UMONS
Université de Mons

Faculté Polytechnique

RL2 - Q-learning avec Gymasium (Open AI Gym)

Advanced Machine Learning

Rapport

Master Ingénieur Civil en Informatique et Gestion

Anthony MOULIN
Joakim LEFEBVRE

POLYTECH MONS

pôle hainuyer

Table des matières

1 Introduction **2**

2 Description de l'environnement **2**
2.1 Jeu : Freeway d'Atari 2
2.2 Modes et Difficultés 2
2.3 Actions 4

3 Methodologie **4**
3.1 Image d'entrée 4
3.2 Modèle 5
3.3 Reward policy 6
3.4 Fonction objectif 6
3.5 Jeu d'expérience 6
3.6 Gèle du q-network 7
3.7 Batch_size 7
3.8 Exploration / exploitation 7
3.9 Action répétée 7
3.10 Évaluation 8

4 Résultats **9**

5 Conclusion **10**

1 Introduction

Ce projet explore le domaine fascinant de l'apprentissage par renforcement en utilisant OpenAI Gym pour développer un algorithme d'apprentissage profond. Notre défi : créer un modèle qui maîtrise le jeu Freeway d'Atari, en utilisant exclusivement les pixels en entrée.

Le choix de ce jeu offre un terrain d'expérimentation riche pour évaluer les capacités des algorithmes d'apprentissage par renforcement. En restreignant notre agent à interpréter les pixels, notre approche vise à tester sa faculté à apprendre des stratégies complexes. Pour cela, nous avons opté pour le deep Q-learning.

Cette introduction établit le cadre du projet en offrant un aperçu concis de l'apprentissage par renforcement, du choix du jeu, et des défis uniques. Le rapport détaillera ensuite notre méthodologie, l'entraînement et l'analyse des résultats, contribuant ainsi à enrichir notre compréhension des capacités de l'apprentissage profond dans le contexte des jeux vidéo classiques.

2 Description de l'environnement

Dans cette section, nous explorerons en détail l'environnement dans lequel notre agent d'apprentissage par renforcement évolue. Nous nous concentrerons sur les caractéristiques du jeu Freeway d'Atari, mettant en lumière les aspects essentiels nécessaires à la compréhension de notre approche.

2.1 Jeu : Freeway d'Atari

Nous avons choisi le jeu Freeway parce qu'il nous rappelle le jeu auquel nous jouions petits, Crossy Road, sur téléphone. Dans ce jeu, notre agent, représenté par un canard, doit naviguer à travers une route en évitant le trafic routier. À l'origine, sur les consoles Atari, ce jeu était un jeu à deux joueurs, ce qui signifie que le but était d'avoir plus de points que son adversaire à la fin de la partie. Une partie dure 2 minutes et 16 secondes. Comment gagne-t-on des points ? Pour cela, il suffit de réussir à traverser la route en entier, ce qui nous rapporte 1 point, puis on recommence en bas de la route.

2.2 Modes et Difficultés

L'environnement d'OpenAI Gymnasium [1] comprend plusieurs types d'environnements ainsi que différents modes et difficultés pour le jeu Freeway.

Env-id	obs_type	frameskip	repeat_action_probability
Freeway-v0	"rgb"	(2, 5)	0.25
Freeway-ram-v0	"ram"	(2, 5)	0.25
Freeway-ramDeterministic-v0	"ram"	4	0.25
Freeway-ramNoFrameskip-v0	"ram"	1	0.25
FreewayDeterministic-v0	"rgb"	4	0.25
FreewayNoFrameskip-v0	"rgb"	1	0.25
Freeway-v4	"rgb"	(2, 5)	0.0
Freeway-ram-v4	"ram"	(2, 5)	0.0
Freeway-ramDeterministic-v4	"ram"	4	0.0
Freeway-ramNoFrameskip-v4	"ram"	1	0.0
FreewayDeterministic-v4	"rgb"	4	0.0
FreewayNoFrameskip-v4	"rgb"	1	0.0
ALE/Freeway-v5	"rgb"	4	0.25
ALE/Freeway-ram-v5	"ram"	4	0.25

Table 1 – Configurations des environnements de Freeway Gymnasium

Conformément au tableau 1, il existe deux types d'observations possibles : "rgb", qui correspond à une image en couleur de notre environnement de taille (210, 160, 3), et "ram", qui correspond à une liste de taille 128 contenant des informations sur l'environnement telles que la position en y du canard, la position des voitures, l'information sur le contact du canard avec une voiture, etc. Le frameskip correspond au nombre d'images qui s'écoulent avant de pouvoir refaire une action. Dans certains environnements, il est fixé, tandis que dans d'autres, il varie. Enfin, le repeat_action_probability correspond à la probabilité de répéter l'action précédente.

