UNIVERSITÉ
CÔTE D'AZUR

RAPPORT DE STAGE

Apprentissage par renforcement :

Étude et implémentation d'algorithmes

HUSSEINI MOHAMAD ALI

Mai 2025 à Juillet 2025

Tuteur de stage : FRANÇOIS DELARUE

Enseignant référent : THOMAS REY

Organisme d'accueil : LABORATOIRE J.A. DIEUDONNE

Responsable légal : THIERRY GOUDON

Table des matières

Fondements et Résolution avec Modèle Connu

1 Introduction à l'Apprentissage par Renforcement

Le contrôle optimal est une branche des mathématiques et de l'ingénierie qui vise à déterminer les meilleures actions ou stratégies à prendre au fil du temps pour piloter un système dynamique afin d'atteindre un objectif spécifique, souvent en maximisant ou minimisant une fonction de coût ou de récompense cumulée. L'idée centrale est de prendre une séquence de décisions qui optimisent la performance globale du système sur une période donnée.

Dans de nombreux problèmes, le système évolue de manière stochastique (aléatoire) et l'information disponible pour prendre des décisions peut être incomplète. C'est ici qu'intervient l'Apprentissage par Renforcement (AR), un domaine de l'intelligence artificielle étroitement lié au contrôle optimal stochastique. L'AR se concentre sur la manière dont un agent (le décideur) peut apprendre une stratégie optimale (une politique) par essais et erreurs en interagissant avec son environnement, sans nécessairement connaître a priori le modèle exact de ce dernier.

Les applications du contrôle optimal et de l'AR sont vastes et touchent de nombreux domaines :

- **Robotique :** Planification de trajectoires optimales pour les bras robotiques, navigation de robots mobiles autonomes.
- **Économie et Finance :** Gestion optimale de portefeuilles d'investissement, allocation de ressources, stratégies de tarification dynamique.
- **Aérospatiale :** Guidage de fusées et de satellites, optimisation de la consommation de carburant.
- **Processus Industriels :** Contrôle de réactions chimiques, gestion de chaînes de production, optimisation énergétique.
- **Systèmes Autonomes :** Conduite autonome de véhicules, gestion de réseaux (trafic routier, énergie, télécommunications).
- **Jeux :** Développement d'intelligences artificielles capables de maîtriser des jeux complexes comme les échecs, le Go ou les jeux vidéo Atari.

Dans le cadre de ce stage, nous nous intéressons à l'application de concepts fondamentaux de l'AR pour résoudre un problème de décision séquentielle dans un environnement simulé de type "GridWorld".

2 Équations de Bellman et Stratégie Optimale

La résolution de problèmes de décision séquentielle dans un environnement stochastique repose souvent sur le formalisme des Processus Décisionnels Markoviens (MDP). Un MDP est défini par :

- Un ensemble d'états possibles $\mathcal{S}$.
- Un ensemble d'actions possibles $\mathcal{A}$.
- Une fonction de transition $P(s'|s,a) = \mathbb{P}(S_{t+1} = s'|S_t = s, A_t = a)$.
- Une fonction de récompense $R(s,a) = \mathbb{E}[R_{t+1}|S_t = s, A_t = a]$.
- Un facteur d'actualisation $\gamma \in [0,1]$.

Une **politique** π définit la stratégie de l'agent. Elle peut être déterministe ($\pi(s) = a$) ou stochastique ($\pi(a|s) = \mathbb{P}(A_t = a|S_t = s)$).

L'**utilité** ou la **valeur** d'un état s sous une politique π, notée $V^\pi(s)$, est l'espérance du retour total actualisé :

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s \right] \tag{1}$$

Les valeurs $V^\pi(s)$ satisfont les **équations de Bellman (d'espérance)** pour π. Pour π déterministe :

$$V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, \pi(s)) V^\pi(s') \tag{2}$$

La **politique optimale** π^* maximise $V^\pi(s)$ pour tout s. $V^*(s) = \max_\pi V^\pi(s)$ satisfait l'**équation de Bellman d'optimalité** :

$$V^*(s) = \max_{a \in \mathcal{A}} \left[R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V^*(s') \right] \tag{3}$$

3 Présentation de l'Environnement : GridWorld

L'environnement est un GridWorld 4x3 inspiré de Russell & Norvig.

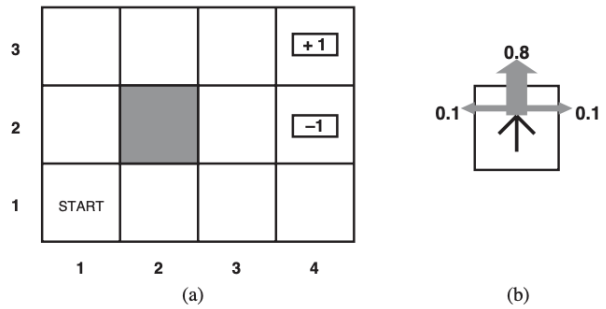


Figure 1 – *L'environnement GridWorld 4x3. Les coordonnées sont (ligne, colonne) avec (0,0) en haut à gauche. L'état (1,1) est un mur. Les états terminaux sont (0,3) avec récompense +1 et (1,3) avec récompense -1. Les autres états non terminaux ont une récompense de -0.04.*

Description :

- **États ($\mathcal{S}$) :** 11 cases accessibles. Le mur en (1,1) (selon indexation Python (ligne, colonne)) est inaccessible. Les états terminaux sont (0,3) et (1,3).
- **Actions ($\mathcal{A}$) :** HAUT (↑), BAS (↓), GAUCHE (←), DROITE (→).
- **Transitions ($P(s'|s, a)$) :** Stochastiques. L'action choisie a 80% de chances de réussir. Il y a 10% de chances d'aller à gauche perpendiculairement à l'action choisie, et 10% de chances d'aller à droite perpendiculairement. Si une action mène à une collision avec un mur extérieur ou l'obstacle intérieur, l'agent reste dans l'état s et gagne la récompense de cet état.
- **Récompenses ($R(s, a)$) :** Atteindre l'état (0,3) donne une récompense de +1. Atteindre l'état (1,3) donne une récompense de -1. Toutes les autres transitions (actions depuis des états non terminaux) coûtent -0.04. Une fois dans un état terminal, l'épisode se termine et aucune autre récompense n'est accumulée.
- **Facteur d'actualisation (γ) :** Dans nos expériences, nous utiliserons $\gamma = 1.0$, ce qui signifie qu'il n'y a pas d'actualisation des récompenses futures. Cela est approprié pour les environnements épisodiques où l'objectif est de maximiser la somme non actualisée des récompenses. Un épisode, aussi appelé "trial", correspond au chemin suivi par l'agent depuis son état de départ jusqu'à ce qu'il atteigne un état terminal. Une fois l'épisode terminé, l'agent est réinitialisé pour le suivant, soit à l'état initial, soit sur une case aléatoire non terminale.

