Report - RL Project Jumbo Mana

Matthieu SCHWARTZ

December 29, 2024

1 Introduction

1.1 Description du problème

Le but de ce projet est de concevoir un agent d'intelligence artificielle capable de naviguer sur une grille de jeu de taille 10×10. L'agent (le héros) doit apprendre à atteindre un trésor à partir d'une position de départ aléatoire tout en évitant des monstres qui se déplacent dynamiquement sur la grille. Si le héros se trouve sur une case adjacente à un monstre, il perd la partie. Les contraintes principales incluent :

- Utilisation de l'algorithme de Q-Learning, implémenté via la bibliothèque Stable Baselines3.
- Ajout de visualisations pour démontrer le comportement de l'agent.
- Gestion des mouvements dynamiques des monstres.

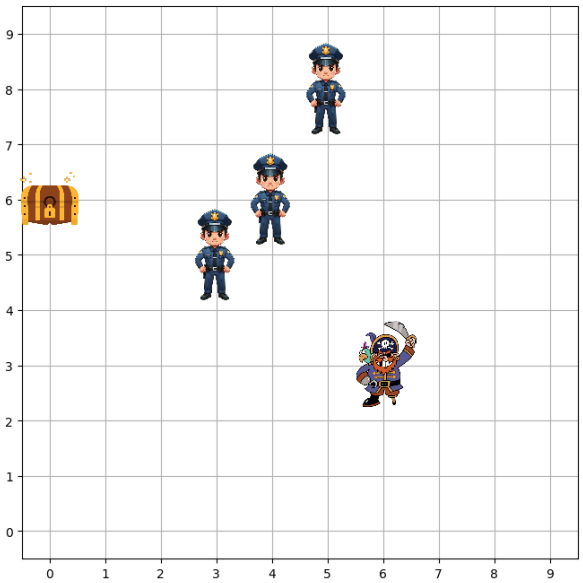


Figure 1: Visualisation du problème

Le début de cette étude sera un peu plus théorique, si vous souhaitez sauter cette partie, rendez-vous directement à l'implémentation.

2 Un problème de Reinforcement Learning

Avant de rentrer dans le vif du sujet, il est intéressant de s'intéresser au concent clé de ce projet : le **Reinforcement Learning**. Il s'agit d'un type d'apprentissage automatique où un agent apprend à interagir avec un environnement afin de maximiser une récompense cumulée. Cet apprentissage

repose sur un processus d'essai-erreur, où l'agent explore l'environnement, prend des actions, observe les résultats et ajuste ses décisions en conséquence.

2.1 Processus de décision markovien

Le RL repose souvent sur un cadre mathématique appelé **Markov Decision Process**, défini par un quintuplet :

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$$

- $\mathcal{S}$: Ensemble des états possibles.
- $\mathcal{A}$: Ensemble des actions possibles.
- $P(s'|s,a)$: Probabilité de transition vers l'état s' après avoir pris l'action a dans l'état s.
- $R(s,a)$: Récompense immédiate obtenue en prenant l'action a dans l'état s.
- $\gamma \in [0,1]$: Facteur d'actualisation (*discount factor*), pondère l'importance des récompenses futures.

L'objectif de l'agent est de trouver une **politique optimale** $\pi^* : S \to A$ qui maximise la récompense totale attendue.

2.2 Fonction de valeur et fonction d'action-valeur

Fonction de valeur d'un état $V^\pi(s)$: Cette fonction mesure la qualité d'un état s lorsqu'on suit une politique donnée π. Mathématiquement :

$$V^\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 = s, \pi\right].$$

Fonction d'action-valeur $Q^\pi(s,a)$: Elle évalue la qualité d'une paire (s,a) lorsqu'on prend l'action a dans l'état s, puis on suit la politique π :

$$Q^\pi(s,a) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 = s, a_0 = a, \pi\right].$$

2.3 Équation de Bellman

L'équation de Bellman exprime une relation récursive pour $V^\pi(s)$ et $Q^\pi(s,a)$:

Pour $V^\pi(s)$:

$$V^\pi(s) = \sum_{a \in A} \pi(a \mid s) \left[R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s, a) V^\pi(s')\right].$$

où $\pi(a \mid s)$ est la politique : une distribution qui indique quelle action a choisir dans l'état s.