Modes disponibles	Mode par défaut
[0, ..., 7]	0

Table 2 – Configurations des modes de Freeway Gymnasium

Au niveau du tableau 2, on peut constater qu'il existe différents modes dans le jeu. Ceux-ci vont faire varier la forme des voitures, leur vitesse et leur nombre.

Difficultés disponibles	Difficulté par défaut
[0, 1]	0

Table 3 – Configurations des difficultés de Freeway Gymnasium

Au niveau du tableau 3, il n'y a uniquement deux types de difficultés. Le niveau facile est lorsque son paramètre est mis sur 0, celui-ci fait en sorte que lors d'une collision avec une voiture, le canard recule un peu et est immobilisé pendant un court laps de temps. Pour le niveau difficile, le paramètre égal à 1, lors de la collision, le canard est téléporté et immobilisé en bas de la route et doit complètement recommencer.

2.3 Actions

Les actions disponibles pour le jeu Freeway sont référencées dans le tableau 4.

Valeur	Signification
0	Ne rien faire
1	Avancer
2	Reculer

TABLE 4 – Configurations des actions de Freeway Gymnasium

Cette description approfondie de l'environnement pose les bases nécessaires pour comprendre le contexte dans lequel notre algorithme d'apprentissage par renforcement sera entraîné.

3 Methodologie

3.1 Image d'entrée

Pour l'image d'entrée, représentée à la figure 1, on observe qu'elle contient beaucoup d'informations inutiles, telles que les contours noirs, le score et le logo Activision. Afin d'améliorer la précision, nous avons décidé de recadrer l'image pour ne montrer que ce qui est réellement utile. Ensuite, pour éviter d'utiliser une image trop grande dans notre modèle, ce qui nécessiterait un traitement d'analyse plus intensif, nous avons décidé de la redimensionner en (84,84,3). Cependant, nous avons constaté une perte d'information due à ce redimensionnement. Comme le montre la figure 2a, les voitures n'ont pas toutes la même taille, et certaines sont moins visibles que d'autres. Lors de nos tests ultérieurs, nous avons remarqué que notre modèle avait du mal à percevoir la taille réelle des voitures, le canard percutant la voiture sur les bords de celle-ci.

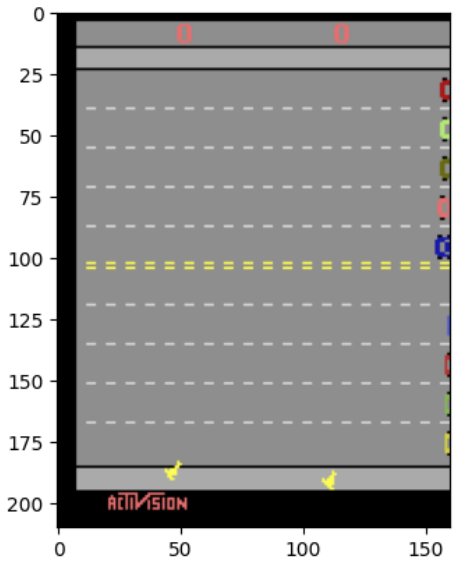
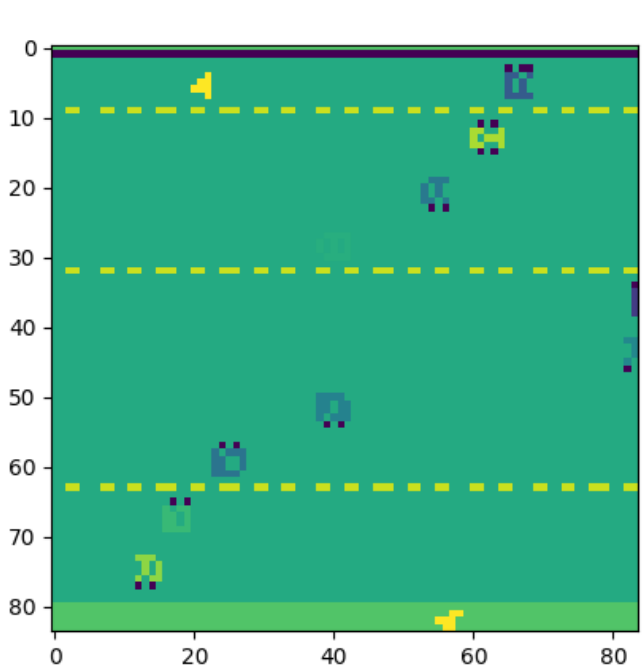