4 Calcul de la Politique Optimale et de ses Utilités

Lorsque le modèle de l'environnement (P et R) est connu, on peut calculer la politique optimale π^* et les utilités optimales $V^*(s)$ correspondantes. Une méthode courante est l'**itération sur les valeurs** (Value Iteration), qui converge vers V^* **en appliquant itérativement l'équation de Bellman d'optimalité** (3). Une fois V^* obtenu, la politique optimale π^* est extraite par :

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^*(s') \right] \tag{4}$$

Une autre approche est l'**itération sur les politiques** (Policy Iteration), qui alterne entre évaluation d'une politique (calcul de V^π via l'équation (2)) et amélioration de la politique (application de l'équation (4) avec V^π). Et une autre méthode plus compacte avec des matrices :

$$V^\pi = (I - \gamma P^\pi)^{-1} R^\pi \tag{5}$$

Cette formulation matricielle est la solution analytique du système d'équations de Bellman (équation 2). Elle permet de calculer directement le vecteur d'utilités V^π pour une politique π fixe en une seule opération, à condition que la matrice $(I - \gamma P^\pi)$ soit inversible.

4.1 Résultats pour la Politique Optimale ($\gamma = 1.0$)

Nous avons utilisé l'itération sur les valeurs pour déterminer la politique optimale π^* et les valeurs d'utilité $V^*(s)$ pour chaque état, avec $\gamma = 1.0$ et un coût de transition de -0.04 pour les états non terminaux.

Les utilités $V^*(s)$ obtenues sont présentées dans le tableau suivant, où les coordonnées sont (ligne, colonne) avec (0,0) en haut à gauche :

Table 1 – *Utilités optimales $V^*(s)$ pour $\gamma = 1.0$ et $R(s,a) = -0.04$ (sauf transitions vers terminaux).*

(0,0) 0.812	**(0,1)** 0.868	**(0,2)** 0.918	**(0,3) [+1]** 1.000
(1,0) 0.762	**(1,1) [MUR]** —	**(1,2)** 0.660	**(1,3) [-1]** -1.000
(2,0) 0.705	**(2,1)** 0.655	**(2,2)** 0.611	**(2,3)** 0.388

La politique optimale π^* correspondante est visualisée dans la Figure 2.

→ (0,0)	→ (0,1)	→ (0,2)	+1 (0,3)
↑ (1,0)	**MUR** (1,1)	↑ (1,2)	-1 (1,3)
↑ (2,0)	← (2,1)	← (2,2)	← (2,3)

Figure 2 – *Politique optimale π^* pour $\gamma = 1.0$. Les flèches indiquent l'action optimale dans chaque état non terminal.*

Interprétation : Les valeurs d'utilité reflètent la somme maximale des récompenses espérées à partir de chaque état. Les états proches de la case terminale +1 ont des utilités élevées, tandis que ceux proches de la case -1 ou éloignés des récompenses positives ont des utilités plus faibles. La politique optimale montre le chemin le plus court et le moins risqué vers la récompense de +1, tout en évitant la récompense de -1 et les coûts de transition. Par exemple, depuis l'état (2,3), l'agent préfère se diriger vers la gauche (vers (2,2)) plutôt que de risquer de tomber dans l'état terminal -1 en allant vers le haut.

4.2 Résultats pour $\gamma = 0.8$

Afin d'observer l'impact du facteur d'actualisation, nous avons également mené l'expérience avec $\gamma = 0.8$. Avec cette valeur, les récompenses futures sont dépréciées, ce qui incite l'agent à chercher des récompenses plus immédiates.

Les utilités optimales $V^*(s)$ obtenues par itération de la valeur sont présentées dans la Table 2, et la politique optimale π^* correspondante est illustrée par les flèches.

Table 2 – *Utilités optimales $V^*(s)$ et Politique optimale π^* pour $\gamma = 0.8$.*

0.301 →	**0.472** →	**0.682** →	**1.000** +1
0.181 ↑	**MUR** —	**0.344** ↑	**-1.000** -1
0.091 ↑	**0.096** →	**0.188** ↑	**0.000** ←

Interprétation pour $\gamma = 0.8$: En comparant ces résultats avec le cas où $\gamma = 1.0$, on observe que toutes les utilités des états non terminaux sont plus faibles. Cela est attendu, car l'agent "dévalorise" les récompenses futures. La politique optimale, cependant, reste remarquablement similaire, indiquant que même avec une vision à plus court terme, le chemin vers la récompense de +1 demeure la stratégie dominante. La seule différence notable est en (2,3), où la politique est maintenant de se diriger vers la gauche, ce qui est une décision encore plus prudente pour éviter le -1.

4.3 Résultats pour $\gamma = 0.3$

Pour pousser l'analyse plus loin, nous avons testé un facteur d'actualisation très faible, $\gamma = 0.3$. Dans ce scénario, l'agent est fortement biaisé en faveur des récompenses immédiates et déprécie très rapidement la valeur des récompenses futures.

Les utilités optimales $V^*(s)$ et la politique π^* correspondante pour ce cas sont présentées dans la Table 3.

Table 3 – *Utilités optimales $V^*(s)$ et Politique optimale π^* pour $\gamma = 0.3$.*

-0.040 →	**0.010** →	**0.206** →	**1.000** +1
-0.053 ↑	**MUR** —	**-0.021** ↑	**-1.000** -1
-0.056 ↑	**-0.055** →	**-0.048** ↑	**-0.057** ↓

Interprétation pour $\gamma = 0.3$: Avec un 'gamma' aussi faible, l'agent devient extrêmement "myope". Les utilités des états éloignés de la récompense de +1 sont maintenant négatives, car le coût de transition de -0.04 accumulé sur quelques pas n'est plus compensé par la perspective lointaine et fortement dépréciée de la récompense finale.

Le changement de politique est particulièrement révélateur :

- La plupart des états pointent toujours vers le chemin le plus court vers +1.
- Cependant, depuis l'état (2,3), l'agent choisit désormais d'aller vers le **bas**. Cette action le fait rester sur place à cause du mur, ce qui est une action a priori inutile. L'explication est que, de son point de vue à court terme, la faible probabilité (10%) d'atterrir sur la case -1 en allant vers la gauche ou vers le haut représente un risque futur immédiat et trop important par rapport à la récompense lointaine de +1. Il préfère donc une action qui, bien que ne le faisant pas progresser, garantit de ne pas recevoir de pénalité immédiate.

Ce comportement illustre parfaitement comment un agent avec un horizon de planification court peut adopter des stratégies sous-optimales à long terme pour éviter des risques à court terme.

Apprentissage sans Modèle Connu

Lorsque le modèle de l'environnement (les probabilités de transitions et les récompenses des états) n'est pas connu, l'agent doit apprendre les utilités d'une politique par l'expérience. Un agent **passif** apprend la valeur V^π d'une politique fixe π qui lui est donnée. Il ne cherche pas à trouver la meilleure politique, mais seulement à évaluer celle qu'il suit. L'agent interagit avec l'environnement en suivant π, observe les transitions et les récompenses obtenues, et met à jour ses estimations d'utilité.

Dans cette section, nous explorons deux approches pour l'apprentissage passif :

— L'apprentissage par **Différence Temporelle (TD)**, une méthode sans modèle.

— L'apprentissage basé sur un modèle par **Programmation Dynamique Adaptative (ADP)**.

