UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

DÉPÔT FINAL

PRÉSENTÉ À

M.FRANÇOIS MEUNIER

DU COURS

PIF6004 : SUJETS SPÉCIAUX EN INFORMATIQUE

PAR

YOUSSEF BARKAOUI ABDELLAH BENKADJA

AVRIL 2021

Table des matières

1 Intérêt et mise en contexte **2**

2 Objectifs **2**

3 Introduction **3**
3.1 A propos de l'apprentissage par renforcement . . . 3
3.2 Bandits Multi-Armés . . . 4
3.3 Terminologie de base de l'apprentissage par renforcement . . . 9
3.4 Taxonomie de l'apprentissage par renforcement . . . 12

4 Application d'apprentissage par renforcement par OpenAI Gym **14**
4.1 OpenAI Gym avec l'environment Frozen-Lake . . . 15
4.2 Programmer un agent pour le problème Frozen Lake avec Q-Learning . . . 15
4.3 Programmer un agent pour Frozen Lake avec Q-Network . . . 17
4.4 Comparison entre Q-Table et Q-Network . . . 18

5 Processus de décision de Markov **18**
5.1 Définitions . . . 18
5.2 Assomptions . . . 19
5.3 Les équations de Bellman . . . 19
5.4 Formation de l'environnement FrozenLake-v0 en utilisant le MDP . . . 21

6 Simulation De Voiture Autonome **21**
6.1 Mise en contexte . . . 21
6.2 Méthodologie . . . 21
6.3 Évaluation et expérimentation . . . 23

7 Conclusion et perspectives **24**

1 Intérêt et mise en contexte

Le domaine de l'apprentissage automatique vise à améliorer la prise de décision. Il trouve son application dans un large éventail de domaine tels que la robotique, la santé, les réseaux intelligents, la finance, les voitures autonomes et bien d'autres.

Le Reinforcement Learning (RL) , que l'on vise d'aborder dans le rapport en cours, est inspirée initialement de la psychologie de comportement. L'idée principale est qu'un agent artificiel peut apprendre en interagissant avec son environnement, de manière similaire à un agent biologique. L'agent artificiel doit être en mesure d'optimiser certains objectifs remis sous forme de récompenses cumulatives. Cette approche s'applique en principe à tout type de problème de prise de décision séquentielle reposant sur des expériences antérieures. L'environnement peut être stochastique, c-à-d, l'agent peut n'observer que des informations partielles sur l'état actuel, et les observations peuvent être de grande dimension (par exemple, des cadres et des séries temporelles).

L'approche de RL est devenue de plus en plus populaire en raison de son succès dans la prise de décision dans un contexte difficile. Plusieurs de ces réalisations sont d'importance majeure dans la littérature grace à la combinaison de RL avec des techniques d'apprentissage profond. Cette combinaison, appelée Deep RL, est la plus utile dans les problèmes avec un espace d'états de grande dimension et a fait preuve d'efficacité dans de nombreux publications scientifiques.

Les approches RL précédentes avaient un problème de conception difficile dans le choix des fonctionnalités. Cependant, le deep RL a réussi dans des tâches compliquées avec des connaissances antérieures plus faibles grâce à sa capacité à apprendre différents niveaux d'abstractions à partir de données. À titre d'exemple et afin d'illuster le concept, un agent Deep RL peut apprendre avec succès à partir d'entrées perceptives visuelles constituées de milliers de pixels. Cela ouvre la possibilité d'imiter certaines capacités de résolution de problèmes humains, même dans un espace de grande dimension - ce qui, il y a seulement quelques années, était difficile à concevoir.

Le Q-learning correspond à l'implémentation de l'approche RL est un algorithme d'apprentissage par renforcement qui cherche à trouver la meilleure action à entreprendre compte tenu de l'état actuel. Plus précisément, le q-learning cherche à apprendre une consuite qui maximise la récompense totale. Le «q» dans le q-learning est synonyme de qualité. Dans ce cas, la qualité représente l'utilité d'une action donnée pour obtenir une récompense future.

Dans notre projet, on vise d'élaborer la partie théorique ainsi qu'une preuve de concept de l'approche de RL tel que montré dans la figure 1 via une conception d'un jeu. On vise de montrer l'intérêt de l'approche et son efficacité dans la résolution de problèmes relatifs à la prise de décision dans un jeu d'application que l'on va exposer.

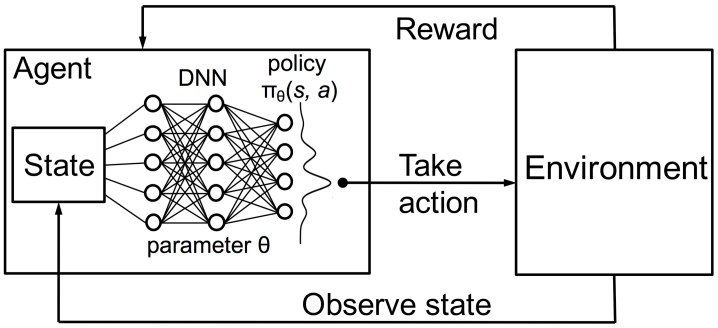

Figure 1. Schéma illustratif des composantes du RL

2 Objectifs

L'objectif du travail est d'exposer et d'implémenter efficacement l'approche de RL avant de procéder à une preuve de concept faisant preuve de besoin à ce type d'approche et exposer les limites ds restes des méthodes tel que le RN et CNN

Plan Le plan envisagé dans la réalisation du projet en cours est comme suit :

1. Revue de littérature relative à l'apprentissage par RL
2. Implémentation des algorithmes principaux choisis en python.

3. Établir une maniere de quantifier de l'efficacité des algorithmes.
4. Analyse des résultats et suggestion des améliorations.

3 Introduction

Apprentissage par renforcement (RL) est une approche informatique utilisée pour comprendre et automatiser l'apprentissage et la prise de décision orientés vers un but[1]. Contrairement aux autres approches, l'apprentissage par renforcement met l'accent sur l'apprentissage par l'interaction directe d'un **agent** avec son **environnement**, sans nécessiter une supervision complète de l'agent ou des modèles "complets" de son environnement.

3.1 A propos de l'apprentissage par renforcement

L'apprentissage par renforcement est le premier domaine à aborder sérieusement les problèmes de calcul découlant des interactions agent-environnement pour atteindre des objectifs à long terme. Il utilise le cadre formel des **processus de décision de Markov** pour définir les interactions entre un agent d'apprentissage et son environnement en termes de :

— **états**
— **actions**
— **récompenses**

Ce cadre est destiné a atteindre une représentation simplifiée des caractéristiques essentielles du **problème de l'intelligence générale artificielle**, à savoir construire un ordinateur capable de "comprendre" ou d'apprendre toute tâche qu'un être humain peut effectuer. [5]. [1]

Les concepts de **valeur** et de **fonction de valeur** sont essentiels à de nombreuses méthodes de RL, fournissant les outils pour une recherche efficace dans l'espace des **politiques** (ou stratégies). (ou stratégies), et aident à distinguer les méthodes RL des **méthodes évolutionnaires**, où les recherches sont menées directement dans l'espace des politiques, guidées par des évaluations de politiques entières.

Histoire de l'apprentissage par renforcement

Historiquement, le terme " renforcement " (dans le contexte de l'apprentissage animal) semble être apparu pour la première fois dans une traduction anglaise du début du 20e siècle des travaux de Pavlov sur les réflexes conditionnés, où le renforcement est décrit comme le " renforcement d'un modèle de comportement dû au fait qu'un animal reçoit un stimulus - un renforçateur - dans une relation temporelle appropriée avec un autre stimulus ou avec une réponse ". Dans les cercles pyschologiques, le renforcement peut parfois inclure l'affaiblissement du comportement, et l'omission/la suppression de stimuli peuvent également être considérés comme des renforçateurs, tant que le changement de comportement persiste une fois qu'ils sont retirés de la situation. célèbres expériences classiques de renforcement du comportement

1. Sans surprise, ce problème est assez difficile à résoudre : les experts ne parviennent même pas à se mettre d'accord sur ce qui constitue une intelligence, bien que la plupart des experts s'accordent à dire qu'une soi-disant intelligence devrait être capable de

— raisonner et utiliser des stratégies pour résoudre des énigmes et porter des jugements dans l'incertitude ;
— représenter des connaissances (sens commun ou non) ;
— planifier ; apprennent ; communique dans un langage naturel, et
— intégrer toutes ces compétences vers des objectifs communs. [5].

D'autres capacités peuvent inclure la capacité de voir, de sentir, de déplacer et de manipuler des objets dans l'univers où se déroule le comportement intelligent, de former des images et des concepts mentaux émergents et de présenter un comportement autonome. Si certains systèmes peuvent accomplir certaines de ces tâches, aucun ne peut encore le faire à un niveau humain, du moins pas selon les différents tests qui ont été proposés :

— the **imitation game** (Turing) – tromper de manière assez cohérente un observateur humain en lui faisant croire qu'un interlocuteur caché est humain/intelligent ;
— the **coffee test** (test du café) (Wozniak) – entrer dans une maison moyenne et comprendre comment préparer une cafetière, à partir de rien ;
— le **test de l'étudiant robot** (Goertzel) – s'inscrire à un programme universitaire, suivre et réussir (parfois échouer ?) les cours comme le ferait un étudiant humain, afin d'obtenir un diplôme ;
— le test d'embauche (Nilsson) – rejoindre la population active et accomplir des tâches économiquement utiles, au moins aussi bien que (certains) humains le font dans le même poste ; [5], et
— the **stand-up comedy test** (Ebert) – faire une apparition raisonnable sur scène. (Ebert) – faire rire une proportion raisonnable d'humains, de manière assez constante.

animal constituent la base de la RL moderne, dans le sens où les agents sont censés imiter le comportement du chien de Pavlov et du pigeon de Skinner lorsqu'ils sont confrontés à des renforçateurs. la première expérience, Pavlov remarque que le fait de présenter de la nourriture à un chien fait saliver l'animal. Il a mis en place une expérience répétée : chaque fois qu'on présente de la nourriture à un chien, une cloche sonne. Avec le temps, le son de la cloche devient un renforçateur, de sorte que le chien finit par saliver en entendant la cloche, que la nourriture soit présentée simultanément ou non. la deuxième expérience, Skinner a installé une boîte dans laquelle on peut appuyer sur un levier pour faire tomber de la nourriture pour oiseaux. Un pigeon a appris par essais et erreurs qu'en appuyant sur le levier, il recevait de la nourriture (notez que le dispositif peut être rendu plus compliqué pour conditionner la livraison de nourriture *via* un signal sonore ou visuel ; les résultats sont les mêmes). la première expérience, il n'y a pas d'interaction entre l'essai et l'erreur : le chien apprend un nouveau comportement par une exposition répétée sur laquelle il n'a aucun contrôle. Dans la deuxième expérience, le pigeon est récompensé en appuyant sur le levier ; la récompense alimentaire renforce l'action (appuyer sur le levier), qui est ensuite exécutée plus souvent dans l'espoir d'autres récompenses. En outre, le comportement du pigeon peut être modifié en modifiant la récompense.