Pour $Q^\pi(s,a)$:

$$Q^\pi(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s, a) \sum_{a' \in A} \pi(a' \mid s') Q^\pi(s', a').$$

2.4 Politique optimale et équation de Bellman optimale

La politique optimale π^* maximise la récompense cumulative pour tous les états s. Les fonctions de valeur optimales $V^*(s)$ et $Q^*(s,a)$ vérifient l'équation de Bellman optimale :

Pour $V^*(s)$:

$$V^*(s) = \max_{a \in A} \left[R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a) V^*(s') \right].$$

Pour $Q^*(s,a)$:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a) \max_{a' \in A} Q^*(s',a').$$

2.5 Les approches possibles

Méthodes basées sur la fonction de valeur (Q-learning) : Apprentissage direct de $Q^*(s,a)$ à l'aide d'une mise à jour incrémentale :

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

où α est un *learning rate*.

Méthodes basées sur les politiques (Policy Gradient): Optimisation directe de la politique $\pi(a \mid s)$ via un gradient ascendant :

$$\nabla_\theta J(\pi_\theta) = E\left[\nabla_\theta \log \pi_\theta(a \mid s) Q^\pi(s,a) \right].$$

Dans notre cas, nous utiliserons le Q-learning.

3 Deep Q-Learning

Une méthode de Q-Learning souvent utilisée pour résoudre des problèmes complexes est le **Deep Q-Learning**, ou DQN. Mais comment ça fonctionne ?

2.1. Approximation de $Q(s,a)$ avec un réseau neuronal

Dans DQN, un réseau neuronal paramétré par θ est utilisé pour approximer $Q(s,a)$:

$$Q(s,a;\theta)$$

où :

- Entrée : Une représentation de l'état s.
- Sortie : Les valeurs $Q(s,a)$ pour toutes les actions a.

3.1 Loss fonction

Le réseau est entraîné pour minimiser l'erreur quadratique moyenne entre $Q(s,a;\theta)$ et la cible y :

$$y = r + \gamma \max_{a'} Q(s',a';\theta^-)$$

$$L(\theta) = E_{(s,a,r,s')} \left[\left(y - Q(s,a;\theta) \right)^2 \right]$$

où θ^- sont les poids du réseau cible (*target network*).

3.2 Les techniques clés de DQN

- **Experience Replay** :
  - Stocke les transitions (s, a, r, s') dans un buffer.
  - Un mini-lot est échantillonné aléatoirement pour l'entraînement.
- **Target Network** :
  - Un réseau séparé $(Q(s, a; \theta^-))$ est utilisé pour calculer la cible.
  - Les poids θ^- sont mis à jour périodiquement en copiant les poids du réseau principal θ.
- **Epsilon-Greedy Exploration** :
  - Avec une probabilité ϵ, l'agent choisit une action aléatoire.
  - Sinon, il choisit l'action $\arg\max_a Q(s, a; \theta)$.

3.3 Algorithme DQN, pseudocode

Pour avoir une idée du déroulement de l'algorithme, nous pouvons nous baser sur ce pseudocode :

1. Initialiser les réseaux principal (θ) et cible $(\theta^- = \theta)$.
2. Pour chaque épisode :
   (a) Réinitialiser l'état initial s.
   (b) Répéter jusqu'à la fin de l'épisode :
      i. Choisir une action a selon epsilon-greedy.
      ii. Exécuter a, observer r et s'.
      iii. Stocker (s, a, r, s') dans le replay buffer.
      iv. Prélever un mini-lot de transitions (s, a, r, s') du buffer.
      v. Calculer la cible :
      $$y = r + \gamma \max_{a'} Q(s', a'; \theta^-)$$
      vi. Mettre à jour θ en minimisant :
      $$L(\theta) = (y - Q(s, a; \theta))^2$$
      vii. Mettre à jour θ^- périodiquement.

Si vous souhaitez le voir sous un autre angle avec plus de détails vous pouvez cliquer ici (lien vers Huggingface).

3.4 Tableau comparatif des algorithmes

Il existe bien entendu plein d'autres algorithmes de Q-Learning que nous n'aurons pas le temps de présenter un à un dans cette étude. Mais vous pouvez jeter un oeil au tableau comparatif de ces algorithmes.