Nous évaluerons ces méthodes en leur faisant suivre la politique optimale π^* calculée précédemment (Section 2) et comparerons leurs performances. Pour toutes les expériences suivantes, $\gamma = 1.0$ et la récompense de transition est de -0.04 (sauf pour atteindre les états terminaux +1 ou -1). Les valeurs initiales des états $V_0(s)$ sont nulles.

5 Apprentissage Passif par Différence Temporelle (TD)

L'apprentissage par différence temporelle (TD) est une méthode sans modèle qui met à jour l'estimation de la valeur d'un état $V(s)$ en se basant sur la valeur estimée de l'état suivant $V(s')$. Pour un agent passif suivant une politique π, après avoir observé une transition de s à s' avec une récompense $r = R(s, \pi(s))$, la règle de mise à jour TD est :

$$V_{k+1}(s) \leftarrow V_k(s) + \alpha \left[(r + \gamma V_k(s')) - V_k(s)\right] \tag{6}$$

Le terme $(r + \gamma V_k(s')) - V_k(s)$ est l'**erreur de différence temporelle** δ_k. Il représente la différence entre l'estimation actuelle $V_k(s)$ et une meilleure estimation "cible" $r+\gamma V_k(s')$. Le taux d'apprentissage α décroît généralement avec le nombre de visites de l'état s (ou globalement avec le temps). Dans nos implémentations pour les graphiques, $\alpha = \frac{60}{59+N(s)}$, où $N(s)$ est le nombre de fois où l'état s a été visité et mis à jour. Un "trial" ou épisode est le chemin pris par l'agent en commençant par la case de départ jusqu'à une case terminal. Dans notre cas, chaque "trial" (ou épisode) commence à l'état (2,0) et se poursuit jusqu'à ce qu'un état terminal soit atteint (case +1 ou case -1).

5.1 Résultats de l'Agent Passif TD

L'agent TD a été exécuté en suivant la politique optimale π^* avec $\gamma = 1.0$. Les utilités des états ont été calculées après chaque trial. Les graphiques suivants montrent les résultats de 500 trials chacune pour les courbes d'apprentissage, et les résultats moyennés sur 20 exécutions de 100 trials pour l'erreur RMS.

FIGURE 3 – *Courbes d'apprentissage de l'agent TD passif. Estimations d'utilité pour une sélection d'états de 500 trials. L'agent suit π^*, $\gamma = 1.0$, $R(s, a) = -0.04$ pour les transitions non terminales.*

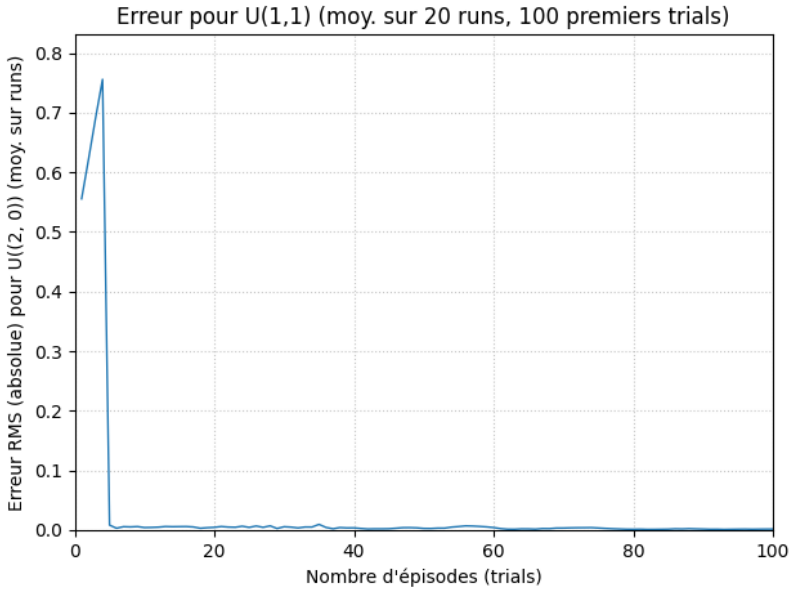


FIGURE 4 – *Erreur RMS dans l'estimation de l'utilité de l'état (1,1) (notre état (2,0), le point de départ des trials) par l'agent TD passif, moyennée sur 20 runs de 100 trials.*

Analyse des résultats TD : La Figure 3 montre que les estimations d'utilité pour les différents états convergent progressivement vers les valeurs optimales $V^*(s)$ (comparer avec Tableau 1). La convergence est plus rapide pour les états fréquemment visités ou ceux proches des états terminaux. Par exemple, la valeur de l'état terminal (4,3) (notre (0,3)) est fixée à $+1.0$ et ne change pas. Les autres états ajustent leurs valeurs en fonction des expériences. On observe des fluctuations, typiques de l'apprentissage par

échantillonnage.

L'erreur RMS pour l'état (1,1) (notre (2,0)), comme illustré dans la Figure 4, diminue avec le nombre de trials. L'erreur initiale est élevée car $V_0((2,0)) = 0$ alors que $V^*((2,0)) \approx 0.705$. L'erreur décroît rapidement au début puis plus lentement, indiquant que l'estimation s'affine. Après 100 trials, l'erreur RMS est d'environ 0.05.

6 Apprentissage Passif par Programmation Dynamique Adaptative (ADP)

L'agent ADP passif est une approche d'apprentissage par renforcement basée sur un modèle. Il apprend explicitement un modèle de l'environnement (P_{appris}, R_{appris}) à partir de son expérience, puis utilise ce modèle pour calculer les utilités des états, généralement en résolvant un système d'équations de Bellman.

Le processus pour un agent ADP suivant une politique fixe π est le suivant :

1. L'agent interagit avec l'environnement. Pour chaque transition $(s, a = \pi(s), s', r')$ où r' est la récompense pour arriver en s', il met à jour :
   — Les comptes de fréquence de transition $N(s, s')$ (nombre de fois où l'on est passé de s à s' en suivant $\pi(s)$).
   — Les comptes de départs de l'état s (sous l'action $\pi(s)$), $N(s)$.
   — Une estimation de la récompense $R^{\pi}_{appris}(s) = R(s, \pi(s))$. Dans notre implémentation, nous avons stocké la dernière récompense observée en quittant l'état s sous l'action $\pi(s)$.
2. À partir de ces comptes, il estime le modèle de transition appris $P^{\pi}_{appris}(s'|s) = N(s, s')/N(s)$.
3. Périodiquement (dans notre cas, après chaque "trial" ou épisode), l'agent utilise son modèle appris $(P^{\pi}_{appris}, R^{\pi}_{appris})$ pour recalculer les utilités $V^{\pi}(s)$ en résolvant le système d'équations de Bellman pour les états non terminaux, en tenant compte des valeurs fixes des états terminaux :

$$V^{\pi} = (I - \gamma P^{\pi}_{appris})^{-1} R^{\pi}_{appris} \tag{7}$$

Pour améliorer l'estimation du modèle et ne pas avoir une matrice singulière, chaque trial commence par un état non terminal choisi aléatoirement.

