace les courbes d'apprentissage pour l'algorithme HER avec une stratégie $softmax$. Différentes valeurs de τ peuvent être fournies en entrée pour comparer leurs performances.
6. `plot_her_3d_goal()` : Cette fonction trace les courbes d'apprentissage pour l'algorithme HER avec les stratégies $\epsilon - greedy$ et $softmax$.

Chaque fonction affiche un graphique où l'axe des abscisses représente le nombre d'épisodes et l'axe des ordonnées représente le nombre de pas effectués pour atteindre l'objectif. Les courbes représentent le nombre d'étapes pour atteindre l'objectif pour chaque épisode d'apprentissage.

Exemples d'utilisation :

```
plot_ql_3d_eps(env, [0.001, 0.01, 0.1], 500, TIMEOUT, ALPHA, False)
plot_her_3d_tau(env, [0.001, 0.01, 0.1], 500, TIMEOUT, ALPHA, False)
plot_her_3d_goal(env, EPSILON, TAU, 500, TIMEOUT, ALPHA, False)
```

L'exécution de ces fonctions générera plusieurs graphiques montrant les performances des différentes combinaisons de paramètres et d'algorithmes.

B.1.9 Comparaison entre Q-Learning et Hindsight Experience Replay (HER)

La fonction `plot_ql_her_goal()` est utilisée pour comparer les performances des algorithmes Q-Learning et HER en termes de nombre d'étapes nécessaires pour atteindre l'objectif. Cette fonction génère deux graphiques de courbes d'apprentissage, l'un pour la stratégie $\epsilon - greedy$ et l'autre pour la stratégie $softmax$.

Pour chaque stratégie, elle compare les performances du Q-Learning et de HER en traçant le nombre d'étapes nécessaires pour atteindre l'objectif pour chaque épisode d'apprentissage. L'axe des x représente le nombre d'épisodes et l'axe des y représente le nombre d'étapes pour atteindre l'objectif.

Exemple d'utilisation :

```
plot_ql_her_goal(env, EPSILON, TAU, 500, TIMEOUT, ALPHA, False)
```

B.2 HER dans BBRL

B.2.1 Dépendances

Installez le dépôt `bbrl_examples` :
`pip install git+https://github.com/osigaud/bbrl_examples.git`

B.2.2 Introduction à la structure du code

Configurations

Le fichier `dqn_gc_cartpole.yaml` se trouve dans le répertoire `algo/configs/`. Il contient des configurations suivantes :

```
logger:
  classname: bbrl.utils.logger.TFLogger
  # the directory to save the log file
  log_dir: ./plot/
  verbose: False
  every_n_seconds: 10

algorithm:
  seed: 4
  # number of seeds to run
  nb_seeds: 1
  # epsilon greedy exploration
  epsilon_init: 0.02
  # clip the gradient norm to this value
```

```
    max_grad_norm: 0.5
    # size of the replay buffer
    buffer_size: 1e5
    # number of the training environment
    n_envs: 10
    # batch size of the samples from the replay buffer
    batch_size: 1024
    # number of evaluations
    nb_measures: 300
    # evaluate every n steps
    eval_interval: 6000
    # update the target network every n steps
    target_critic_update: 5000
    # number of the evaluation environments
    nb_evals: 4
    discount_factor: 0.99
    architecture:
      # number of hidden layers of the dqn network
      hidden_size: [256, 256]
    # if use HER
    her: False
    # if render the evaluation environment
    render_eval: False

goal:
    # goal dimension
    goal_dim: 1
    # the range of the goal
    goal_range: [[-1,1]]
    # the type of the goal
    goal_type: float

gym_env:
    classname: __main__.make_gym_env
    env_name: CartPole-v1

optimizer:
    classname: torch.optim.Adam
    lr: 2.3e-3
```

Exécution

Le fichier `dqn_gc_cartpole.py` qui se trouve dans le répertoire `algo` implémente la méthode HER (Hindsight Experience Replay) au problème CartPole. Exécutez ce fichier directement après la configuration.

Voici la description des fonctions de ce fichier :

1. **Importation des modules nécessaires :** Cela comprend les modules système,

pytorch, gym, hydra, etc.,= ainsi que les modules BBRL personnalisés.

2. **Création de l'environnement Gym :** La fonction `make_gym_env` est utilisée pour créer l'environnement Gym.
3. **Calcul de la perte du Critic :** La fonction `compute_critic_loss` est utilisée pour calculer la perte du Critic basée sur la différence temporelle.
4. **Calcul de la récompense :** La fonction `compute_reward` est utilisée pour calculer la récompense.
5. **Création de l'agent DQN :** La fonction `create_dqn_agent` crée tous les agents dont nous avions besoin.
6. **Exécution du DQN :** La fonction `run_dqn` est utilisée pour exécuter la boucle principale de l'algorithme DQN, pour l'entraînement et l'évaluation, et pour enregistrer les données du processus d'entraînement.
7. **Boucle principale :** La fonction `main_loop` est la boucle principale du programme, elle exécute l'algorithme DQN pour chaque graine aléatoire et enregistre la récompense.
8. **Fonction principale :** La fonction main est le point d'entrée du programme, elle utilise d'abord l'outil Hydra pour charger le fichier de configuration, puis elle appelle la fonction `main_loop` pour commencer à exécuter la boucle principale du programme.

Agent

— Le package `algo/GoalConditioned` contient les différents agents utilisés dans le cadre de apprentissage conditionné par des buts.

Voici une brève description de chaque agent :

- `GoalAgent` est chargé de génère un objectif à chaque début de épisode et l'écrit dans le workspace.
- `RewardAgent` compare l'état actuel avec l'objectif, calcule la récompense et l'écrit dans le workspace. Dans l'environnement CartPole, nous vérifions si la coordonnée x du chariot est proche de l'objectif.
- `HerAgent` est une classe de base qui définit comment choisir de nouveaux objectifs dans la méthode HER. Dans la pratique, nous n'utilisons pas directement cette classe, mais plutôt ses sous-classes spécifiques.
  - `HerFinal` est une classe héritant de `HerAgent`. Dans `HerFinal`, nous modifions l'objectif pour chaque trajectoire en utilisant le dernier état de la trajectoire.

— Le package `algo/critics` contient une classe appelée `GCQAgent`, qui est utilisée dans le réseau DQN pour calculer les valeurs Q.