Algorithme	Espaces d'action	Complexité	Applications
Q-Learning Standard	Discret	Faible	Petits environnements
Double Q-Learning	Discret	Modérée	Environnements stochastiques
DQN	Discret	Élevée	Jeux vidéo, tâches complexes
Double DQN	Discret	Élevée	Jeux vidéo, tâches stochastiques
HER	Discret/Continu	Élevée	Objectifs difficiles
Dueling DQN	Discret	Élevée	Environnements avec actions redondantes
Rainbow DQN	Discret	Très élevée	Scénarios nécessitant des performances maximales
DDPG	Continu	Élevée	Contrôle robotique, simulation physique

Le choix d'un algorithme spécifique dépend de la nature de l'environnement, de la taille de l'espace d'état et des actions, ainsi que de la complexité requise pour résoudre la tâche.

3.5 Pourquoi avoir choisi DQN dans notre cas ?

Le choix de **Deep Q-Network (DQN)** pour résoudre ce problème spécifique repose sur les caractéristiques de l'environnement et ses avantages.

Caractéristiques de notre étude

- **Environnement discret** :
  L'espace d'action est discret avec 5 actions possibles (mouvements du héros). DQN est particulièrement adapté aux environnements discrets, où il peut approximer la fonction $Q(s, a)$ pour un nombre fini d'actions.
- **Taille modérée de l'espace d'état** :
  L'état est représenté par une grille 10×10, soit 100 cellules. Ce n'est pas un espace immense, et un réseau neuronal utilisé dans DQN peut efficacement approximer les valeurs $Q(s, a)$.
- **Besoin de planification à court terme** :
  Le héros doit éviter les monstres tout en se rapprochant du trésor. DQN excelle dans les scénarios où les récompenses sont différées mais les décisions à court terme ont un fort impact.

Si le comportement reste insuffisant, on pourra toujours envisager dans une autre étude vers une évolution vers PPO ou SAC pour explorer les avantages d'autres paradigmes d'apprentissage.

4 Implémentation

Dans ce projet d'apprentissage par renforcement, nous avons conçu un environnement personnalisé pour entraîner un agent à atteindre un trésor tout en évitant des monstres.

4.1 Caractéristiques de l'environnement

Pour rappel, l'environnement, nommé `GameEnv`, est une grille 10×10 où le héros, contrôlé par l'agent, doit naviguer pour atteindre un trésor tout en évitant les monstres.

- **Grille discrète :** L'espace de grille fournit un environnement simple mais riche en stratégies, où chaque case représente un état distinct.
- **Agents et objets :**
  - **Héros :** L'agent apprend à naviguer dans la grille.
  - **Trésor :** L'objectif final de chaque épisode.
  - **Monstres :** Des obstacles dynamiques avec des mouvements aléatoires, introduisant un élément de danger.
- **Détails techniques :** Il y a naturellement eu besoin de :
  - Interdire les individus de sortir du tableau.
  - Interdire les monstres de spawn juste à coté de l'agent.
  - Interdire les monstres d'être à moins d'une case du trésor.

4.2 Définition des Récompenses

La fonction de récompense est cruciale pour guider l'apprentissage de l'agent. Elle est définie pour encourager les comportements désirables et pénaliser les comportements non souhaités. J'ai fait le choix d'implémenter des récompenses positives et négatives avec des valeurs totalement arbitraires. Ces choix peuvent être discutés et de meilleures alternatives existent sans aucun doute.

4.2.1 Récompenses positives

- +30 lorsque l'agent atteint le trésor, signalant la fin réussie de l'épisode.
- +3 lorsque l'agent réduit sa distance au trésor.

4.2.2 Récompenses négatives

- -1 lorsque l'agent augmente ou maintient sa distance au trésor.
- -5 lorsqu'un monstre est trop proche (≤ 2 cases).
- -10 si l'agent est adjacent à un monstre (fin de l'épisode).
- -0.1 pour chaque étape effectuée, afin d'encourager des actions rapides et efficaces.

4.2.3 Justification des choix

- Les récompenses positives encouragent l'agent à se rapprocher du trésor et à atteindre l'objectif.
- Les pénalités dissuadent l'agent de prendre des actions risquées (proximité des monstres) ou inutiles (rester inactif).
- Cette structure guide l'approximation de la fonction $Q(s, a)$ en valorisant les actions qui maximisent les récompenses cumulées.