6.1 Résultats de l'Agent ADP Passif

L'agent ADP a été exécuté en suivant la politique optimale π^* avec $\gamma = 1.0$. Les graphiques suivants montrent les résultats de 100 trials chacune.

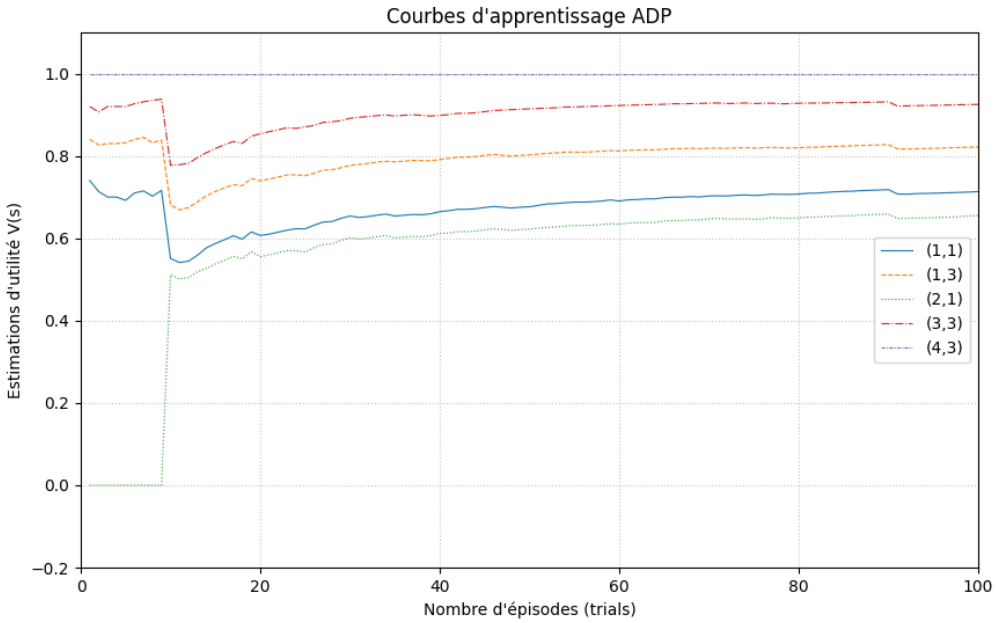


Figure 5 – *Courbes d'apprentissage de l'agent ADP passif. Estimations d'utilité pour une sélection d'états ,de 100 trials. L'agent suit* π^*, $\gamma = 1.0$, $R(s, a) = -0.04$ *pour les transitions non terminales.*

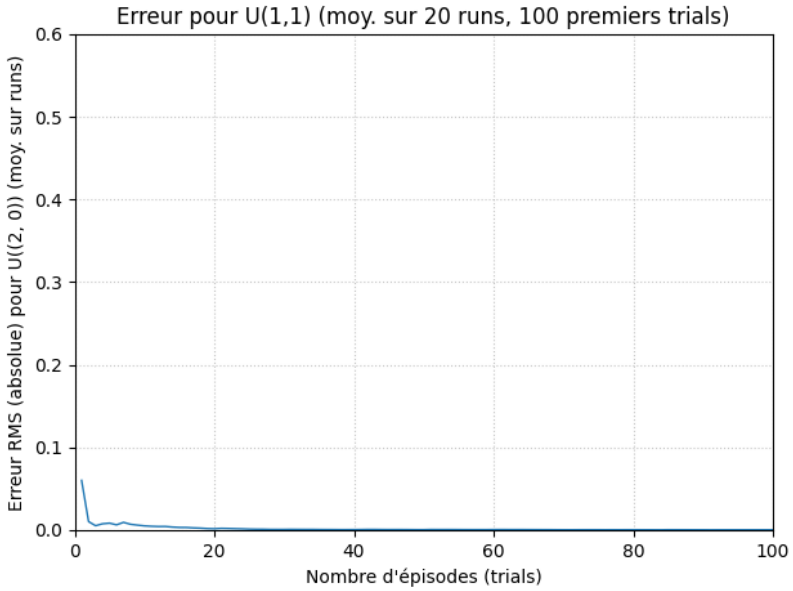


Figure 6 – *Erreur RMS dans l'estimation de l'utilité de l'état (1,1) (notre état (2,0)) par l'agent ADP passif, moyennée sur 20 runs de 100 trials.*

Analyse des résultats ADP : Les estimations d'utilité de l'agent ADP (Figure 5) convergent également vers les valeurs optimales $V^*(s)$. Comparé à l'agent TD, la convergence semble plus rapide et les courbes sont moins bruitées. Ceci est attendu car l'ADP utilise l'ensemble des informations collectées jusqu'à présent pour construire un modèle global et résoudre les équations de Bellman, ce qui a un effet de lissage. Des sauts dans les estimations peuvent se produire lorsqu'une nouvelle transition cruciale

(par exemple, vers un état terminal ou un état à haute valeur) est découverte pour la première fois, modifiant significativement le modèle appris et donc les utilités calculées.

L'erreur RMS pour l'état (1,1) (notre (2,0)), illustrée à la Figure 6, diminue très rapidement au cours des premiers trials. Après environ 20 trials, l'erreur est déjà très faible et se stabilise. Cette convergence rapide vers une faible erreur est une caractéristique des agents ADP, qui exploitent plus efficacement les données collectées en construisant un modèle. L'erreur RMS semble atteindre un plateau plus bas et plus stable que pour l'agent TD sur le même nombre de trials. Mais dès qu'on a un grand nombre d'état, cette méthode devient pas efficace.

7 Agent Actif Q-Learning TD

Contrairement à l'agent passif qui se contente d'évaluer une politique fixe, un **agent actif** doit apprendre à la fois les utilités et la politique optimale. Il est confronté au dilemme fondamental de l'apprentissage par renforcement : l'**exploration** (essayer de nouvelles actions pour découvrir leur valeur) contre l'**exploitation** (utiliser les meilleures actions connues pour maximiser la récompense).

Le Q-Learning est une méthode de différence temporelle (TD) sans modèle, conçue pour résoudre ce problème. L'idée centrale est de ne pas apprendre la valeur d'un état $V(s)$, mais la valeur d'une paire (état, action), notée $Q(s, a)$.

La **Q-valeur**, $Q(s, a)$, représente la récompense immédiate que l'on obtient en prenant l'action a dans l'état s, plus la somme de toutes les récompenses futures actualisées, en supposant que l'agent agira de manière optimale **à chaque étape future**.

Formellement, les Q-valeurs optimales $Q^*(s, a)$ satisfont l'**équation de Bellman d'optimalité** :

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) \max_{a'} Q^*(s', a') \tag{8}$$

Le terme clé ici est $\max_{a'} Q^*(s', a')$. Il représente la valeur maximale que l'on peut espérer obtenir à partir de l'état suivant s'. Ceci représente la politique optimale : à partir de s', l'agent choisira l'action a' qui maximise sa Q-valeur, et la valeur de cette décision est $\max_{a'} Q^*(s', a')$. Notons que cette valeur est exactement $V^*(s')$.

Exemple concret : Prenons l'état $s = (1, 2)$ (à droite du mur). Que signifie $Q^*((1, 2), \text{HAUT})$?