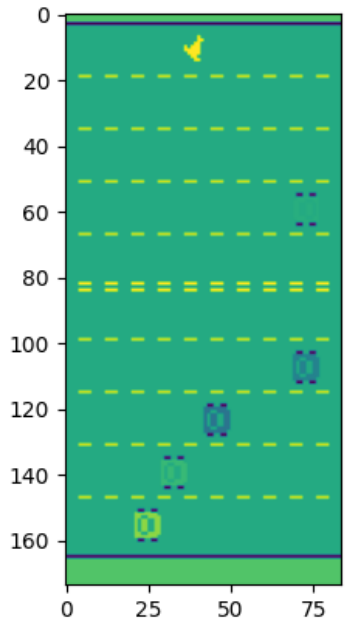
FIGURE 1 – Affichage de la sortie de l'environnement

Afin de résoudre ce problème, nous avons décidé de conserver la taille d'origine de l'image après recadrage (174,84,3). Cela signifie que nous ne redimensionnons pas l'image en hauteur (174), et en largeur (84), nous ne prenons pas toute la largeur de l'écran. Le canard peut voir au maximum 42 pixels à sa gauche, et nous avons décidé qu'il ne verrait que 42 pixels à sa droite. De cette manière, nous obtenons une image plus petite que l'image d'origine, qui conserve néanmoins toutes les informations nécessaires, comme illustré à la figure 2b.

L'étape suivante a été de convertir cette image en niveau de gris pour encore réduire sa taille. Ainsi, notre image finale a une taille de (174,84,1). Enfin, nous avons effectué une normalisation de l'image pour uniformiser nos données et accélérer l'entraînement de notre modèle.



(a) Image redimensionnée en 84x84



(b) Image non redimensionnée et conservée en 174x84

FIGURE 2 – Affichage de la sortie de l'environnement après traitement

3.2 Modèle

Concernant notre modèle, étant donné que nous analysons des images, notre modèle est un CNN. En entrée de celui-ci, l'image est de taille (174,84,4). Cependant, au préalable, nous avons expliqué que nous prétraitons l'image de l'environnement pour qu'elle soit de type (174,84,1). Ici, la dimension 4 au lieu de 1 correspond à 4 images consécutives de l'environnement. Cela signifie que nous conservons les images de l'environnement à l'instant t ainsi que les trois précédentes (t-1, t-2, t-3). Nous concaténons ensuite ces quatre images pour n'en former qu'une qui sera l'entrée de notre modèle. De cette façon, nous permettons au modèle d'avoir une notion temporelle des images, car avec une seule image, nous ne pouvons pas connaître la vitesse ni la direction des voitures.

```python
def create_q_model(num_actions):
    # Network defined by the Deepmind paper
    inputs = layers.Input(shape=(174, 84, 4,))

    # Convolutions on the frames on the screen
    layer1 = layers.Conv2D(32, 8, strides=4, activation="relu")(inputs)
    layer2 = layers.Conv2D(64, 4, strides=2, activation="relu")(layer1)
    layer3 = layers.Conv2D(64, 3, strides=1, activation="relu")(layer2)

    layer4 = layers.Flatten()(layer3)

    layer5 = layers.Dense(512, activation="relu")(layer4)
    action = layers.Dense(num_actions, activation="linear")(layer5)

    return keras.Model(inputs=inputs, outputs=action)
```

FIGURE 3 – Architecture du modèle (CNN)

Ensuite, pour le reste de l'architecture, représenté à la figure 3, nous avons choisi un modèle assez simple, en nous basant sur l'article [2] pour sa création. Au niveau de l'optimiseur, nous avons décidé d'utiliser RMSprop, souvent utilisé pour les entraînements de type DQN.