4.3 Choix de l'algorithme et paramètres

Sans surprise pour ceux qui ont lu la partie théorique, le modèle utilisé est un **Deep Q-Network**. Avec un tel algorithme, il est question de gérer le dilemme entre l'exploration et exploitation. Mon choix est le souvent :

- L'agent commence avec une **forte exploration** ($\epsilon = 0.8$) pour découvrir l'environnement.
- **L'exploration diminue progressivement** jusqu'à $\epsilon = 0.2$, favorisant l'exploitation des actions apprises.

5 Suivi des performances

Pour assurer un suivi détaillé, j'ai mis en place un callback personnalisé, `RewardTrackerCallback`, pour suivre les métriques clés pendant l'entraînement :

- **Récompenses cumulées par épisode :** Indique si l'agent apprend à **maximiser ses gains**.
- **Longueur des épisodes :** Mesure l'**efficacité** croissante de l'agent.
- **Taux de succès :** Évalue la **fréquence** à laquelle l'agent atteint le trésor.

5.1 Résultats

Les graphiques suivant ont été générés durant la phase d'entrainement avec `total_timesteps=200000` ce qui représente une phase d'entrainement sur près de 17500 scénarios différents.

Figure 2: Evolution des récompenses par épisode

Description Ce graphique montre la récompense totale obtenue par épisode (ligne bleue) et la moyenne mobile des récompenses (ligne orange, calculée sur une fenêtre de 50 épisodes).

Analyse

- La moyenne mobile augmente régulièrement, ce qui indique que l'agent apprend à maximiser ses récompenses.
- Les fluctuations des récompenses individuelles montrent encore des épisodes avec des aléas, comme des erreurs ou des situations imprévues.
- Vers la fin de l'entraînement, la ligne orange se stabilise, suggérant que l'agent a appris une stratégie relativement efficace.

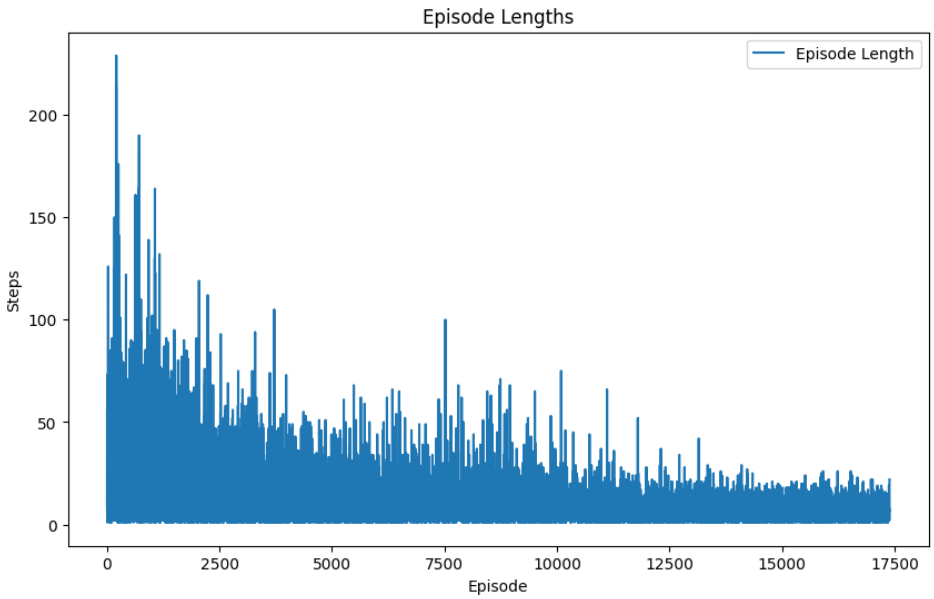


Figure 3: Evolution de la longueur des épisodes

Description Ce graphique illustre la durée de chaque épisode (en nombre de pas de l'agent) au fil de l'entraînement.

Analyse

- La longueur des épisodes diminue rapidement dans les premiers épisodes (qui étaient très exploratoires), montrant que l'agent apprend à atteindre ses objectifs plus efficacement.
- Après environ 5000 épisodes, les longueurs se stabilisent, indiquant que l'agent suit une stratégie optimisée.
- Une durée plus courte combinée à des récompenses croissantes indique que l'agent atteint ses objectifs (comme éviter les monstres ou trouver le trésor) en moins de pas.