3.2 Bandits Multi-Armés

Nous commençons par étudier l'aspect évaluatif de l'apprentissage par renforcement dans un cadre simple, qui est utilisé pour introduire un certain nombre de méthodes d'apprentissage de base, pour finalement aboutir au **problème d'apprentissage par renforcement complet**.

*Le problème à K bandit armé .

Le **problème du bandit à plusieurs bras** est un problème classique qui illustre le **exploration/exploitation** dilemme central de l'apprentissage par renforcement. D'un côté, si nous avons appris toutes les informations sur l'environnement, nous pouvons trouver la meilleure action à entreprendre dans n'importe quelle situation en explorant simplement l'espace des conséquences, en utilisant une approche de force brute. En pratique, cependant, seules des informations incomplètes sont (généralement) disponibles, de sorte que toute action entreprise ne peut être garantie comme étant la meilleure : nous devons prendre en compte le risque de prendre une mauvaise décision en raison d'informations non optimales.
Dans le cadre d'une approche d'exploitation, nous choisissons la meilleure option disponible compte tenu des informations dont nous disposons ; dans le cadre d'une approche d'exploration, nous pouvons suivre un chemin d'action sous-optimal si cela peut nous aider à recueillir davantage d'informations sur l'environnement (et donc à prendre de meilleures décisions par la suite) - la meilleure stratégie à long terme peut impliquer des sacrifices à court terme. exemple, un spécialiste des mots croisés qui ne donne une réponse que lorsqu'il est certain qu'il s'agit de la bonne réponse (approche d'exploitation) fera peu d'erreurs, mais il ne réussira pas toujours à atteindre son objectif à long terme, qui est de terminer le puzzle, alors qu'un spécialiste des mots croisés qui donne parfois une réponse même s'il n'est pas certain qu'il s'agit de la bonne réponse (essai d'exploration) fera certainement plus d'erreurs (et ses grilles remplies seront plus désordonnées !), mais il est probable qu'il aura un taux de réussite global plus élevé, car ces réponses supplémentaires qui ont été posées avec moins de 100 % de certitude peuvent tout de même ouvrir des réponses à d'autres indices.

Dans le problème du **K-bandit armé**, nous nous trouvons dans un casino face à K machines à sous [2], chacune d'entre elles étant configurée avec une probabilité constante inconnue de récompenser le joueur après un tour. Les probabilités peuvent être différentes d'une machine à sous à l'autre. En supposant que toutes les machines sont gratuites et que l'on peut passer de l'une à l'autre après n'importe quel tour, quelle stratégie devrions-nous utiliser pour obtenir la plus grande récompense à long terme (voir Figure 2)

La situation est plus facile à analyser si l'on suppose que l'on peut continuer à jouer indéfiniment, car la restriction d'un nombre fini d'essais introduit un type différent de problème d'exploration. [3]

L'approche naïve consiste à continuer à jouer avec une machine pendant plusieurs tours afin d'utiliser la **loi des grands nombres** pour estimer sa probabilité de récompense v̈raie ", puis de passer à une autre machine et d'estimer sa probabilité de récompense v̈raie ", puis de passer à une autre machine et ainsi de suite. Cette approche présente le double inconvénient d'être très gaspilleuse, tout en ne

2. qui sont connues, dans le jargon, sous le nom de *bandits manchots*.
3. Si le nombre de tours autorisés est inférieur au nombre de machines à sous, par exemple, nous ne pouvons même pas essayer toutes les machines pour estimer les probabilités de récompense et nous devons nous comporter intelligemment par rapport à des connaissances et des ressources limitées.

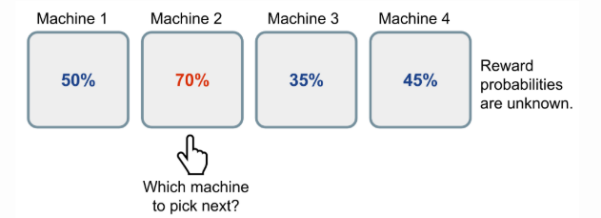

FIGURE 2. Une illustration d'un bandit armé de Bernoulli à 4. Les probabilités de récompense sont inconnues du joueur.

garantissant même pas qu'elle fournira la meilleure récompense à long terme.

Formalisme

A **les bandits K armées de Bernoulli** est un tuple $\langle A, R, \theta \rangle_K$, avec :

- les K probabilités de récompenses sont $\theta = \{\theta_1, \ldots, \theta_K\}$;
- a chaque étape t, l'action $a_t \in A$ consiste a choisir une machine $i \in \{1, \ldots, K\}$ et recevoir les récompenses $r_t \in R$;
- A est un ensemble d'actions, chacun fait référence a une interaction avec une machine spécifique – la **valeur** des actions a la récompense espérée, $Q(a) = \mathrm{E}[r|a] = \theta$; si on décide de prendre l'action $a_t \in A$ a chaque étape t sur la $i$eme machine, alors $Q(a_t) = \theta_i$, et
- R est une fonction de récompense $R : A \to [0, 1]$ – dans le cas des bandits de Bernoulli , on observe une récompense $r$dans le mode stochastique, qui est, a chaque étape t,

$$r_t = R(a_t) = \begin{cases} 1 & \text{with probability } Q(a_t) \\ 0 & \text{otherwise} \end{cases}$$

Ce processus est une instance du **processus de décision de Markov** avec un ensemble vide des états possibles $S = \varnothing$ (see Section 3).

Le but du $K-$armée bandit est de maximiser la récompense cumulative

$$\text{cumulative reward} = \sum_{t=1}^{T} r_t, \quad T \in \mathbb{N} \cup \{\infty\}.$$

Ce problème est équivalent a minimiser le **regret potentiel** (ou perte) de ne pas avoir choisi l'action optimal.

La probabilité de la récompense optimale θ^* de l'action optimale a^* est alors :

$$\theta^* = Q(a^*) = \max_{a \in A} \{Q(a)\} = \max_{1 \le i \le K} \{\theta_i\}.$$

La **fonction de perte** est le total du regret, en moyenne , par ne pas sélectionner la stratégie optimal :

$$\mathcal{L}_T = \mathrm{E}\left[\sum_{t=1}^{T} \left(\theta^* - Q(a_t)\right)\right].$$

Stratégies des Bandits

Selon la façon dont nous explorons l'espace d'action, il existe plusieurs façons de résoudre le problème du bandit à bras multiples :

- L' **algorithme glouton** (greedy) ne laisse aucune place à l'exploration ;
- le $\varepsilon-$ **algorithme glouton** nécessite une exploration aléatoire, tandis que
- l' **algorithme des limites supérieures de confiance** utilise des explorations intelligentes.

L' $\varepsilon-$ algorithme glouton

l' $\varepsilon-$ **algorithme glouton** exécute la meilleure action disponible , mais pourrait occasionnellement utiliser des explorations aléatoires . La valeur de l'action cible a est estimée selon les expériences passées, par un calcul de la moyenne des récompenses associées aux a lorsqu'elle était prise durant les états précédentes :

$$\hat{Q}_{t+1}(a) = \frac{1}{N_t(a)} \sum_{\tau=1}^{t} r_\tau \mathbb{1}[a_\tau = a],$$

avec $\mathbb{1}$ est la **fonction indicatrice binaire** et $N_t(a)$ enregistre la fréquence des actions a , i.e.

$$N_t(a) = \sum_{\tau=1}^{t} \mathbb{1}[a_\tau = a].$$

Lors de l'utilisation de l' algorithme glouton, l'action cible avec la plus grande valeur $\hat{Q}_t(a)$ est sélectionnée dans tout étape de temps t.

Avec une faible probabilité ε, l'algorithme $\varepsilon-$glouton choisira plutôt une action aléatoire ; avec une probabilité $1-\varepsilon$ (ce qui devrait s'avérer être la plupart du temps), il reviendra à l'action de l'algorithme $\hat{a}_t^* = \arg\max_{a\in A} \hat{Q}_t(a)$ (notez que l'action aléatoire pourrait s'avérer être a^*).

Si, pour chaque action a, nous gardons la trace des récompenses et des valeurs à chaque pas de temps où elle est utilisée, nous pouvons facilement concevoir des formules incrémentales qui mettent à jour la moyenne des récompenses avec une puissance de calcul très faible. Étant donné une valeur d'action $Q_{t_n}(a)$ et la récompense correspondante $r_{t_n}(a)$, et en supposant que l'action a a été effectuée $n-1$ fois au cours des pas de temps précédents, aux temps $t_1, \ldots, t_{n-1}$, la (nouvelle) moyenne de toutes les récompenses pour a jusqu'au pas de temps t_n est la suivante

$$\begin{aligned}
\frac{1}{n}\sum_{i=1}^{n} r_{t_i}(a) &= \frac{1}{n}\left(r_{t_n}(a) + \sum_{i=1}^{n-1} r_{t_i}(a)\right) \\
&= \frac{1}{n}\left(r_{t_n}(a) + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} r_{t_i}(a)\right) \\
&= \frac{1}{n}\left(r_{t_n}(a) + (n-1)Q_{t_n}(a)\right) \\
&= \frac{1}{n}\left(r_{t_n}(a) + nQ_{t_n}(a) - Q_{t_n}(a)\right) \\
&= Q_{t_n}(a) + \frac{1}{n}\left[r_{t_n}(a) - Q_{t_n}(a)\right]
\end{aligned}$$

Le pseudo-code d'un algorithme complet de bandit armé $\varepsilon-$greedy $K-$ utilisant des moyennes d'échantillons calculées de manière incrémentielle est présenté ci-dessous ; la fonction *bandit*(a) prend une action $a \in \{l, \ldots, K\}$ et renvoie la récompense correspondante.

Algorithm 1: The $\varepsilon-$Greedy Algorithm

Input : a threshold T for time steps, a set of actions $A = \{1, \ldots, K\}$
Initialize : for $a \in A$
$Q(a) \leftarrow 0$
$N(a) \leftarrow 0$
for $1 \leq t < T$ **do**

$$\alpha \leftarrow \begin{cases} \arg\max\limits_{a\in A}\{Q(a)\} & \text{with probability } 1-\varepsilon \\ 1 & \text{with probability } \varepsilon/K \\ \vdots & \vdots \\ K & \text{with probability } \varepsilon/K \end{cases}$$

$R \leftarrow$ *bandit*(α)
$N(\alpha) \leftarrow N(\alpha) + 1$
$Q(\alpha) \leftarrow Q(\alpha) + \frac{1}{N(\alpha)}[R - Q(\alpha)]$
end
Output : $Q(1), \ldots, Q(K)$

Noter que for $\varepsilon = 0$, L'algorithme se réduit à l'algorithme classique, dans lequel aucune exploration n'est effectuée.