3.3 Reward policy

Pour la "reward policy" lors de l'entraînement, nous avons décidé de prendre en compte trois paramètres :

- Une récompense lorsque le canard arrive en haut du jeu et est téléporté tout en bas.
- Une récompense lorsque le canard percute une autre voiture.
- Une récompense lorsque le canard n'avance pas.

Le jeu Freeway donne un point à chaque fois qu'il atteint le sommet de la route, ce qui est équivalent à ce que nous avons représenté dans la première récompense de notre politique. Lors de nos tests, nous avons veillé à ne prendre que des récompenses comprises dans l'intervalle [-1, +1]. Cela permet d'empêcher les q-values de devenir trop grandes et assure que les gradients sont bien conditionnés.

3.4 Fonction objectif

La fonction objectif utilisée est définie comme suit, conformément au cours [3] :

$$L(w) = E\left[\left(\underbrace{R(s,a,s') + \gamma \max_{a'} q(s',a',w)}_{\text{target}} - \underbrace{q(s,a,w)}_{\text{network}}\right)^2\right] \tag{1}$$

Ici, s correspond à l'état à l'instant t, a à l'action choisie, q correspond à la prédiction du modèle avec w ses poids, R correspond à la récompense, et les $'$ correspondent à l'instant $t+1$. Gamma (γ) est un paramètre que nous pouvons modifier à notre guise ; il correspond au poids que l'on accorde à la récompense à long terme par rapport à la récompense instantanée. Si $\gamma = 0$, on n'accorde aucune importance à la récompense à long terme. Bien évidemment, si on arrive à la fin du jeu, donc au dernier état (s), on ne prend pas en compte la récompense future au niveau de la target puisque (s') n'existe pas.

Ensuite, on calcule la loss que pour l'action choisie :

$$L(w) = \begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \tag{2}$$

Et seul le poids correspondant à l'action choisie sera modifié lors du calcul de la perte. Cette fonction objectif nous a donc permis de calculer les backpropagation du modèle avec la descente du gradient.

3.5 Jeu d'expérience

Nous avons établi un jeu d'expérience fixé à une taille de 10 000 qui comprend l'état de l'instant t et $t+1$, ainsi que l'action et la récompense de l'instant t (s_t, a_t, r_t, s_{t+1}). Une fois le jeu d'expérience créé, il est important de le mélanger aléatoirement. L'objectif d'utiliser un jeu d'expérience est d'éviter les corrélations dans les données. Sinon, le modèle pourrait apprendre par cœur sur les données, et ce n'est pas ce que nous voulons.

3.6 Gèle du q-network

Lors de l'entraînement, nous utilisons deux modèles à la place de 1, "model" et "model_target". Le "model" sert à prédire l'action sur les états à l'instant t (s), tandis que le "model_target" sert à prédire les états à l'instant t+1 (s'). Le "model" (modèle principal), lors de l'entraînement, est mis à jour à chaque époque (une époque correspond à une partie entière du jeu Freeway), tandis que le "model_target" lui n'est mis à jour que toutes les 3 époques (valeur que nous pouvons modifier). Lors de la mise à jour du "model_target", nous copions les poids du "model" pour les mettre dans celui de "model_target". L'avantage de travailler de cette manière permet d'éviter les oscillations et fixe les paramètres lors de l'entraînement d'un épisode. Encore une fois, cela permet d'éviter les corrélations entre les deux modèles pour un entraînement efficace.

3.7 Batch_size

Pour éviter de trop grands changements d'un coup dans les poids du modèle, nous avons établi un batch. Cela permet de mettre à jour notre modèle petit à petit pour qu'il puisse mieux converger. Dans notre cas, nous avons décidé d'utiliser un batch de taille 32. C'est-à-dire que lors de l'entraînement, les poids du modèle sont mis à jour sur 32 états.