- L'agent prend l'action HAUT. Il reçoit immédiatement une récompense $R((1, 2), \text{HAUT}) = -0.04$.
- Il arrive ensuite dans un nouvel état s'. En raison de la stochasticité de l'environnement, s' pourrait être :
  - $(0, 2)$ avec une probabilité de 0.8.
  - $(1, 1)$ (mur, donc reste en $(1, 2)$) avec une probabilité de 0.1.
  - $(1, 3)$ (le piège -1) avec une probabilité de 0.1.
- La valeur future espérée est la somme pondérée par ces probabilités de la meilleure valeur possible à partir de chaque état d'arrivée : $0.8 \times V^*((0, 2)) + 0.1 \times V^*((1, 2)) + 0.1 \times V^*((1, 3))$.
- Et chaque $V^*(s')$ est simplement $\max_{a'} Q^*(s', a')$. Par exemple, $V^*((0, 2)) = \max_{a'} Q^*((0, 2), a')$.

La Q-valeur $Q^*((1, 2), \text{HAUT})$ est donc la somme de la récompense immédiate et de cette valeur future espérée.

L'avantage de cette approche est que si l'agent connaît les Q-valeurs optimales, la politique optimale est triviale à extraire sans connaître le modèle : $\pi^*(s) = \arg\max_a Q^*(s, a)$. De même, l'utilité optimale d'un état est $V^*(s) = \max_a Q^*(s, a)$.

L'agent apprend ces Q-valeurs par l'expérience. Après avoir exécuté une action a depuis un état s, observé une récompense r et atterri dans un nouvel état s', la règle de mise à jour du Q-Learning est :

$$Q(s,a) \leftarrow Q(s,a) + \alpha[N(s,a)] \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \tag{9}$$

Ici, α est le taux d'apprentissage qui dépend du nombre de fois que l'action a est choisi dans l'état s. Le terme $\max_{a'} Q(s',a')$ est l'estimation de l'utilité optimale de l'état suivant, directement extraite de la table Q actuelle. Cette mise à jour se fait sans connaître P ou R.

Pour gérer le dilemme exploration/exploitation, nous utilisons une stratégie ε**-greedy**. Avec une probabilité ε, l'agent choisit une action aléatoire (explore) ; avec une probabilité $1-\varepsilon$, il choisit l'action qui maximise la Q-valeur actuelle (exploite).

7.1 Résultats de l'Agent Actif Q-Learning

Le succès de l'agent Q-Learning dépend de manière critique du réglage de ses hyperparamètres. Après une phase d'expérimentation, une configuration robuste a été trouvée, permettant à l'agent de converger vers la politique optimale. Les résultats présentés ici ont été obtenus après 50 000 épisodes.

7.1.1 Influence des Hyperparamètres et Choix de la Configuration

Le défi majeur de notre GridWorld pour un agent actif est le coût de transition de -0.04. Cette pénalité constante peut dissuader l'exploration : l'agent peut apprendre qu'il est "moins mauvais" de rester dans une petite zone près du point de départ que de risquer de longs trajets coûteux pour trouver une récompense incertaine. Le choix des hyperparamètres doit surmonter ce biais.

- **Taux d'apprentissage (α) :** Nous avons conservé la formule dynamique $\alpha = \frac{60}{59+N(s,a)}$. Cette approche permet à l'agent d'apprendre rapidement au début (quand $N(s,a)$ est petit, $\alpha \approx 1$) et de stabiliser ses connaissances plus tard (quand $N(s,a)$ est grand, $\alpha \to 0$). Un α constant, même petit, empêcherait une convergence fine des Q-valeurs.
- **Facteur d'actualisation (γ) :** Fixé à $\gamma = 1.0$, car l'environnement est épisodique (après avoir atteint une case terminal, l'agent est réinitialisé dans une case aléatoire) et nous voulons maximiser la somme totale des récompenses sans la déprécier au fil du temps.
- **Taux d'exploration (ε) :** C'est l'hyperparamètre le plus sensible. Un ε trop faible ne permettra jamais à l'agent de s'aventurer assez loin pour découvrir la récompense de +1. Un ε trop élevé ralentit la convergence car l'agent agit trop souvent de manière aléatoire. Nous avons testé plusieurs valeurs pour ε (voir Tableau 4).
- **Nombre d'épisodes :** Il doit être suffisamment grand pour que l'exploration ait le temps de porter ses fruits et que les valeurs se propagent à travers toute la grille, depuis les états terminaux jusqu'aux états les plus éloignés.

Le tableau suivant résume les résultats de nos expériences sur le taux d'exploration, en utilisant 50000 épisodes pour chaque test. Nous considérons un succès si la politique apprise est identique à la politique optimale théorique.

Table 4 – *Impact du taux d'exploration ε sur la convergence de Q-Learning.*

Valeur de ε	Résultat	Analyse
0.05	Échec	Politique sous-optimale (l'agent reste bloqué).
0.10	Échec	Politique sous-optimale (exploration insuffisante).
0.20	Succès (parfois)	Convergence instable.
0.30	**Succès**	**Convergence robuste, trouve la politique optimale.**
0.50	Succès	Convergence plus lente mais fiable.

Nous avons donc retenu $\varepsilon = 0.3$ comme un bon compromis, garantissant une exploration suffisante pour surmonter les minima locaux tout en permettant à l'agent d'exploiter efficacement ses connaissances pour affiner sa politique.

7.1.2 Politique et Utilités Apprises

Avec la configuration retenue ($\varepsilon = 0.3$, α dynamique, 50 000 épisodes), les utilités des états, extraites des Q-valeurs apprises ($V(s) = \max_a Q(s,a)$), sont présentées dans le Tableau 5.

Table 5 – *Utilités $V(s)$ apprises par l'agent Q-Learning.*

(0,0) 0.853	**(0,1)** 0.905	**(0,2)** 0.952	**(0,3) [+1]** 1.000
(1,0) 0.809	**(1,1) [MUR]** —	**(1,2)** 0.691	**(1,3) [-1]** -1.000
(2,0) 0.755	**(2,1)** 0.703	**(2,2)** 0.656	**(2,3)** 0.494

La politique correspondante, extraite des Q-valeurs, est visualisée dans la Figure 7.