L'Algorithme des limites supérieures de confiance

L'exploration aléatoire donne l'occasion d'essayer des options que nous ne connaissons peut-être pas aussi bien ; cependant, nous pouvons finir par explorer des avenues sans issue. [4]

Pour réduire le risque d'une exploration inefficace, nous pouvons réduire la valeur du paramètre $\varepsilon = \varepsilon_t$ au fil du temps, afin de refléter une confiance croissante dans les actions "éprouvées, testées et vraies" ; une autre approche pourrait consister à rester (artificiellement) optimiste quant aux options pour lesquelles la valeur de l'action n'a pas encore été estimée avec confiance – nous pourrions vouloir favoriser

4. "Barking up the wrong tree", comme on dit en anglais.

"l'exploration des actions ayant un fort potentiel pour avoir une valeur optimale". [1]

Les **limites superieures de confidence** (UCB) mesure ce potentiel *via* une borne de confiance supérieure sur la valeur action/récompense, $\hat{U}_t(a)$, de sorte que la valeur réelle soit limitée par

$$Q(a) \leq \hat{Q}_t(a) + \hat{U}_t(a)$$

avec une probabilité élevée. La limite supérieure $\hat{U}_t(a)$ est une fonction de $N_t(a)$; un plus grand nombre d'essais $N_t(a)$ devrait donner une limite plus petite $\hat{U}_t(a)$.

L'algorithme UCB sélectionne l'action la plus rapide qui maximise la borne de confiance supérieure :

$$a_t^{UCB} = \arg\max_{a \in A} \left\{ \hat{Q}_t(a) + \hat{U}_t(a) \right\}.$$

Comment l'estimation $\hat{U}_t(a)$ est-elle calculée ? Si l'on ne suppose aucune connaissance préalable de la distribution des récompenses, on peut utiliser l'inégalité de Hoeffding pour obtenir l'estimation.

Soit $X_1, \ldots, X_t$ des variables aléatoires indépendantes et identiquement distribuées (i.i.d.) sur l'intervalle $[0, 1]$, avec la moyenne d'échantillon

$$\overline{X}_t = \frac{1}{t} \sum_{\tau=1}^{t} X_\tau.$$

Alors, pour $u > 0$, on a :

$$P\left[\mathrm{E}[X] > \overline{X}_t + u\right] \leq e^{-2tu^2}.$$

Dans ce contexte ,on utilise :

- $X_\tau = r_\tau(a)$ les variables aléatoires ;
- $E[X_\tau] = Q(a)$ est la veleur moyenne réelle de l'action ;
- $\overline{X} = \hat{Q}_t(a)$ est la moyenne de l'échantillon , et
- $u = U_t(a)$ est la limite supérieure de confiance.

alors l'inégalité Hoeffding devient

$$P\left[Q(a) > \hat{Q}_t(a) + U_t(a)\right] \leq e^{-2tU_t(a)^2}.$$

Si $e^{-2tU_t(a)^2}$ est petit, alors $Q(a) < \hat{Q}_t(a) + U_t(a)$ avec une grande probabilité . pour un petit seuil p,

$$e^{-2tU_t(a)^2} = p \implies U_t(a) = \sqrt{\frac{-\log p}{2N_t(a)}}.$$

Comme nous pourrions vouloir réduire le seuil $p = p_t$ dans le temps afin de devenir plus confiant dans l'UCB au fur et à mesure de l'observation des récompenses, nous pourrions fixer $p_t = t^{-\ell}$, $\ell \in \mathbb{N}$. Pour $\ell = 4$, on obtient l'algorithme **UCB1**:

$$U_t(a) = \sqrt{\frac{2\log t}{N_t(a)}} \text{ and}$$

$$a_t^{UCB1} = \arg\max_{a \in A} \left\{ Q(a) + \sqrt{\frac{2\log t}{N_t(a)}} \right\}.$$

3.3 Terminologie de base de l'apprentissage par renforcement

Rappelons que la RL est une branche de l'IA qui traite des agents qui perçoivent leur environnement à travers le **espace d'état** et agissent sur lui à travers le **espace d'action**, résultant en un état (potentiellement) différent et recevant une récompense en retour pour cette action, qui est assignée au nouvel état. Dans l'apprentissage supervisé, nous formons des modèles en minimisant une fonction de coût ; dans la RL, l'agent maximise la récompense globale pour trouver la **politique optimale de résolution des tâches**.

Reinforced vs. Supervised/Unsupervised Learning

Dans l'apprentissage supervisé (SL), l'ensemble d'apprentissage

$$\mathbf{Z} = [\mathbf{X}, \mathbf{Y}] \in \mathbb{M}_{n,p} \times \mathbb{M}_{m,p}$$

possède des **caractéristiques d'entrée** $X = \{X_1, \ldots, X_p\}$ et des **étiquettes de sortie** $Y = \{Y_1, \ldots, Y_m\}$ correspondants.

On entraîne un modèle f sur l'ensemble d'apprentissage $\mathbf{Z}$ et on obtient des prédictions $\mathbf{Y}^*$ pour l'ensemble de données de test $\mathbf{Z}' = [\mathbf{X}', \mathbf{Y}']$, telles que $\mathbf{Y}^* = f(\mathbf{X}')$; les bons modèles sont ceux pour lesquels les prédictions $\mathbf{Y}^*$ et les véritables étiquettes de sortie $\mathbf{Y}'$ sont "proches". [5]

Dans l'apprentissage non supervisé (UL), seules les valeurs d'entrée sont prises en compte. (UL), seules les valeurs d'entrée $\mathbf{X}$ de l'ensemble de formation sont utilisées à des fins de formation : aucune valeur d'étiquette de sortie $\mathbf{Y}$ n'est associée. exemple, l'objectif peut être de construire un modèle qui apprend à séparer les données en différents **clusters** en identifiant les groupes naturels présents dans les données. Ce modèle pourrait ensuite être utilisé sur des cas de test $\mathbf{X}'$ pour prédire leur similarité (et la probabilité éventuelle d'affectation) aux clusters. [6]

RL prend une voie différente, puisqu'il guide les agents sur la façon d'agir dans leur environnement ; l'interface comprend plus que de simples vecteurs de formation (comme c'est le cas dans SL et UL). L'objectif de la formation est de permettre aux agents d'atteindre un état cible, et non de maximiser une probabilité ou de minimiser un coût comme dans le cas de SL.

RL, les agents reçoivent automatiquement un retour d'information, sous la forme de **récompenses de l'environnement**, et des activités telles que l'étiquetage (qui prennent du temps pour les humains) ne sont pas nécessaires.

5. Cela peut être fait d'une multitude de façons [7]. La validité d'un modèle dépend également de l'ensemble de données : aucun modèle ne peut systématiquement surpasser les prédictions aléatoires lorsqu'il s'agit de prédire des tirages à pile ou face, par exemple. Notez également que le modèle f peut contenir des éléments stochastiques.

6. L'absence d'étiquettes pour "vérifier" les affectations de clusters, par exemple, crée une bonne dose d'ambiguïté, qui est aggravée par la grande variété d'algorithmes et de mesures de similarité qui peuvent être utilisés en pratique [8].

L'un des principaux avantages de la RL par rapport à la SL/UL est que la formulation de l'objectif d'une tâche sous la forme d'un but permet de résoudre une grande variété de problèmes. Par exemple, l'objectif d'un agent de jeu vidéo peut être de gagner la partie en obtenant le meilleur score possible, en atteignant la ligne d'arrivée en premier, ou en découvrant de nouvelles façons d'atteindre ces objectifs. [7]

Terminologies et Conventions

Nous avons utilisé divers termes pour décrire (de manière assez vague) un certain nombre de concepts ; nous allons maintenant définir les termes tels qu'ils seront utilisés dans la suite de ce travail :

- un **agent** est programmé pour "sentir" son environnement, effectuer des actions (sélectionnées dans un espace d'action), recevoir un retour (sous la forme d'une récompense numérique ou d'une pénalité), et tenter de maximiser les récompenses ou de minimiser les pénalités ;
- **L'environnement** de l'agent est le monde dans lequel il réside ; il peut être réel (comme en robotique) ou simulé (virtuel) ;
- l'agent perçoit son environnement en lisant son **état**, c'est-à-dire la configuration dans laquelle se trouve l'environnement à chaque instant (paramètres, position des objets, lieux occupés, etc.) ; l'espace des états de l'environnement peut être fini (et petit, comme dans le jeu du morpion, ou grand, comme dans les jeux de Go et d'échecs), ou infini (comme la position de la tête de lecture d'une machine de Turing) ;
- tout ce que l'agent est capable de faire dans son environnement est appelé une **action** ; les actions disponibles à un moment donné peuvent dépendre de l'état de l'environnement ; l'espace des actions peut être fini (comme le serait la liste des coups disponibles dans le tic-tac-toe, le Go ou les échecs) ou infini (comme la sélection d'une valeur dans l'intervalle $[0, 1]$ et son ajout à une valeur d'état initialisée pour obtenir un nouvel état) ;
- le retour que l'agent reçoit en fonction de l'action qu'il a entreprise est donné sous la forme d'une **récompense** ou d'une **pénalité** numérique ; l'objectif de l'agent est de maximiser (resp. minimiser) la **récompense globale** Les récompenses sont définies avant que l'agent ne soit lâché dans l'environnement et doivent être créées correctement pour que l'agent puisse atteindre efficacement son ou ses objectifs ;
- la combinaison d'un état, d'une action et d'une récompense, est appelée **SAR triple**, et est représentée par le tuple $(\mathbf{s}, \mathbf{a}, \mathbf{r})$,(state, action,reward) et, enfin,
- l'exécution complète de la tâche par un agent (de l'initialisation à l'achèvement, en supposant que la tâche puisse être achevée) est un RL **episode**.

7. Lorsqu'AlphaGo, un système d'IA conçu par DeepMind, a battu Lee Sedol, l'un des meilleurs joueurs de go au monde, dans une série de matchs au meilleur des cinq manches en 2016, il a trouvé des façons nouvelles et peu orthodoxes de gagner. Il est intéressant de noter que Sedol aussi a gagné une partie contre AlphaGo en utilisant un mouvement "miraculeux" surnommé "God's Touch" par les joueurs du monde entier. Il semblerait que les humains et les agents d'IA puissent (encore) apprendre les uns des autres dans le monde de la RL [9].