3.8 Exploration / exploitation

Pour la répartition entre exploration et exploitation, nous avons décidé d'utiliser un epsilon qui diminue au cours du temps. L'epsilon représente le pourcentage entre 0 et 1 de faire une action aléatoire, sachant que 1 veut dire qu'on force une action aléatoire. Notre epsilon a été défini de cette manière :

$$\text{epsilon} -= \frac{(\text{epsilon\_max - epsilon\_min})}{\text{epsilon\_greedy\_epi}} \tag{3}$$

$$\text{epsilon} = \max(\text{epsilon}, \text{epsilon\_min}) \tag{4}$$

où epsilon_max = epsilon = 1.0, epsilon_min = 0.1 et epsilon_greedy_epi = nombre d'épisodes (époques) qu'il faut pour passer du epsilon_max à epsilon_min. Généralement, nous prenions :

$$\text{epsilon\_greedy\_epi} = \frac{\text{nombre d'épisode}}{2} \tag{5}$$

Epsilon est donc mis à jour à la fin de chaque épisode, c'est-à-dire après un jeu complet de 2 minutes et 16 secondes.

3.9 Action répétée

Nous avons utilisé une méthode d'apprentissage qui consiste à répéter des actions. Cela implique de répéter une même action pendant plusieurs images consécutives. Cette approche accélère l'apprentissage en permettant au modèle d'explorer plus efficacement les conséquences à long terme des actions. Cette technique stabilise l'entraînement en réduisant la variance des observations et capture de manière plus efficace les effets différés, améliorant ainsi la convergence du modèle dans des environnements où les récompenses sont rares ou retardées. Dans notre cas, cela permet au canard d'arriver en haut de la route plus rapidement ainsi que de faciliter l'esquive des voitures. La fréquence de répétition que nous avons choisie est de 4.

3.10 Évaluation

Pour évaluer correctement si notre modèle a bien appris à jouer aux jeux, nous utilisions plusieurs indices. Le premier consistait à regarder le graphique de la perte (loss) au cours du temps. Si celle-ci diminue, cela signifie que le modèle apprend correctement (il fait moins d'erreurs). Ensuite, nous examinions le graphique indiquant la somme des récompenses comprises dans notre jeu d'expérience. Si les récompenses augmentent dans le temps, c'est encore un signe positif que l'entraînement se passe bien. Pour le dernier graphique, nous analysons le score réel dans le jeu pour chaque épisode. Repris à la figure 4 et à la figure 5, nous pouvons constater à quoi ressemblent les différents graphiques pour un bon entraînement ainsi qu'un mauvais entraînement.

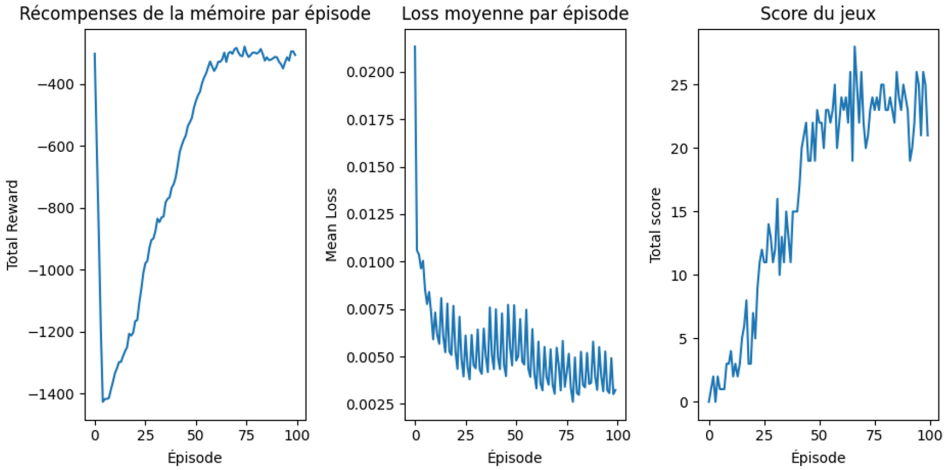


FIGURE 4 – Graphique utilisé pour l'entraînement du modèle : exemple d'un bon entraînement