→ (0,0)	→ (0,1)	→ (0,2)	+1 (0,3)
↑ (1,0)	**MUR** (1,1)	↑ (1,2)	-1 (1,3)
↑ (2,0)	← (2,1)	← (2,2)	← (2,3)

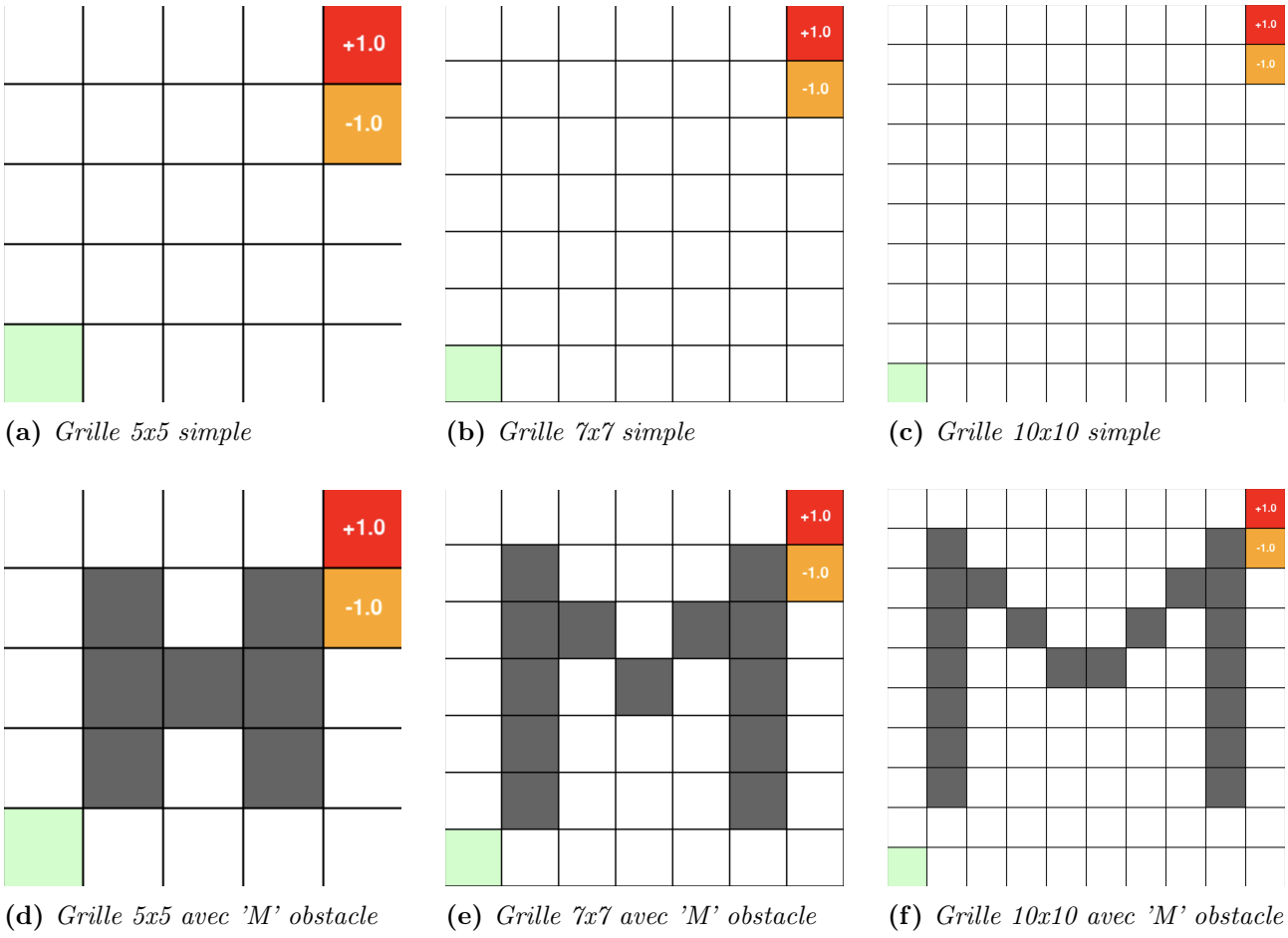
Figure 7 – *Politique apprise par l'agent Q-Learning.*

Analyse des résultats du Q-Learning : Les résultats sont un succès. L'agent actif a réussi à apprendre une politique optimale sans aucune connaissance a priori du modèle. Les valeurs d'utilité du Tableau 5 sont proches des valeurs théoriques calculées par itération de la valeur (Tableau 1). La politique illustrée à la Figure 7 est identique à la politique optimale théorique. Ce succès est dû à l'exploration agressive et constante ($\varepsilon = 0.3$) qui a forcé l'agent à ne jamais cesser de chercher, ce qui lui a permis de surmonter la pénalité de -0.04 par pas et de découvrir le long chemin vers la récompense.

Analyse des Performances

8 Visualisation des Environnements de Test

Pour évaluer la scalabilité des algorithmes, des environnements de taille croissante ont été générés. La Figure 8 illustre la structure de trois de ces grilles, avec une complexité croissante due à l'ajout d'obstacles.



(a) *Grille 5x5 simple* **(b)** *Grille 7x7 simple* **(c)** *Grille 10x10 simple*

(d) *Grille 5x5 avec 'M' obstacle* **(e)** *Grille 7x7 avec 'M' obstacle* **(f)** *Grille 10x10 avec 'M' obstacle*

Figure 8 – *Visualisation des environnements de test de complexité croissante utilisés pour l'analyse des performances.*

8.1 Temps de Calcul

L'efficacité computationnelle est un critère essentiel pour évaluer un algorithme. Nous avons mesuré le temps d'exécution de chaque méthode sur des environnements de complexité croissante. Les trois graphiques suivants présentent ces performances sous différentes configurations d'environnement.