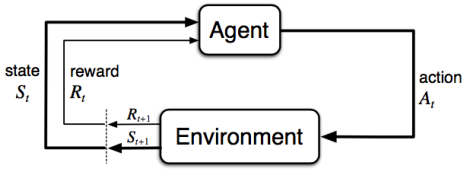

Figure 3. L'interaction agent-environment [1].

Chaque tâche RL peut donc être représentée comme une séquence de triples SAR : à partir d'un état S_t, l'agent effectue une action A_t donnant le nouvel état S_{t+1}, pour lequel il reçoit une récompense R_{t+1}. [8] que, S_t et A_t résultent en S_{t+1}, nous pourrions également représenter les étapes comme un triple SAS (état actuel, action, nouvel état), c'est-à-dire : (S_t, A_t, S_{t+1}) ou (s, a, s') (voir Figure 3).

Critères d'optimalité et validation

Critères d'optimalité sont des mesures de la qualité de l'ajustement du modèle créé sur les données. Dans les modèles SL de classification/régression, par exemple, le maximum de vraisemblance est souvent utilisé comme critère d'optimalité.

Les critères d'optimalité diffèrent en fonction de l'énoncé du problème et des objectifs de la tâche. En RL, l'objectif principal est de maximiser les récompenses futures (ou de minimiser les pénalités futures). En tant que tel, il existe deux exigences :

— on a besoin de **la fonction des valeurs** afin de quantifier les états en se basant sur les récompenses futurs probables , et
— la **stratégie** afin de guider l'agent sur quelle action a exécuter dans une certaine (state).

Fonction de Valeur Les agents intelligents doivent être capables de "penser" à la fois aux récompenses immédiates et aux récompenses futures. Une valeur doit être attribuée à chaque état rencontré pour refléter cette information future, à l'aide de la fonction **valeur**. C'est là que le concept de **récompenses différées** entre en jeu.

La **value** $V_t(s)$ de l'état s au temps t est la valeur attendue des récompenses futures pour toutes les actions entreprises sur l'état s jusqu'à ce que l'agent atteigne l'état cible (c'est-à-dire que l'objectif a été atteint). Les états ayant des valeurs élevées sont des états à partir desquels il est plus probable que l'état cible puisse finalement être atteint.

Alors que la récompense attribuée à chaque triplet SAS (s, a, s') est fixe, ce n'est pas le cas de $V_t(s)$; elle est susceptible de changer avec chaque action au sein de l'épisode et d'un épisode à l'autre, puisque $V_t(s)$ dépend de t ainsi que de s. conséquent, il peut sembler préférable de stocker la connaissance de chaque état possible, plutôt que la fonction de valeur, mais ce n'est pas une idée pratique, sauf pour les systèmes de RL les plus simples : cette approche est à la fois **consommatrice de temps** et coûteuse, le coût augmentant de manière exponentielle avec la complexité du problème.

Formellement, nous avons

$$V_t(s) = \mathrm{E}\left[\text{toutes les récompenses futures actualisées} | S_t = s\right].$$

Plus de détails sur la fonction de valeur et sur les remises sont fournis dans les autres sections.

Modèle de Stratégie Une **stratégie** est un modèle qui guide l'agent lorsque vient le temps de choisir une action dans différents états. La politique est typiquement dénotée par π, la probabilité qu'une certaine action soit entreprise dans un état donné :

$$\pi(a, s) = P(A_t = a | S_t = s).$$

Un **policy map** doit donc fournir l'ensemble des probabilités pour les différentes actions étant donné un état particulier.

L'utilisation d'une politique appropriée, associée à la fonction de valeur, aidera l'agent à naviguer dans son environnement de manière efficace.

8. Le couple état-action actuel donne les récompenses pour l'étape suivante.

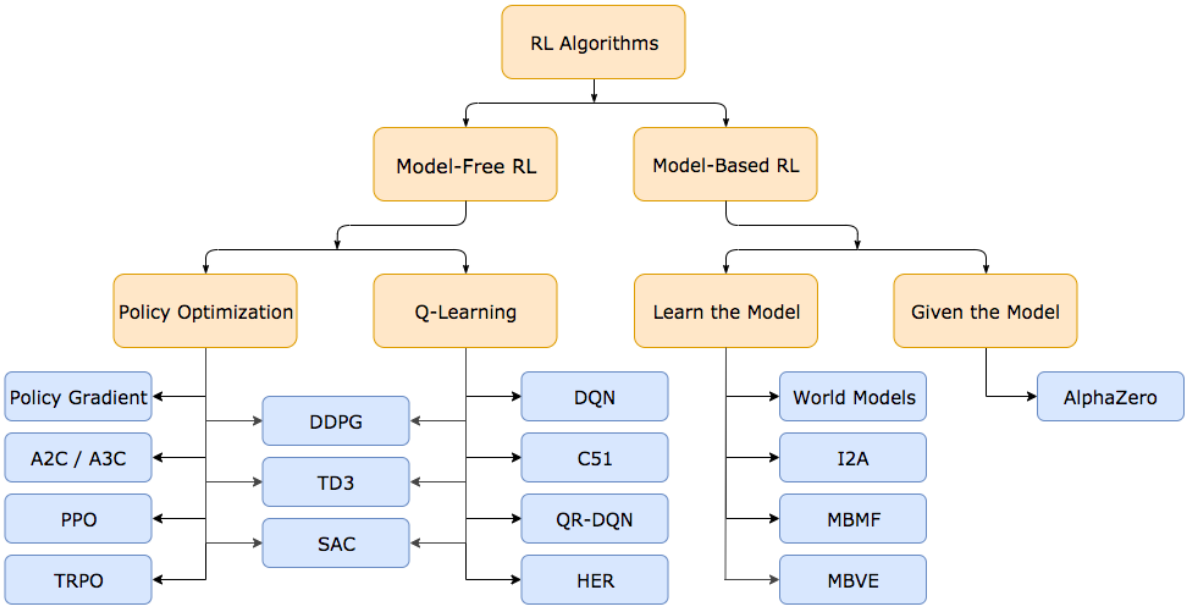

Figure 4. Taxonomie du RL [10].

3.4 Taxonomie de l'apprentissage par renforcement

Récemment, un certain nombre d'algorithmes et d'approches d'apprentissage par renforcement sont devenus plus populaires, tels que l'apprentissage $Q-$ et l'algorithme A3C (une taxonomie est fournie dans la Figure 4). Nous discutons de certains des points saillants dans le reste de cette section.

Model-Free RL vs. Model-Based RL

L'un des facteurs de différenciation les plus fondamentaux pour les différents types d'algorithmes de RL est de savoir si l'agent a accès à (ou apprend) un modèle de l'environnement sur la voie de la réalisation de son ou ses objectifs, ou s'il agit sur son environnement sans essayer de **prédire** les transitions d'état et les récompenses (ou pénalités).

Le principal avantage de l'accès à un modèle est qu'il permet à l'agent de planifier en "réfléchissant" à l'avance, en voyant ce qui se passerait pour une gamme de choix possibles, et en décidant explicitement entre les options disponibles, en distillant finalement les résultats dans une politique apprise.

Les algorithmes qui utilisent un tel modèle sont appelés méthodes basées sur un modèle, tandis que ceux qui ne le font pas sont appelés méthodes sans modèle. Les méthodes sans modèle ont tendance à être moins efficaces, car elles ne permettent pas de "penser" à l'avance, mais elles ont aussi tendance à être plus faciles à mettre en œuvre et à ajuster.

Aperçu $Q-$Learning

$Q-$**learning** [11] est un algorithme RL sans modèle, dont le but est **l'apprentissage d'une politique**, à savoir comment un agent devrait agir dans un ensemble de circonstances. Il n'a pas besoin d'un modèle de l'environnement et peut traiter des problèmes de transitions et de récompenses stochastiques, sans nécessiter d'adaptation.

Pour tout *processus de décision de Markov fini* (FMDP, voir la section **??**), $Q-$learning trouve une stratégie optimale qui maximise l'espérance de la récompense totale sur toutes les étapes successives, en commençant par l'étape actuelle.

Le $Q-$learning tente d'apprendre la valeur $Q(s,a)$ d'une action spécifique donnée à l'agent dans un état particulier, en construisant un $Q-$**table**, une matrice $|S| \times |A|$ avec une ligne par état et une colonne par action. La $Q-$table peut ensuite être utilisée pour déterminer quelle action est la meilleure pour un agent dans un état donné (voir le pseudocode dans l'Algorithme 2). En clair, la valeur Q pour une paire

Algorithm 2: $Q-$Learning Steps

1 **Input :** actions A, states S, discount factor $\gamma \in [0,1]$, learning rate $\alpha \in (0,1]$, initial condition $\beta \in [0,1]$
2 **Initialize :** $Q(s,a) \leftarrow \beta$
3 **while** *current state s is not a final state* **do**
4 Select an action $a \in A$ using the $\varepsilon-$greedy method (or any other action selection policy), and apply the action to s
5 Record the reward r and the resulting state s'
6 Update the $Q-$value for the (s,a) pair *via* the Bellman equation :

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left[r + \gamma \cdot \max_{a'}\{Q(s',a')\}\right]$$

7 Set $s = s'$
8 **end**
9 Run multiple episodes to train the agent
10 **Output :** $Q-$table

état-action donnée (s,a) est mise à jour en ajoutant la récompense actuelle r à la valeur Q maximale actualisée (avec un facteur γ) pour le nouvel état s' (résultant de l'action a sur l'état actuel s) parmi toutes les actions possibles ; cette somme est ensuite modifiée par le taux d'apprentissage α et ajoutée à une version modifiée de l'ancienne valeur Q (voir l'étape 6 de l'algorithme).

Le **facteur d'actualisation** $\gamma \in [0,1]$ équilibre l'impact des récompenses futures et présentes ; de petites valeurs de γ rendent l'agent "myope" (privilégiant les récompenses à court terme) tandis que de grandes valeurs le font tendre vers les "récompenses à long terme". Bien que l'adoption d'une vision à plus long terme soit généralement une bonne approche, d'un point de vue conceptuel, des difficultés techniques sont associées à un facteur d'actualisation trop important.