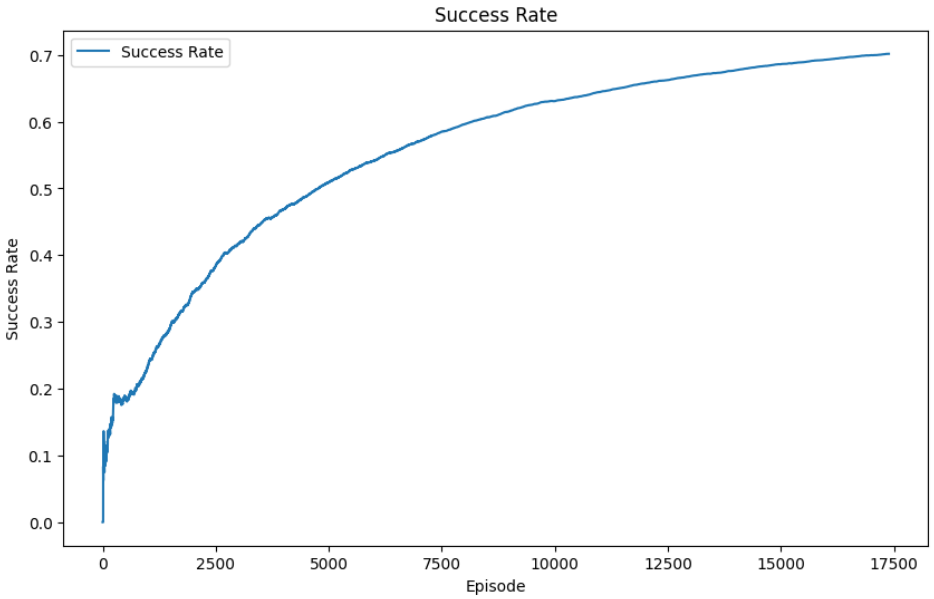


Figure 4: Evolution du taux de succès

Description Le taux de succès cumulatif (proportion d'épisodes réussis) est tracé au cours de l'entraînement.

Analyse

- Le taux de succès augmente régulièrement, atteignant environ 70 % à la fin de l'entraînement.
- L'augmentation continue montre que l'agent apprend progressivement à atteindre ses objectifs.
- Le ralentissement de l'augmentation vers la fin indique que l'agent approche de ses limites de performance dans l'environnement actuel.

5.2 Phase de test

Pour se donner une idée des performances de notre modèle hors entrainement, on peut mettre en place une phase de test sur un échantillon de 100 scénarios. Nous pourrons voir à l'issu de chacun de ces tests si l'agent arrive bien à trouver le trésor tout en évitant les monstres.

```
Episode 80: Reward = 53.20, Success = Yes
Episode 81: Reward = -1.30, Success = No
Episode 82: Reward = -4.20, Success = No
Episode 83: Reward = -10.00, Success = No
Episode 84: Reward = -4.20, Success = No
Episode 85: Reward = 38.70, Success = Yes
Episode 86: Reward = 44.50, Success = Yes
Episode 87: Reward = 56.10, Success = Yes
Episode 88: Reward = -10.00, Success = No
Episode 89: Reward = 30.00, Success = Yes
Episode 90: Reward = 41.60, Success = Yes
Episode 91: Reward = -7.10, Success = No
Episode 92: Reward = -10.00, Success = No
Episode 93: Reward = -4.20, Success = No
Episode 94: Reward = 35.80, Success = Yes
Episode 95: Reward = 38.70, Success = Yes
Episode 96: Reward = -7.10, Success = No
Episode 97: Reward = 41.60, Success = Yes
Episode 98: Reward = 38.70, Success = Yes
Episode 99: Reward = 32.90, Success = Yes
Episode 100: Reward = 32.90, Success = Yes

Test terminé : 100 épisodes
Récompense moyenne : 32.48
Taux de succès : 83.00%
```

Figure 5: Phase de test sur 100 échantillons

On obtient de bon résultats, à peu près 80% pour chaque échantillon (si on relance le code). Mais ces résultats peuvent sans doute être améliorés en effectuant quelques changements.

5.2.1 Optimisation de la fonction de récompense

- On pourrait donner une récompense encore plus importante à l'éloignement des monstres pour améliorer l'instinct de survie de l'agent.
- On pourrait aussi ajouter d'autres récompenses intermédiaires pour améliorer le guidage du héros.
- On pourrait réduire certaines des pénalités quitte à prendre un chemin plus long mais sans risques.