8.1.1 Environnement sans obstacles

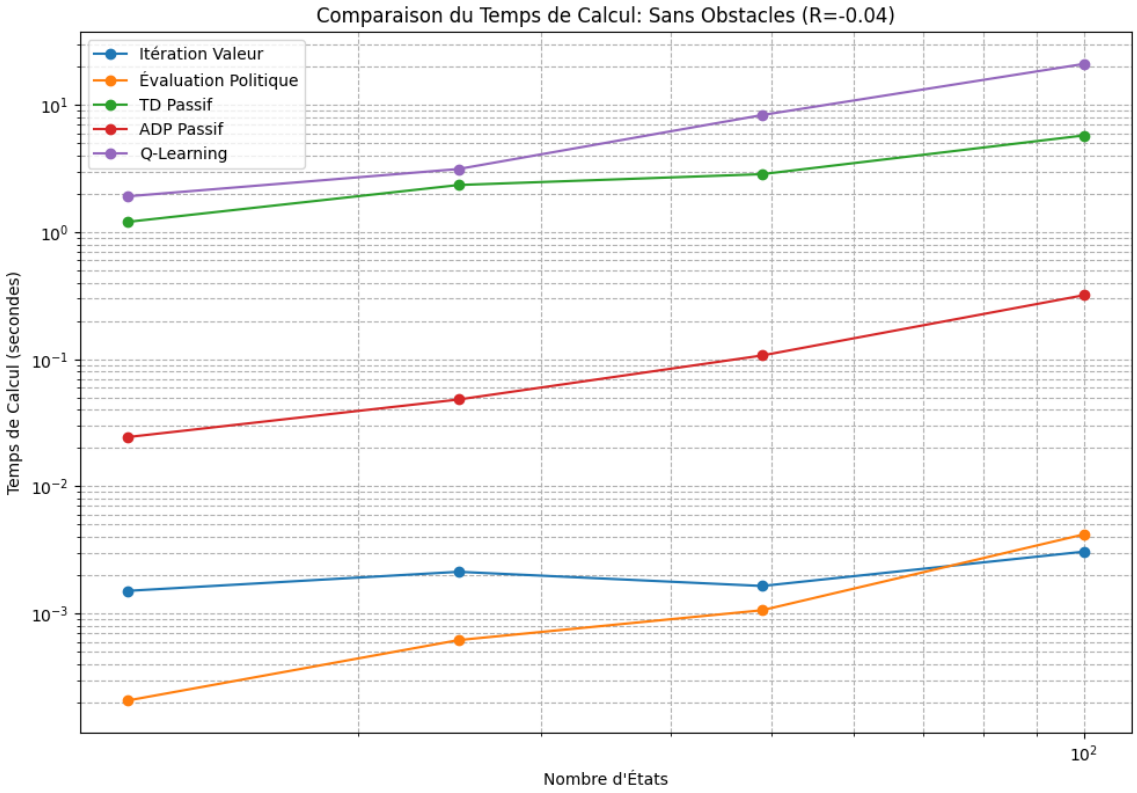


Figure 9 – *Performance sur des grilles simples, sans obstacles complexes.*

Analyse (Grilles Simples) : Ce premier graphique, sur une échelle log-log, illustre les performances dans des conditions idéalisées, sans obstacle. Les droites quasi parfaites confirment que la complexité des algorithmes est polynomiale par rapport au nombre d'états. La hiérarchie des performances est très claire :

- **Évaluation Politique** est la méthode la plus rapide, grâce à la connaissance totale de l'environnement et à la résolution directe d'un système linéaire. On remarque que pour un grand nombre d'états, l'itération de la valeur devient plus efficace en temps de calcul que l'évaluation de la politique, qui nécessite d'inverser une matrice de plus en plus grande.
- Les autres méthodes qui apprennent par l'expérience (**ADP, TD, Q-Learning**) sont intrinsèquement plus lentes. On peut aussi voir que la pente du temps de calcul de l'agent ADP est plus grande que celle de l'agent TD Passif à partir de 50 états, ce qui suggère que l'agent TD pourrait être plus efficace pour des environnements avec un très grand nombre d'états.
- Le **Q-Learning** est le plus coûteux, car il doit non seulement simuler un grand nombre d'épisodes, mais aussi gérer l'exploration active pour découvrir la politique optimale.

8.1.2 Environnement avec Obstacles

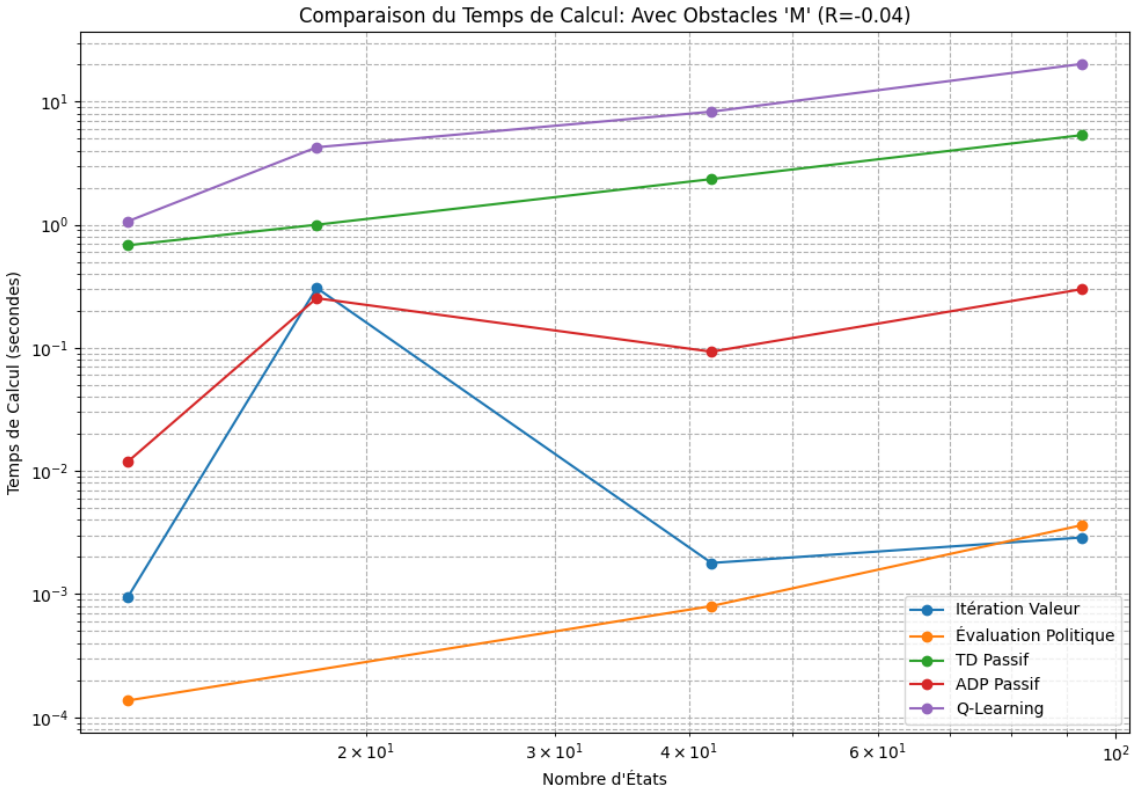


Figure 10 – *Performance sur des grilles contenant les obstacles en forme de 'M'.*

Analyse (Grilles avec Obstacles 'M') : L'introduction d'obstacles complexes ne change pas la hiérarchie fondamentale des performances, mais elle accentue les écarts et révèle un phénomène important :

- La performance ne dépend plus seulement du nombre d'états, mais aussi de la forme de l'environnement. On observe un pic de performance non monotone pour l'**Itération de la Valeur**, qui nécessite plus d'itérations pour converger dans ce cas précis.
- L'écart entre les méthodes basées sur un modèle et les méthodes d'apprentissage se creuse. Résoudre un problème plus complexe par simulation (TD, Q-Learning) est, comme attendu, plus coûteux en temps que de le résoudre avec une connaissance parfaite du modèle (Itération de la Valeur).
- La pente de la courbe pour le Q-Learning est particulièrement raide, soulignant la difficulté de l'exploration dans des environnements plus contraints.