Le **taux d'apprentissage** $\alpha \in (0,1]$ détermine l'effet des anciennes informations sur le processus de mise à jour ; les petites valeurs de α font que l'agent exécute un processus d'exploitation (en se concentrant principalement sur les connaissances passées, ce qui serait bon pour les problèmes stochastiques), tandis que les grandes valeurs lui font exécuter un processus d'exploration (en se concentrant principalement sur les informations les plus récentes, ce qui serait bon pour les problèmes déterministes). **condition initiale** β a également un effet ; les grandes valeurs de β correspondent à des “conditions initiales optimistes” et encouragent l'exploration. Il existe diverses stratégies pour **réinitialiser** les conditions initiales, par exemple en fixant $Q(s,a) = r$ la première fois qu'une action a est entreprise.[9] pagePlusieurs épisodes peuvent être nécessaires car, dans tout épisode donné, il est concevable qu'une paire état-action (s,a) ne soit pas rencontrée par l'agent, ce qui laisserait la table $Q-$ sans valeur actualisée pour cette paire. tant donné que l'équation de Bellman nécessite également la connaissance d'une ligne complète de la table $Q-$ lors de la mise à jour de $Q(s,a)$, l'exécution de plusieurs épisodes permet de garantir que les valeurs de la table $Q-$ convergent vers des représentations précises de la récompense future attendue pour une paire état-action (s,a). les espaces d'état et d'action continus, $|S| = |A| = \infty$; même pour les systèmes discrets, le maintien d'une $Q-$table devient très coûteux lorsque $|S|$ et $|A|$ sont grands. **L'ajustement de surface** (ou apprentissage de fonction) peut produire les valeurs $Q-$ sans table $Q-$. méthode d'ajustement de surface raisonnable peut être utilisée pour approximer $Q(s,a)$; lorsque $Q(s,a)$ est approximé par un réseau neuronal, on parle de $Q-$**réseau** ; lorsque $Q(s,a)$ est approximé par un **réseau neuronal plus profond**, tel qu'un réseau neuronal convolutif (CNN) [27], l'algorithme est appelé **réseau** $Q-$ **profond**. (DQN) [18–20].

9. Note que $\gamma = \gamma_t$, $\alpha = \alpha_t$ et $\beta = \beta(s,a)$ ne doivent pas nécessairement être constants.

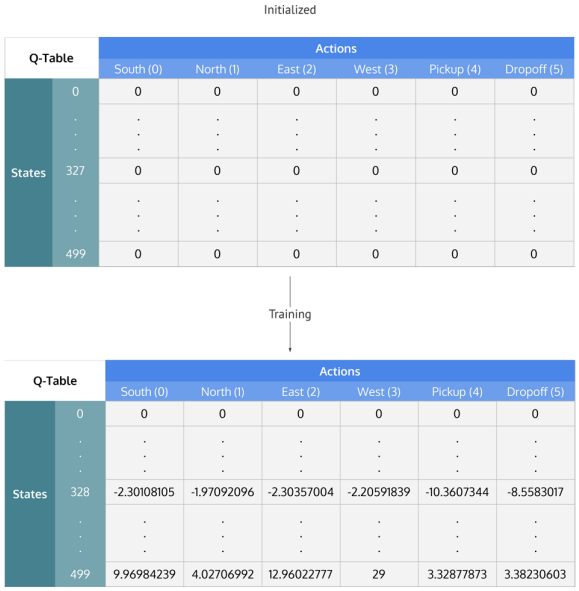

Figure 5. Illustration of $Q-$table updating : pre- and post-training [28].

Aperçu A3C

L'approche **asynchronous advantage actor-critic** (A3C) [12] est un cadre récent, simple et rapide qui utilise la descente de gradient asynchrone pour optimiser le réseau neuronal profond sous-jacent. L'un de ses principaux avantages est qu'elle peut fonctionner à la fois sur des espaces d'action continus et discrets, ce qui ouvre la voie à de nombreux nouveaux problèmes difficiles qui présentent des espaces d'état et d'action complexes. l'instant, nous nous contentons de déballer le nom de l'algorithme afin d'obtenir un aperçu de base de son fonctionnement.

A la base, A3C est **asynchrone**, ce qui signifie qu'il crée de multiples copies de l'agent et de son environnement, mais que les agents n'interagissent pas entre eux ; chaque agent utilise un DQN différent (ou le même réseau mais avec des paramètres différents) pour apprendre la $Q-$fonction $Q(s,a)$. Puisque tous les agents sont indépendants les uns des autres, l'expérience globale résulte d'un apprentissage diversifié, et des procédures telles que le bagging ou l'apprentissage d'ensemble peuvent être utilisées pour fournir de meilleures estimations de la fonction $Q-$.

A3C est également **agent-critique**, combinant les avantages à la fois de **l'itération des valeurs** et **l' itération ds stratégies**. Le DQN sous-jacent estime à la fois une fonction de valeur $V(s)$, et une politique $\pi(s)$ pour un état donné s. Il contient *deux couches distinctes et entièrement connectées* au sommet du réseau neuronal d'approximation de fonction ; ces couches produisent respectivement $\hat{V}(s)$ et $\hat{\pi}(s)$. L'agent utilise ensuite le **valeur**, qui agit comme un **critique**, pour mettre à jour le **stratégie** d'un **agent** intelligent.

Enfin, le **avantage** d'une action a sur l'état s est la différence entre la valeur $Q(s,a)$ de la paire état-action et la valeur de l'état $V(s)$. Cette métrique permet non seulement de quantifier une action comme étant "bonne" ou "mauvaise", mais elle fournit un moyen d'encourager (ou de décourager, selon le cas) les actions mieux que les rendements actualisés des méthodes de gradient de la stratégie.

4 Application d'apprentissage par renforcement par OpenAI Gym

Dans cette partie , on vise à réaliser une application préliminaire de l'apprentissage par renforcement en utilisant l'environment OpenAI Gym.

L'OpenAI Gym fournit un grand nombre d'environnements virtuels pour former vos agents d'apprentissage par renforcement. Dans l'apprentissage par renforcement, la tâche la plus difficile est de créer l'environnement. C'est là qu'OpenAI Gym vient à la rescousse, en fournissant un grand nombre d'environnements de jeux jouets pour offrir aux utilisateurs une plateforme pour former et évaluer leurs agents

d'apprentissage par renforcement.

En d'autres termes, il fournit à l'agent d'apprentissage par renforcement un terrain de jeu pour apprendre et évaluer ses performances, où l'agent doit apprendre à naviguer de l'état de départ à l'état d'arrivée sans subir de contretemps.

Ainsi, dans ce chapitre, nous apprendrons à comprendre et à utiliser les environnements d'OpenAI Gym et nous essaierons d'implémenter le Q-learning de base et le Q-network pour que nos agents puissent apprendre.

4.1 OpenAI Gym avec l'environment Frozen-Lake

Pour comprendre les bases de l'importation des paquets Gym, du chargement d'un environnement et d'autres fonctions importantes associées à OpenAI Gym, le détail de l'exemple d'un **Frozen Lake** est donné dans le notebook associé.

L'agent contrôle le mouvement d'un personnage dans un monde quadrillé. Certaines tuiles de la grille sont praticables, tandis que d'autres conduisent l'agent à tomber dans l'eau. De plus, la direction du mouvement de l'agent est incertaine et ne dépend que partiellement de la direction choisie. L'agent est récompensé lorsqu'il trouve un chemin praticable vers une tuile cible.

L'hiver est là. Vous et vos amis jouiez au frisbee dans le parc quand vous avez fait un lancer sauvage qui a laissé le frisbee au milieu du lac. L'eau est en grande partie gelée, mais il y a quelques trous où la glace a fondu. Si vous entrez dans un de ces trous, vous tomberez dans l'eau glacée. En ce moment, il y a une pénurie internationale de frisbee, il est donc impératif que vous traversiez le lac et récupériez le disque. Cependant, la glace étant glissante, vous ne pourrez pas toujours avancer dans la direction souhaitée. [6]

La surface est décrite comme le suivant :

- **SFFF :** (S : starting point, safe)
- **FHFH :** (F : frozen surface, safe)
- **FFFH :** (H : hole, fall to your doom)
- **HFFG :** (G : goal, where the frisbee is located)

Le jeu se termine lorsqu'on arrive au but ou tombe dans une cellule H. On reçoit une récompense de 1 si on arrive a G, et 0 autrement.

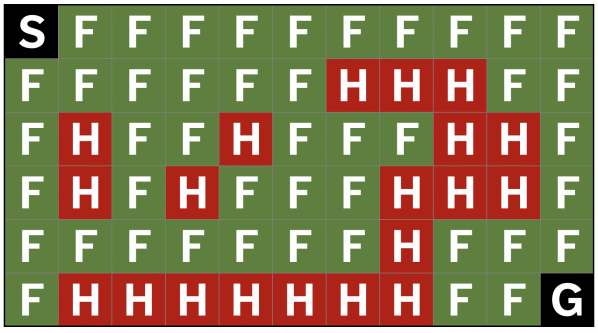

Figure 6. FrozenLake

4.2 Programmer un agent pour le problème Frozen Lake avec Q-Learning

Essayons maintenant de programmer un agent d'apprentissage par renforcement en utilisant le Q-learning. Le Q-learning consiste en une table Q qui contient des valeurs Q pour chaque paire état-action. Le nombre de lignes de la table est égal au nombre d'états dans l'environnement et le nombre de colonnes est égal au nombre d'actions. Comme le nombre d'états est de 16 et le nombre d'actions de 4, la table Q pour cet environnement est constituée de 16 lignes et de 4 colonnes.

Comme nous l'avons décrit dans l'algorithme 2 (chapitre 1), les étapes de l'apprentissage Q sont :

1. Initialiser la table Q avec des zéros (éventuellement, la mise à jour se fera avec une récompense reçue pour chaque action prise pendant l'apprentissage). mise à jour d'une valeur Q pour une paire état-action, c'est-à-dire $Q(s, a)$ est donnée par :

$$Q(s,a) \longleftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q\left(s', a'\right) - Q(s,a) \right]$$

   with :

   — s : état actuel

   — a : action choisie (choisir une nouvelle action via l'approche epsilon-greedy)

   — $s^{'}$: nouvel état résultant

   — $a^{'}$: action pour le nouvel état

   — r : récompense reçu pour l' action

   — α : learning rate, le taux avec lequel l'apprentissage de l'agent converge vers l'erreur minimal.

   — γ : facteur de réduction,'est-à-dire escompter la récompense future pour avoir une idée de l'importance de cette récompense future par rapport à la récompense actuelle.

2. En mettant à jour les valeurs de Q selon la formule mentionnée à l'étape 2, la table converge pour obtenir des valeurs précises pour une action dans un état donné.

Le **Epsilon-Greedy** que nous avons vu dans le premier chapitre (section sur le bandit à bras multiples) est une solution largement utilisée pour résoudre le dilemme explorer-exploiter. L'exploration consiste à rechercher et à explorer de nouvelles options par l'expérimentation et la recherche afin de générer de nouvelles valeurs, tandis que l'exploitation consiste à affiner les options existantes en répétant ces options et en améliorant leurs valeurs.

L'approche Epsilon-Greedy a été décrite dans l'algorithme 1 (section 1).