5.2.2 Ajustements des hyperparamètres

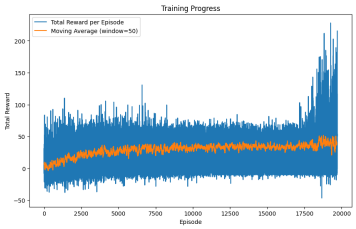
- On pourrait augmenter la fraction d'exploration (`exploration_fraction`) pour permettre davantage d'exploration initiale.
- Ou alors expérimenter avec des stratégies d'exploration alternatives comme l'épsilon-décroissant linéaire ou exponentiel (lien reddit ici).
- Ou encore utiliser une taille de mémoire (replay buffer) plus grande pour stocker plus d'expériences diversifiées.

5.2.3 Optimisation de l'entrainement

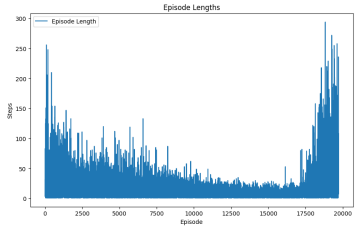
- On pourrait entrainer le modèle plus longtemps pour améliorer les performances.
- Ou bien utiliser un critère d'arrêt précoce basé sur une validation croisée ou une stagnation du taux de succès pour éviter de gaspiller du temps d'entraînement (on appelle ça l'early stopping).
- Ou enfin commencer avec un environnement simple (moins de monstres, des distances plus courtes) et augmenter progressivement la difficulté.

5.3 Jouons avec les paramètres

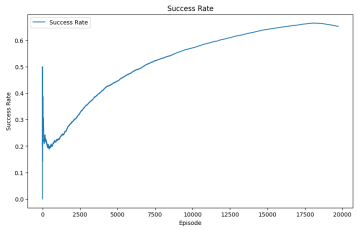
Comme énoncé précedemment, il est possible de jouer sur le nombre de scénarios qu'on va lui donner pendant l'entrainement à l'aide de `time_step`. Mais il faut également faire attention au surapprentissage qui peut se manifester de différentes manières lorsque l'on réalise un entrainement sur plus de **20000 scénarios** cette fois-ci :



(a) Evolution des récompenses par épisode



(b) Evolution du nombre de pas par épisode



(c) Taux de succès

Figure 6: Cas de surapprentissage

Par exemple, la longueur des épisodes augmente drastiquement à la fin de l'entraînement, ce qui pourrait indiquer que l'agent utilise des stratégies complexes pour prolonger les épisodes. Mais comme cela ne correspond pas à une vraie amélioration généralisée de ses performances, cela peut donc refléter un surapprentissage.

6 Conclusion

Ce projet a démontré l'efficacité de l'apprentissage par renforcement pour résoudre un problème de navigation dans un environnement dynamique. L'agent a montré des améliorations notables au fil de l'entraînement, avec une augmentation des récompenses, une réduction de la durée des épisodes, et un taux de succès atteignant environ 80%. Ces résultats reflètent la capacité de l'agent à apprendre des stratégies efficaces et constituent une base solide pour des recherches d'améliorations futures.

Sources

Shruti Dhume, *Deep Q-Network*, URL : https://medium.com/@shruti.dhumne/deep-q-network-dqn-90e1a8799871

Huggingface, *From Q to DQN*, URL : https://huggingface.co/learn/deep-rl-course/unit3/from-q-to-dqn

Jacob Murel Ph.D., *What is reinforcement learning?*, URL : https://www.ibm.com/think/topics/reinforcement-learning

Doc Baselines3, URL : https://stable-baselines3.readthedocs.io/en/master/modules/dqn.html

Fadi AlMahamid, *Reinforcement Learning Algorithms: An Overview and Classification*, URL : https://arxiv.org/pdf/2209.

Salar, *Deep Q-Learning: Understanding the Math Behind the Magic*, URL : https://ai.plainenglish.io/deep-q-learning-understanding-the-math-behind-the-magic-9ab03d893baf

Peter Foy, *Deep Reinforcement Learning: Guide to Deep Q-Learning*, URL : https://blog.mlq.ai/deep-reinforcement-learning-q-learning/

Arcaweb, *De l'équation de Bellmann au Q-Learning*, URL : https://www.arcaweb.ch/fr/apprentissage-par-renforcement-q-learning