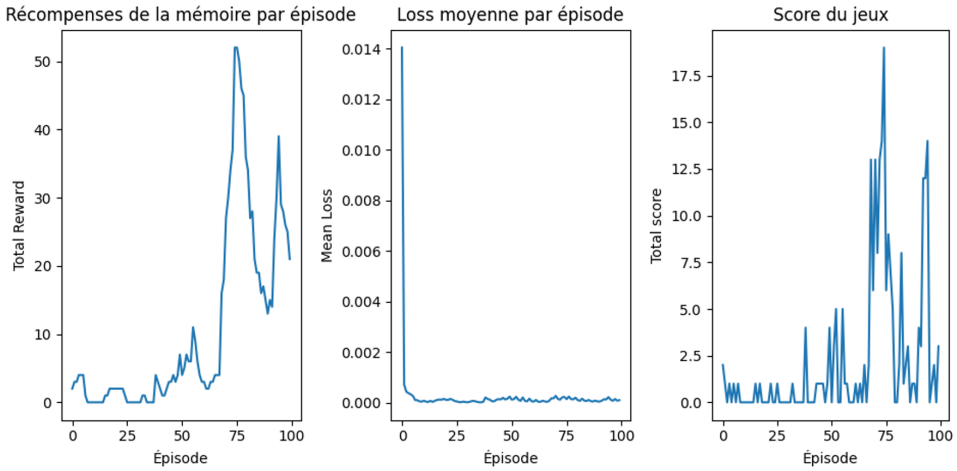


FIGURE 5 – Graphique utilisé pour l'entraînement du modèle : exemple d'un mauvais entraînement

Pour finir, il va de soi que nous testions notre modèle pour vérifier visuellement comment il se comporte et obtenir son vrai score dans le jeu.

4 Résultats

Pour l'entraînement de notre modèle, nous avons décidé d'utiliser l'environnement "ALE/Freeway-v5" avec le mode 0 et la difficulté facile (= 0). Les résultats sont regroupés dans le tableau 5 ci-dessous :

# action	**Reward**			**Gamma (γ)**	**Score**
	En haut	Collision	Ne pas bouger		
2	1	0	0	0.99	22
	1	-1	0	0.99	0
	1	**-1**	**-0.25**	**0.99**	**29**
	1	-1	-0.25	0.6	21
	1	-1	-0.25	0.3	23
3	1	-1	-0.25	0.99	**29**

TABLE 5 – Résultats des divers entrainements et tests sur le mode facile

Lors de nos entraînements, nous avons testé l'influence que les différents paramètres avaient sur les résultats de notre modèle. Pour chacun de nos entraînements, nous avons effectué 100 épisodes avec un epsilon_greedy_epi égal à 50. Cependant, pour le modèle comportant 3 actions, nous avons utilisé 300 épisodes avec un epsilon_greedy_epi égal à 150. Les raisons de cette différence seront expliquées dans la suite.

Tout d'abord, pour faciliter l'entraînement, nous avons jugé qu'il serait intéressant de réduire le nombre d'actions possibles. En effet, nous nous sommes interrogés sur l'intérêt qu'apportait l'action "ne rien faire" au jeu. Ainsi, nous avons étudié l'impact entre l'utilisation de 2 actions (avancer et reculer) et 3 actions (avancer, reculer et ne rien faire). Pour le même nombre d'épisodes, un modèle avec 2 actions a donné de meilleurs résultats qu'un modèle avec 3 actions. Il a donc été nécessaire d'ajouter 200 épisodes supplémentaires pour que le score du modèle à 3 actions soit similaire à celui du modèle à 2 actions. Cette différence s'explique aisément, car un bon modèle à 3 actions n'utilise presque jamais l'action "ne rien faire", démontrant ainsi son inutilité dans notre cas. Afin d'accélérer les entraînements, nous avons donc décidé de n'utiliser que 2 actions.