Finalement, après plusieurs itérations, nous découvrons les meilleures actions parmi toutes à chaque état car il a la possibilité d'explorer de nouvelles actions aléatoires ainsi que d'exploiter les actions existantes et de les affiner.

Essayons d'implémenter un algorithme de base de Q-learning pour qu'un agent apprenne à naviguer sur ce lac gelé de 16 grilles, du début à la fin, sans tomber dans le trou.

Le Notebook joint contient l'implementation de cet example.

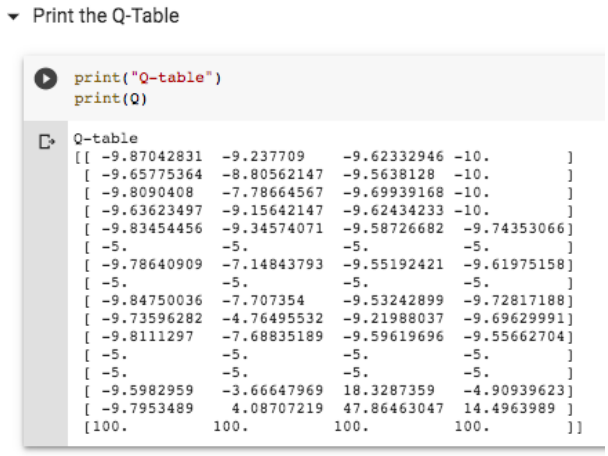

Figure 7. Q-Learning Table After Training

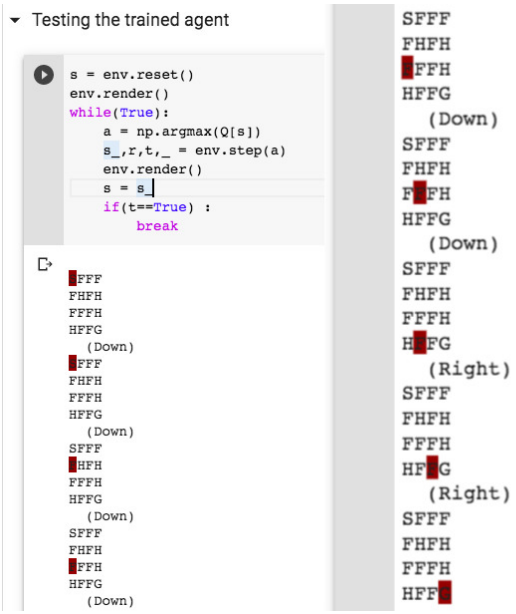

FIGURE 8. Agent test après le training

4.3 Programmer un agent pour Frozen Lake avec Q-Network

Maintenir une table pour un petit nombre d'états est possible mais dans le monde réel, les états deviennent infinis. Il est donc nécessaire de trouver une solution qui incorpore les informations d'état et donne les valeurs Q pour les actions sans utiliser la table Q. C'est là que le **réseau neuronal** agit comme un approximateur de fonction, qui est formé sur des données de différentes informations d'état et leurs valeurs de Q correspondantes pour toutes les actions, ce qui lui permet de prédire les valeurs de Q pour toute nouvelle entrée d'information d'état. Le réseau neuronal utilisé pour prédire les valeurs Q au lieu d'utiliser une table Q est appelé **Q-network**.

Utilisons un seul réseau neuronal qui prend en entrée des informations sur l'état, où celles-ci sont représentées sous la forme d'un vecteur codé à une chaleur de la forme du nombre d'états de 1 × (ici, 1 × 16) et qui sort un vecteur de la forme du nombre d'actions de 1 × (ici, 1 × 4). Le résultat est la valeur de Q pour toutes les actions. Avec la possibilité d'ajouter des couches cachées et des fonctions d'activation différentes, un réseau Q présente de nombreux avantages par rapport à un tableau Q. Contrairement à un tableau de Q, dans un réseau de Q, les valeurs de Q sont mises à jour en minimisant la perte par **backpropagation**. La fonction de perte est donnée par :

$$\begin{aligned} \text{Loss} &= \sum \left(Q_{\text{target}} - Q_{\text{predicted}}\right)^2 \\ Q(s,a)_{\text{target}} &= r + \gamma \max_{a'} Q\left(s', a'\right) \end{aligned}$$

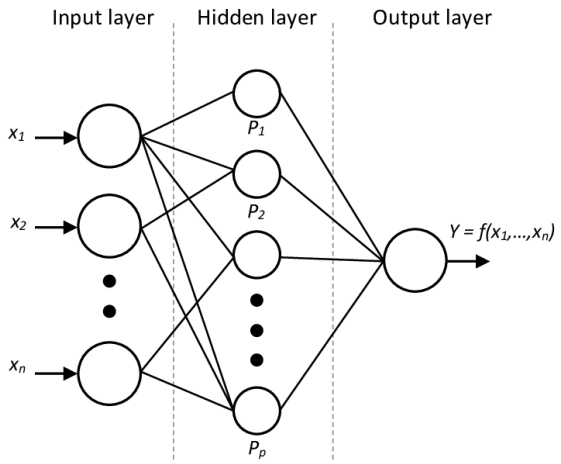

Figure 9. L'architecture du réseau, n = 16

4.4 Comparison entre Q-Table et Q-Network

Il y a un coût de stabilité associé à la fois au Q-learning et aux Q-networks. Il y aura des cas où avec l'ensemble donné d'hyperparamètres des valeurs Q ne convergent pas, mais avec les mêmes hyperparamètres, on assiste parfois à une convergence. Ceci est dû à l'instabilité de ces approches d'apprentissage.

Afin de résoudre ce problème, une meilleure politique initiale devrait être définie (ici, la valeur Q maximale d'un état donné) si l'espace d'état est petit. De plus, les hyperparamètres, notamment le taux d'apprentissage, les facteurs d'actualisation et la valeur epsilon, jouent un rôle important. Par conséquent, ces valeurs doivent être initialisées correctement.

En général, les réseaux Q offrent plus de flexibilité par rapport à l'apprentissage Q, en raison de l'augmentation des espaces d'état. Un réseau neuronal profond dans un Q-network pourrait permettre un meilleur apprentissage et de meilleures performances. Pour ce qui est de jouer à Atari en utilisant des Q-Networks profonds.

5 Processus de décision de Markov

5.1 Définitions

Le Processus de décision de Markov est un tuple $(S, A, \{P_{sa}\}, \gamma, R)$ avec :

- S :est un ensemble des **états**. (S est l'ensemble de toutes les possibilités).
- A : est un ensemble d' **actions**.
- $T(s, a, s^{'}) \sim P_{\text{sa}}$: sont les probabilités de transition d'état. Pour chaque état $s \in S$ et action $a \in A$, P_{sa} est une distribution sur l'espace d'état. Nous en dirons plus à ce sujet plus tard, mais en résumé, P_{sa} donne la distribution sur les états vers lesquels nous ferons la transition si nous effectuons l'action a dans l'état s.
- $\gamma \in [0, 1)$, est appelé le **facteur de réduction**.
- $R : S \times A \mapsto \mathbb{R}$ est la **fonction de récompense**. (Les récompenses sont parfois écrites en fonction d'un état S seulement, auquel cas nous aurions $R : S \mapsto \mathbb{R}$.
- $\pi(s) \to a$ est une fonction qui prend l'état comme entrée et produit l'action à entreprendre.

Idée générale sur la dynamique d'un MDP

Nous commençons dans un état s_0, et nous pouvons choisir une action $a_0 \in A$, en conséquence de notre choix, l'état du MDP transite aléatoirement vers un état successeur s_1, tiré selon $\mathbb{P}_{s_0 a_0}$. Ensuite, nous pouvons choisir un autre a_1, et ainsi de suite $\cdots$.

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \dots$$

Après avoir visité la séquence d'états $s_0, s_1, \cdots$ avec des actions $a_0, a_1, \cdots$ notre gain total est donné par :

$$R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \ldots$$

Ainsi, notre principal objectif dans l'apprentissage par renforcement est de choisir des actions au fil du temps de manière à maximiser la valeur attendue du gain total :

$$\mathrm{E}\left[R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \ldots\right]$$

5.2 Assomptions

La séquence des récompenses joue un rôle important dans la recherche de la politique optimale pour un problème MDP mais certaines hypothèses révèlent comment une séquence de récompenses met en œuvre le concept de récompenses retardées. met en œuvre le concept de récompenses différées.

Les horizons infinis

La première hypothèse est celle des horizons infinis, c'est-à-dire la quantité infinie de pas de temps pour atteindre l'état cible à partir de l'état de départ. Par conséquent,

$$\pi(s) \rightarrow a$$

La fonction de politique ne prend pas en compte les étapes de temps restantes. Si l'horizon avait été fini, la politique aurait été la suivante ,

$$\pi(s, t) \rightarrow a$$

où t est le nombre de pas de temps restant pour accomplir la tâche. Par conséquent, sans l'hypothèse de l'horizon infini, la notion de politique ne serait pas stationnaire, c'est-à-dire $\pi(s) \rightarrow a$, elle serait plutôt $\pi(s, t) \rightarrow a$.

Utilité des séquences

L'utilité des séquences se réfère à la récompense globale reçue lorsque l'agent passe par des séquences d'états. les séquences d'états. Elle est représentée par $U(s_0, s_1, s_2 \cdots)$ où $s_0, s_1, s_2 \cdots$ représente la séquence d'états.

La deuxième hypothèse est que s'il existe deux utilités, $U(s_0, s_1, s_2 \cdots)$ et $U(s_0^{'}, s_1^{'}, s_2^{'} \cdots)$ telles que l'état de départ des deux séquences est le même et,

$$U(s_0, s_1, s_2 \cdots) > U(s_0^{'}, s_1^{'}, s_2^{'} \cdots)$$

alors

$$U(s_1, s_2 \cdots) > U(s_1^{'}, s_2^{'} \cdots)$$

Cette hypothèse est appelée **le stationnaire des préférences**, et l'équation suivante satisfait à cette hypothèse.

$$U(s_0, s_1, s_2 \cdots) = \sum_{t=0}^{\infty} y^t R(s_t)$$

5.3 Les équations de Bellman

Puisque la politique optimale π^* est celle qui maximise les récompenses attendues, donc,

$$\pi^* = \mathrm{argmax}_\pi \, E\left[\sum_{\infty}^{t=0} \gamma^t R(s_t) \,|\pi\right]$$

avec $E\left[\sum_{\infty}^{t=0} \gamma^t R(s_t) \,|\pi\right]$ signifie la valeur attendue des récompenses obtenues à partir de la séquence d'états que l'agent observe s'il suit la politique π. Ainsi, argmax_π produit la politique π qui a la récompense attendue la plus élevée.