8.1.3 Environnement avec obstacle et récompenses non terminaux nulles

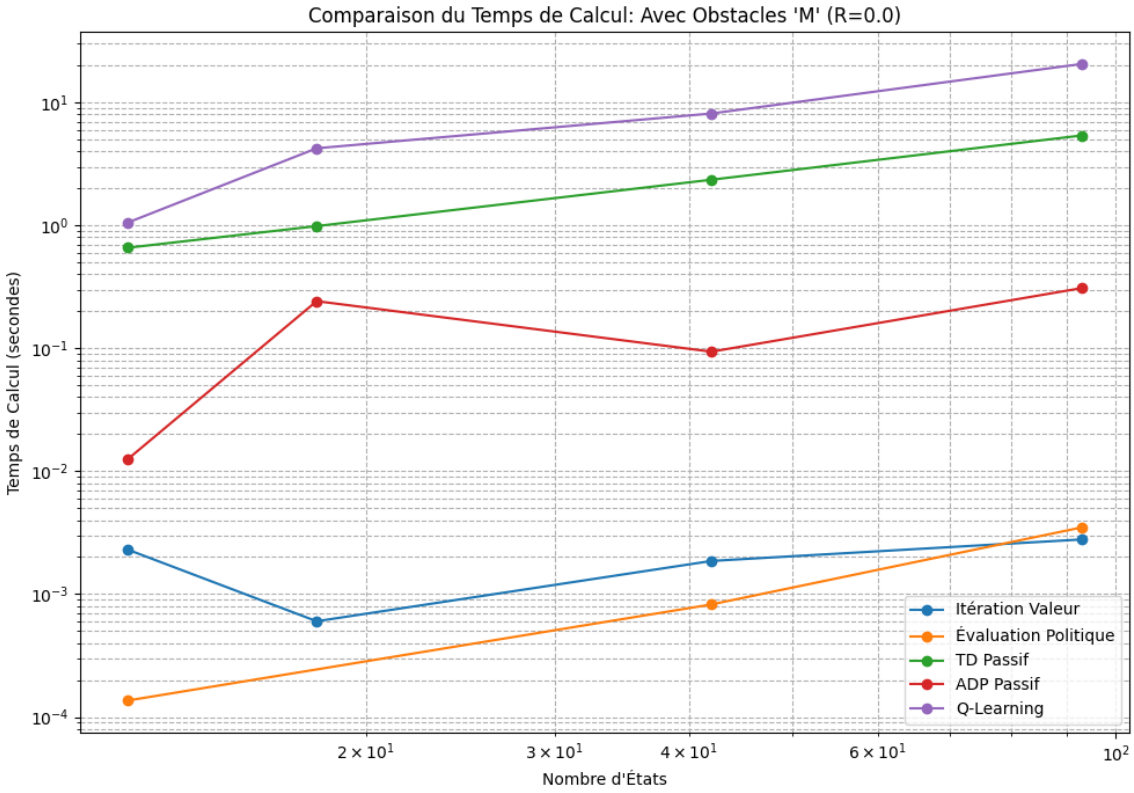


Figure 11 – *Performance sur des grilles avec obstacles et récompenses de transition nulles (R=0.0).*

Analyse (Obstacles et Récompenses Éparses) : Ce graphique illustre le cas des récompenses éparses, un défi majeur en apprentissage par renforcement.

- À première vue, ce graphique est presque identique au précédent. La hiérarchie et les temps de calcul semblent inchangés.
- Cette similarité révèle une limite de notre méthode de mesure à budget fixe. Théoriquement, apprendre sans les "indices" des récompenses négatives est beaucoup plus difficile. Cependant, comme nous mesurons le temps pour exécuter un nombre fixe d'étapes, le graphique ne capture pas cette augmentation de la difficulté d'apprentissage.
- Il ne montre que le coût de calcul brut, qui est le même que la récompense soit de -0.04 ou de 0.0. Une mesure basée sur le temps pour atteindre une *qualité de solution* équivalente montrerait des temps beaucoup plus longs pour les agents d'apprentissage dans ce scénario. Il faut une force computationnel plus importante pour pouvoir voir les performances réelles.

Conclusion

Ce projet a permis de mettre en œuvre et de comparer une gamme d'algorithmes fondamentaux de l'apprentissage par renforcement, allant de la planification avec un modèle parfait à l'apprentissage actif sans aucune connaissance a priori.

La première conclusion majeure est la **valeur inestimable du modèle**. Lorsque les dynamiques de l'environnement (P) et les récompenses (R) sont connues, les algorithmes de programmation dynamique comme l'Itération de la Valeur sont extraordinairement efficaces, trouvant la politique optimale en une fraction du temps requis par les méthodes d'apprentissage.

En l'absence de modèle, l'agent doit apprendre de ses interactions, ce qui introduit un compromis fondamental entre l'efficacité des données et la complexité de calcul.

- L'**agent ADP**, en apprenant un modèle, a montré une convergence plus rapide en termes de nombre d'épisodes. Il exploite plus efficacement chaque information collectée. Cependant, ce gain en "sample efficiency" (il a besoin de peu d'interactions avec l'environnement ie peu de pas, peu d'épisodes, pour apprendre une bonne politique.) a un coût : la résolution périodique d'un système d'équations, qui le rend coûteux en calcul.
- L'**agent TD**, purement "sans modèle", est plus simple et plus rapide à chaque mise à jour. Il est computationnellement moins exigeant que l'ADP mais nécessite souvent plus d'interactions avec l'environnement pour atteindre le même niveau de précision.
- Enfin, le **Q-Learning** a démontré sa capacité à résoudre le problème complet de l'apprentissage par renforcement : trouver une politique optimale à partir de zéro. Ce succès, qui est au cœur des applications modernes de l'AR, se paie par un coût de simulation élevé, nécessité par une exploration suffisante pour surmonter les minima locaux et découvrir les récompenses à long terme.

Ce travail illustre concrètement le spectre des solutions en contrôle optimal stochastique : d'un côté, la planification rapide si tout est connu ; de l'autre, l'apprentissage patient par essais et erreurs si l'environnement est une "boîte noire". Le choix de l'algorithme dépend donc entièrement de la connaissance que l'on a du problème à résoudre.

Remerciements

Au terme de ce projet de stage, qui marque une étape importante de mon parcours académique et personnel, je souhaite exprimer ma plus profonde gratitude aux personnes dont le soutien, l'encadrement et les conseils ont été essentiels à la réalisation de ce mémoire.

Je tiens tout d'abord à remercier chaleureusement et sincèrement mon maître de stage et professeur, Monsieur François Delarue. Sa vision scientifique, sa rigueur intellectuelle et sa grande expertise dans le domaine de l'apprentissage par renforcement ont été une source d'inspiration constante. Je lui suis extrêmement reconnaissant pour sa disponibilité sans faille, pour la patience dont il a fait preuve face à mes questions, et pour la pertinence de ses conseils qui ont guidé ma démarche à chaque étape. Au-delà de l'encadrement technique, je le remercie pour la confiance qu'il m'a témoignée et pour m'avoir encouragé à explorer des pistes de réflexion ambitieuses. Cette expérience à ses côtés a été exceptionnellement formatrice.

Mes remerciements s'adressent également avec une grande sincérité à Monsieur Thomas Rey, responsable du master. Son engagement envers la réussite des étudiants et sa gestion bienveillante de la formation ont créé un environnement d'apprentissage stimulant et de grande qualité. Je le remercie pour son écoute, pour son soutien tout au long de l'année et, bien entendu, pour m'avoir offert

l'opportunité de réaliser ce stage dans un cadre aussi propice au développement de mes compétences.

Enfin, je souhaite étendre ma reconnaissance à l'ensemble du corps enseignant du master pour la richesse des connaissances transmises.

9 Référence et Annexe

9.1 Référence

S. J. Russell et P. Norvig, *Artificial Intelligence : A Modern Approach*, 3e éd., Pearson, 2010.

Al Sweigart, *Invent Your Own Computer Games with Python*, 4e éd., No Starch Press, 2016

9.2 Annexe

Le code et le jeu sont disponibles sur github :
https://github.com/Mohamadali0602/Reinforcement-Learning