Ensuite, lorsque l'on utilise uniquement les récompenses disponibles dans le jeu, c'est-à-dire uniquement +1 lorsque le modèle est en haut, le modèle n'a aucune information sur s'il touche une voiture ou non, et si cela est bénéfique ou néfaste. En conséquence, l'agent (le canard) effectue uniquement des mouvements vers l'avant sans jamais reculer. Cela nous sert de référence pour évaluer si l'agent fait mieux qu'un score de 22, signifiant ainsi sa capacité à esquiver les voitures.

Dans l'idée de lui faire esquiver les voitures, nous avons ajouté la récompense négative de -1 lors d'une collision. Cependant, cela a eu pour conséquence que l'agent avait trop peur d'avancer et de toucher une voiture, ce qui le conduisait à rester immobile au départ. L'étape suivante a consisté à trouver une solution. Nous avons créé la récompense négative "ne pas bouger" pour l'empêcher de rester au départ et le forcer à avancer.

Au niveau des valeurs choisies pour chacune des récompenses, celles-ci ont été fixées après de nombreux essais-erreurs.

Pour la définition du gamma (γ) au début, nous souhaitions un gamma très élevé étant donné que la récompense positive est très éloignée dans le jeu. En effet, l'agent doit réussir à esquiver 10 voitures pour espérer obtenir une récompense positive. Cependant, par la suite, nous avons constaté que l'agent esquivait bien les voitures, mais parfois il était un peu trop pressé et touchait

la voiture sur son extrémité en voulant aller trop vite. C'est pourquoi nous avons eu l'idée de diminuer le gamma. Toutefois, nous constatons qu'en diminuant le gamma, cela n'a pas amélioré les résultats mais, au contraire, cela les a diminués.

Nous avons voulu tester le mode difficile sur nos modèles entraînés sur le mode facile. Les résultats sont regroupés dans le tableau 6 ci-dessous :

# action	**Difficulté**	**Reward**			**Gamma** (γ)	**Score**
		En haut	Collision	Ne pas bouger		
2	1	1	0	0	0.99	16
		1	-1	-0.25	0.99	**28**
	0	1	-1	-0.25	0.99	**29**

TABLE 6 – Résultats des tests sur le mode difficile

Encore une fois, la première ligne du tableau, avec un score de 16, correspond au modèle où l'agent va toujours tout droit sans jamais esquiver de voiture. Nous constatons néanmoins que notre meilleur modèle parvient à obtenir un excellent résultat de 28.

Nous pourrions également tester d'autres modes, mais ceux-ci sont beaucoup plus complexes et demandent donc beaucoup plus de temps d'entraînement. En effet, avec des voitures plus rapides, plus nombreuses et de tailles différentes, il devient difficile d'atteindre le haut de la route. C'est pourquoi nous n'avons utilisé que le mode 0, le mode le plus simple.

5 Conclusion

En conclusion, nous avons atteint l'objectif qui était de créer un algorithme d'apprentissage par renforcement pour le jeu Freeway d'Atari. Nous sommes plutôt satisfaits des résultats, avec un score maximum de 29. Cependant, nous constatons que le modèle n'est pas encore parfait, car il percute encore des voitures. L'entraînement pour le DQN est très long et prend du temps, mais nous pourrions encore ajuster les paramètres et effectuer de nouveaux tests pour tenter d'améliorer ce dernier.

Nous avons construit un code général, c'est-à-dire qu'il peut facilement être adapté à d'autres types de jeux d'Atari de Gymnasium. Pour cela, il suffirait de changer la "reward policy", de cropper correctement l'image d'entrée et de spécifier le nombre d'actions propres à l'environnement.

Annexes

Le code source complet de l'implémentation se trouve sur GitHub :
`https://github.com/Riipou/Q-learning-avec-Gymasium-Open-AI-Gym-`

Vous pouvez également consulter une vidéo sur YouTube pour visualiser notre meilleur modèle : `https://youtu.be/xaYKJYwxMpo`

Références

[1] *Freeway de Atarin, Gymnasium.* URL : https://gymnasium.farama.org/environments/atari/freeway/.

[2] Matteo Hessel et al. « Rainbow : Combining Improvements in Deep Reinforcement Learning ». In : (2017). DOI : https://arxiv.org/pdf/1710.02298.pdf.

[3] Xavier Siebert. *Advanced Machine Learning.* Faculté Polytechnique de Mons. 2024.