De même, nous pouvons aussi calculer **l'utilité de la politique d'un état**, c'est-à-dire que si nous sommes à l'état état s, étant donné une politique, alors, l'utilité de la politique pour l'état s, c'est-à-dire, $U^{\pi}(s)$ serait est la récompense attendue à partir de cet état :

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^{t} R\left(s_{t}\right) | \pi, s_{0} = s\right]$$

La récompense immédiate de l'état, c'est-à-dire $R(s)$ est différente de l'utilité de l'état $U(s)$) (c'est-à-dire l'utilité de la politique optimale de l'état $U^{\pi}(s)$) en raison de la notion de récompenses différées. A partir de maintenant, l'utilité de l'état $U(s)$ se référera à l'utilité de la politique optimale de l'état, c'est-à-dire l'état $U^{\pi^{*}}(s)$.
De plus, la politique optimale peut également être considérée comme celle qui maximise l'utilité attendue. l'utilité attendue. Par conséquent,

$$\pi^{*} = \text{argmax}_{a} \sum_{s'} T\left(s, a, s'\right) U\left(s'\right)$$

où $T(s, a, s^{'})$ est la probabilité de transition, c'est-à-dire $P(s^{'}|s, a)$ et $U(s^{'})$ est l'utilité du nouvel état d'atterrissage après que l'action a ait été effectuée. nouvel état d'atterrissage après la réalisation de l'action a sur l'état s.
$T\left(s, a, s'\right) U\left(s'\right)$ fait référence à la somme de tous les résultats possibles du nouvel état pour une action particulière, alors l'action qui donne la valeur maximale de $T\left(s, a, s'\right) U\left(s'\right)$ qui est considérée comme faisant partie de la politique optimale et ainsi, l'utilité de l'état 's' est donnée par l'**équation de Bellman** suivante,

$$U(s) = R(s) + \gamma \max_{a} \sum_{s'} T\left(s, a, s'\right) U\left(s'\right)$$

avec, $R(s)$ est la récompense immédiate et
$\max_{a} \sum_{s'} T\left(s, a, s'\right) U\left(s'\right)$ est la récompense du futur, c'est-à-dire l'utilité actualisée de l'état s que l'agent peut atteindre à partir de l'état s donné si l'action, a, est entreprise.

Résolution des équations Bellman

itération des valeurs :

Disons que nous avons quelques n états dans l'environnement donné et si nous voyons l'équation de Bellman,

$$U(s) = R(s) + \gamma \max_{a} \sum_{s'} T\left(s, a, s'\right) U\left(s'\right)$$

on découvre que n **états** sont donnés ; donc, on aura n **équations** et n inconnues mais la fonction max_{a} la rend **non linéaire**. Ainsi, nous ne pouvons pas les résoudre comme des équations linéaires.
Alors ,pour les résoudre :

1. Commencez par une utilité arbitraire
2. Mettre à jour les utilités en fonction du voisinage jusqu'à convergence, c'est-à-dire mettre à jour l'utilité de l'état en utilisant l'équation de Bellman en fonction des utilités des états d'atterrissage à partir de l'état donné.

Répétez cette opération plusieurs fois pour aboutir à la véritable valeur des états. Ce processus d'itération vers convergence vers la vraie valeur de l'état est appelé **valeur itération**.

Pour les états terminaux où le jeu se termine, l'utilité de ces états terminaux est égale à la récompense immédiate que l'agent reçoit en entrant dans le jeu. récompense immédiate que l'agent reçoit en entrant dans l'état terminal.

Itération des Stratégies :

Le processus d'obtention de l'utilité optimale par itération sur la politique et mise à jour de la politique elle-même au lieu de la valeur jusqu'à ce que la politique converge vers l'optimum est appelé itération de politique. Le processus d'itération de la politique est le suivant :

1. Commencer avec une stratégie arbitraire' π_0
2. Pour π_t stratégie a l'étape t, calculer $U_t = U_t^{\pi}$ en utilisant :

$$U_t(s) = R(s) + \gamma \sum_{s'} T\left(s, \pi_t(s), s'\right) U_{t-1}\left(s'\right)$$

3. Améliorer la stratégie π_{t+1} avec :

$$U_t(s) = R(s) + \gamma \sum_{s'} T\left(s, \pi_t(s), s'\right) U_{t-1}\left(s'\right)$$

5.4 Formation de l'environnement FrozenLake-v0 en utilisant le MDP

Il s'agit d'un environnement de grille dans OpenAI gym appelé FrozenLake-v0, discuté précédemment. Nous avons mis en œuvre Q-learning et Q-network pour comprendre l'environnement d'OpenAI gym. Maintenant, nous allons implémenter **value itération** pour obtenir la valeur d'utilité de chaque état dans l'environnement FrozenLake-v0. Soit la représentation de l'état comme suit :

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

6 Simulation De Voiture Autonome

6.1 Mise en contexte

L'idée est d'implémenter une simulation d'un véhicule apprenant la pratique d'éviter les collisions via le QLearning. Le choix est pertinent en raison de l'actualité du sujet ainsi que l'approche QLearning qui convient le mieux à ce type de problème. Notre exemple est pertinent quant à l'évaluation de l'efficacité et la fiabilité de l'approche QLearning dans ce champs d'application, en l'occurrence, les voitures autonome.

Au final, on va avoir un circuit et une simulation de véhicule qui au fur et à mesure qu'il avance dans ses tests, va améliorer sa conduite. On essayera de quantifier la précision ainsi que la fiabilité du modèle.

Il existe de nombreux exemples d'entrainement par QLearning sur internet ayant démontré tant la fiabilité que l'efficacité pour des jeux Atari. Les exemples travaillent directement avec des images Atari brutes. Dans notre exemple, on voudrait introduire le concept au domaine des jeux d'apprentissage de conduite au sein d'un circuit prédéfini. Le jeu sera conçu à l'aide de Unity/C#.

Cependant, Unity expose a une limite quant à la disponibilité des librairies servant d'appliquer le Reinforcement Learning. On a opté de diviser l'application en deux partie : partie d'entrainement du modèle via une interface Python et la partie graphique à l'aide de Unity/c#.

Les deux composantes du projet communiquent à l'aide du protocole TCP/IP. Un module est implémenté pour les deux composantes afin d'établir la communication tel que montré dans la figure 10

Figure 10. *Architecture de l'implémentation*

6.2 Méthodologie

L'idée du projet est d'implémenter une simulation de véhicule devant respecter le chemin défini par un circuit tel que montré dans la figure 11. Pour atteindre notre objectif, on a scindé le projet en deux implémentations séparées :

— Partie graphique : concue à l'aide de Unity/C#

— Partie d'entrainement : concue avec Python et les librairies d'apprentissage.

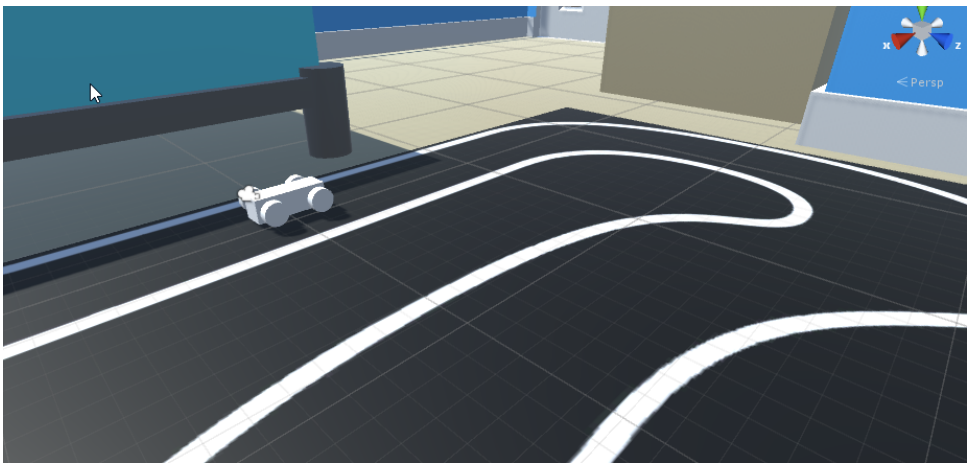

Figure 11. *Simulation du jeu*

De ce qui est la partie graphique du simulateur, on a commencé par construire la simulation de la piste (le chemin de la course) tel que montré dans la figure 12 à l'aide d'un simple dessinateur.

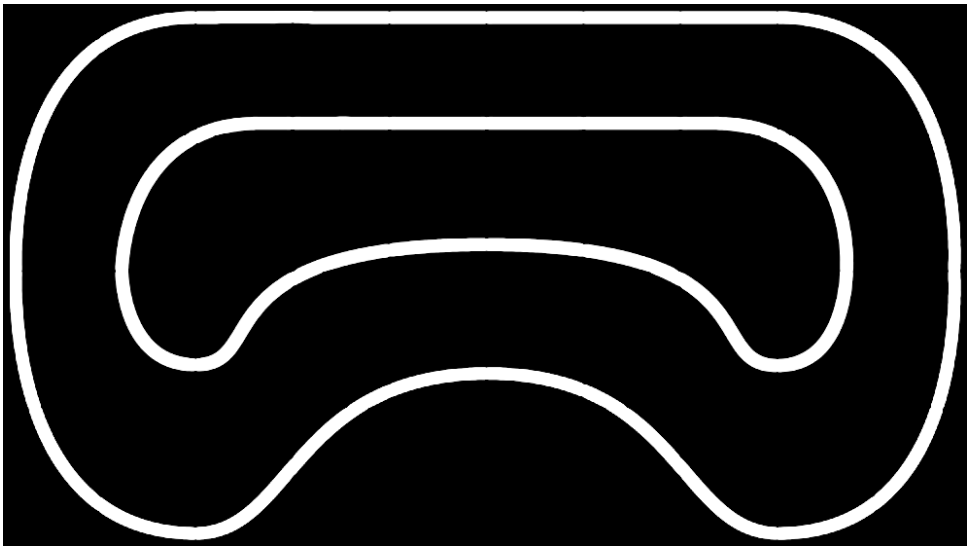

Figure 12. *Illustration de la piste*

Ensuite, les bases des composantes graphiques sont construites et intégrées à l'aide de Unity. le reste de l'implémentation est réalisé selon l'approche Orienté Objet tel que montré dans le diagramme de class illustré dans la figure 13.

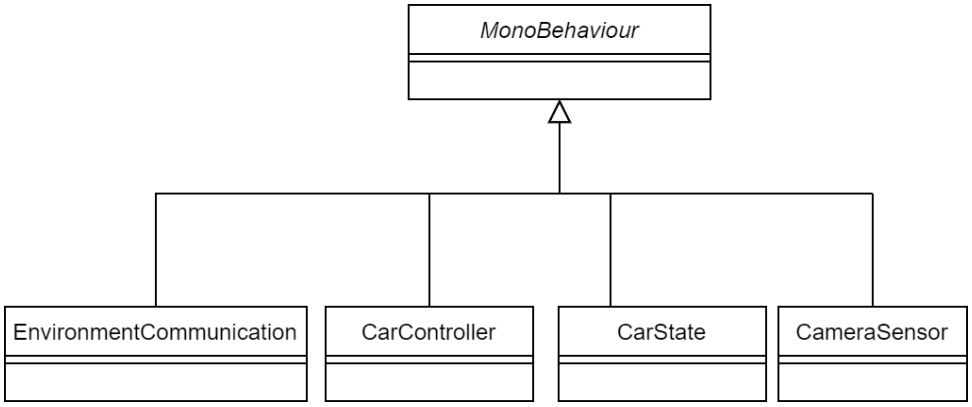

Figure 13. *Le diagramme de classe*

— **EnvironnementCommunication :** est l'implémentation de l'instance de communication TCP/IP avec le programme Python afin d'accéder aux ressource du controlleur du jeu.

— **CarController :** est la classe permettant d'appliquer les différentes actions sur le simulateur du véhicule.

— **CarState :** permet la gestion de l'état du simulateur du véhicule.

— **CameraSensor :** permet de gérer l'instance de la visualisation de l'environnement.

Quant à la partie de l'entrainement conçu à l'aide de Python, elle est composée de différents modules que l'on a implémenté et montré dans la figure 14

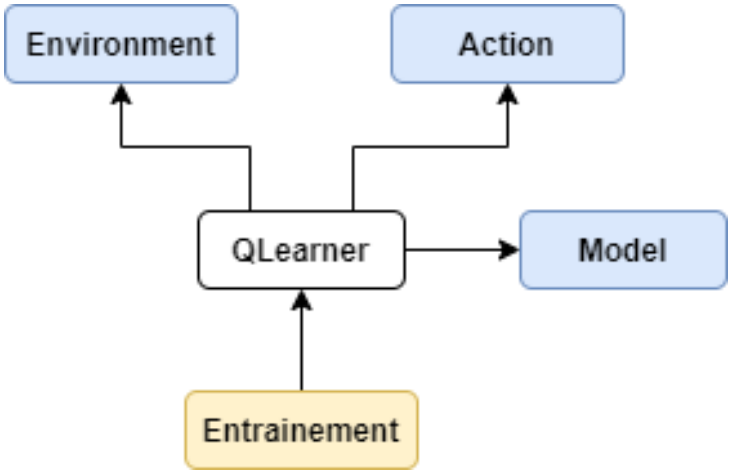

Figure 14. *Le module de l'entrainement*

6.3 Évaluation et expérimentation

Nous avons effectué des expériences différentes pour voir la convergence du modèle. On procède ensuite à l'évaluation des agent réels et l'environnement, nous avons apporté une modification à la structure de récompense des jeux uniquement pendant l'entraînement. Nous avons fixé toutes les récompenses positives à 0.1 et toutes les récompenses négatives à -0.1, laissant 0 récompenses inchangées(voir les sections précédentes pour en savoir davantage). Le découpage des récompenses de cette manière limite l'échelle des dérivées d'erreur et facilite l'utilisation du même taux d'apprentissage.

Dans le même temps, cela pourrait affecter les performances de notre agent car il ne peut pas faire la différence entre des récompenses d'ampleur différente. Dans ces expériences, nous avons utilisé l'algorithme RMSProp avec des minibatchs de taille 1. Plus précisément, l'agent voit et sélectionne des actions sur chaque k^{me} image au lieu de chaque image, et sa dernière action est répétée sur les images ignorées. Étant donné que l'exécution du simulateur pendant une étape nécessite beaucoup moins de calculs que

Table 1. Les résultats obtenus

Épisode	Récompenses	Temps de succès
10	1.26	3
20	1.82	6
30	0.65	3
60	2.35	6
80	3.65	16
100	4.21	22
120	1.21	21
130	4.19	20
150	4.98	23
160	5.36	23
184	5.16	25

la sélection d'une action par l'agent, cette technique permet à l'agent de jouer environ k fois plus de jeux sans augmenter considérablement le temps d'exécution. Nous utilisons k = 4.

Dans l'apprentissage supervisé, on peut facilement suivre les performances d'un modèle pendant la formation en l'évaluant sur les ensembles de formation et de validation. Dans l'apprentissage par renforcement, cependant, évaluer avec précision les progrès d'un agent pendant la formation peut être difficile. Puisque notre métrique d'évaluation, est la récompense totale que l'agent collecte dans un épisode nous la calculons périodiquement pendant l'entraînement. La mesure de la récompense totale moyenne a tendance à être très bruyante, car de petits changements dans les pondérations d'une politique peuvent entraîner de grands changements dans la répartition des États que la politique visite.

Le tableau 1 montre les résultats obtenus suite à nos test. On remarque nettement que le temps de succès(correspondant à la durée que le simulateur du véhicule avance vers son objectif sans collision dans son chemin) augmente de façon significative au fur et à mesure que l'apprentissage avance en épisode. Celà est fortement corrélé sur le taux de récompenses qui tend également à augmenter conjointement avec le temps de succès et conséquemment avec les épisodes réalisé. En terme de l'experience, le véhicule réussit à réaliser tout son chemin de manière entièrement autonome.

Les résultats du tableau 1 montrent comment la récompense totale moyenne évolue pendant l'entraînement. Les récompenses moyennes sont en effet assez bruyantes, ce qui donne l'impression que l'algorithme d'apprentissage ne progresse pas régulièrement. Une autre mesure, plus stable, est la durée de succès, qui fournit une estimation du montant de la récompense actualisée que l'agent peut obtenir en suivant sa politique à partir de n'importe quel état donné. Nous n'avons rencontré aucun problème de divergence dans aucune de nos expériences. Cela suggère que, malgré l'absence de garanties de convergence théorique, notre méthode est capable d'entraîner de grands réseaux de neurones en utilisant un signal d'apprentissage par renforcement et une descente de gradient stochastique de manière stable.

7 Conclusion et perspectives

L'apprentissage par renforcement est un problème complexe. Dans notre travail, on a répondu amplement aux objectifs annoncés dans le travail préliminaire avec un preuve de concept qui a su montré l'efficacité de l'approche proposée. Notre implémentation a permis de valider l'efficacité et la fiabilité le l'algorithme QLearning choisi qui reste le plus convenable à notre simulation.

Le modèle adopté reste dépendant des paramètres. Cependant, suite à une multitude de tests, on est arrivé à saisir empiriquement les meilleurs paramètres permettant l'optimiser le rendement de l'entrainement par le QLearning.

On suggère que des travaux poussés soient envisagés dans l'optique de voir dans quelle mesure l'amélioration de la modalité de récompense pourrait influencer positivement la convergence de l'algorithmique et pourrait ainsi apporter une amélioration des résultats de performance.

Références

[1] Richard S. Sutton and Andrew G. Barto [2018], Reinforcement Learning : An Introduction, *MIT Press, Cambridge, MA*.

[2] Silva et. al, Reinforcement Learning, *UCL*.

[3] White et. al, Fundamentals of Reinforcement Learning, *University of Alberta*.

[4] Sayon Dutta - Reinforcement Learning with TensorFlow A beginner's guide to designing self-learning systems with TensorFlow and OpenAI Gym-Packt Publishing (2018).

[5] **Artificial general intelligence** on Wikipedia.

[6] **openai-gym** official gym-openai library description.

[7] Aggarwal, C.C. (ed.) [2015], *Data Classification*, CRC Press.

[8] Aggarwal, C.C., Reddy, C.K. (eds.) [2015], *Data Clustering*, CRC Press.

[9] Wood, G. [2016], **In Two Moves, AlphaGo and Lee Sedol Redefined the Future**, Wired Magazine.

[10] **Spinning Up**, on OpenAI.com.

[11] `Q-learning` : classic paper by Tsitsiklis & van Roy.

[12] `A2C/A3C` (Asynchronous Advantage Actor-Critic) : Mnih et al, 2016

[13] `PPO` (Proximal Policy Optimization) : Schulman et al, 2017

[14] `TRPO` (Trust Region Policy Optimization) : Schulman et al, 2015

[15] `DDPG` (Deep Deterministic Policy Gradient) : Lillicrap et al, 2015

[16] `TD3` (Twin Delayed DDPG) : Fujimoto et al, 2018

[17] `SAC` (Soft Actor-Critic) : Haarnoja et al, 2018

[18] `DQN` (Deep $Q-$Networks) : Mnih et al, 2013

[19] `C51` (Categorical 51$-$Atom DQN) : Bellemare et al, 2017

[20] `QR-DQN` (Quantile Regression DQN) : Dabney et al, 2017

[21] `HER` (Hindsight Experience Replay) : Andrychowicz et al, 2017

[22] `WorldModels` : Ha and Schmidhuber, 2018

[23] `I2A` : (Imagination-Augmented Agents) Weber et al, 2017

[24] `MBMF` (Model-Based RL with Model-Free Fine-Tuning) : Nagabandi et al, 2017

[25] `MBVE` (Model-Based Value Expansion) : Feinberg et al, 2018

[26] `AlphaZero` : Silver et al, 2017

[27] Patel, S., Pourhasan, R., Boily, P. [in preparation], *Deep Learning and Applications*, Data Science Report Series, Data Action Lab.

[28] Kansal, S., Martin, B., **Reinforcement Q-Learning from Scratch in Python with OpenAI Gym**, LearnDataSci.com.

[29] Phadnis, A. [2018], **A3C – What is it, and What I Built**, on GoodAudience.com.

[30] Szita, I. [2012], *Reinforcement Learning in Games*, in Reinforcement Learning : State-of-the-Art, Wiering, M and can Otterlo, M. (eds.), Springer Verlag.

[31] Heinz, S. [2019], **Using Reinforcement Learning to Play Super Mario Bros. on NES Using TensorFlow**, on towardsdatascience.com.

[32] Jung, A. [2016], **AI playing Super Mario World with Deep Reinforcement Learning**, on YouTube.com.

[33] Zychlinski, S. [2019], **Qrash Course: Reinforcement Learning 101 & Deep Q Networks in 10 Minutes**, on towardsdatascience.com.

[34] Mahmood, R., Korenkevych, D., Komer, B., Bergstra, J. [2018], **Setting up a Reinforcement Learning Task with a Real-World Robot**, on YouTube.com. Paper at **https://arxiv.org/abs/1803.07067**.

[35] [2008], **Reinforcement learning for a robotic soccer goalkeeper**, by YouTube user africlean.