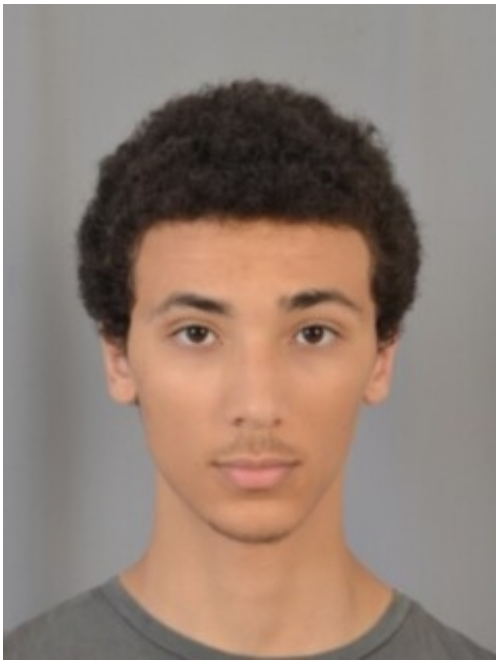
NET4203 - Allocation dynamique des ressources dans le cloud



Taha Habib

BENJEMAA Yassine

Introduction :

Les algorithmes d'itération de la valeur Q (Q-value iteration) sont un type d'algorithmes d'apprentissage par renforcement qui visent à déterminer la stratégie optimale pour prendre des décisions en minimisant le coût ou en maximisant la récompense associée à chaque action possible. Deux algorithmes de Q-value iteration couramment utilisés sont SARSA et Q-Learning.
Notre but est de coder les deux algorithmes Q-learning et Sarsa pour gérer l'allocation dynamique de ressources dans le cloud en prenant en compte la charge du système et le nombre de VMs en fonctionnement. Nous considérons que nous contrôlons le système en décidant d'allouer ou désallouer des VMs. Une comparaison des résultats obtenus par les deux algorithmes sera établi afin de vérifier l'efficacité de chacun.

Description du système :

Dans un premier temps, décrivons le système. L'allocation de machines virtuelles peut être modéliser par un MDP car le modèle est stochastique, où un agent prend des décisions et où les résultats de ses actions sont aléatoires. La MDP pour ce système est définie par le triplet (S, A, R), où S est l'ensemble des états, A est l'ensemble des actions disponibles et R est la récompense associée à chaque transition d'état. Les états s = (m, n) peuvent être représentés par un tuple, où m représente le nombre de clients dans le système et n représente le nombre de serveurs actifs. Dans notre cas, nous associons à chaque état un indice i, en mettant tous les états possibles dans une liste et nous ne manipulons ainsi que ces indices tout au long du sujet.
La matrice de transition sera alors une matrice 2D avec les différentes indexation des états sur les lignes et les actions en colones.
Une action, définie par l'expression $\alpha_j = j$, est l'action de maintenir j serveurs en activité, les actions peuvent ainsi être représentées par l'entier j indiquant le nombre de serveurs à activer.

On souhaite par la suite calculer à chaque itération le coût de chaque couple (état ,action) :

$$r(s, \alpha_j) = j \cdot Cf + (j - n) \cdot 1_{j>n} \cdot C_a + (n - j) \cdot 1_{j<n} \cdot C_d + m \cdot C_H \tag{1}$$

Ce côut sera par la suite ajouté négativement dans le reward total étant donné que notre *reward= -coût*. Ainsi en voulant minimiser le coût, on se retrouve à maximiser le reward, cela explique les valeurs négatives du reward que nous trouverons par la suite.

Avec :
C_f : coût d'une VM en fonctionnement par slot,
C_a : coût d'activation d'une VM,
C_d : coût de désactivation d'une VM,
C_H : coût de maintien d'un client dans la file,
Pour simplifier, nous prenons :

$$Cf = C_a = C_d = C_H = C = 1 \tag{2}$$

Le problème consiste à choisir le nombre de VMs (machines virtuelles) à allouer, à travers les actions α_j, pour traiter les paquets d'arrivée tout en minimisant les coûts totaux et en respectant la capacité limitée du système B. Cela en ayant une distribution de probabilité pour les arrivées des paquets, notée P_A, tel que $P_A(i)$ est la probabilité qu'il y ait i arrivées de paquets. Et avec K le nombre maximal de VMs.
Notons bien que nous mettrons les probabilités des différentes arrivés, suivant la distribution P_A, sous une liste, avec le ième élément de la liste $P_A(i)$. Cette liste sera fixé au début de chaque algorithme avec les autres paramètres du sujet..

Q-Learning :

Premièrement, nous essayons de résoudre ce problème en utilisant le Q-learning.
Le Q-learning est un algorithme d'apprentissage par renforcement qui se base sur le principe de maximisation, à chaque itération, de la valeur de la fonction Q, qui représente la valeur attendue de prendre une action dans un état donné. Le Q-learning, se base sur la formule suivante pour la mise à jour de la table Q :

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

Dans ce contexte, la nouvelle valeur de Q(s,a) prend en compte la récompense actuelle, r, ainsi que la valeur attendue à long terme $\gamma * \max_{a'} Q(s',a')$, càd en prenant l'action qui maximise la valeur de Q pour l'état suivant s'. Ce qui donne en algorithme :

Algorithm 1 Algorithme Q-learning

On initialise la table Q avec des valeurs arbitraires
for each *episode* $\in \{episodes\}$ **do**
    Initialiser l'état s
    for each *pas* $\in \{episode\}$ **do**
        while Not atteinte d'un état terminal **do**
            Choisir l'action a avec la plus grande valeur Q dans l'état s ou avec une probabilité ϵ sélectionner une action aléatoire
            Prendre l'action a, observer la récompense r et le nouvel état s'
            Mettre à jour la valeur Q en utilisant la formule :

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

            Définir $s \leftarrow s'$
        end while
    end for
end for

Après avoir codé cet algorithme sous python, nous avons tracé la courbe du reward total de chaque épisode en fonction du nombre d'épisodes. Nous prenons comme paramètre :
- Nombre de VMs : K = 10
- Capacité du système : B = 50
- Taille maximale du batch : b = 5
- P_A : une liste de probabilités uniformes de taille b
- Nombre de paquets servis par slot d = 2

Fixons la valeur de γ à 0.9 et jouons sur le paramètre α : taux d'apprentissage. Nous remarquons que dans les 3 premières figures, les courbes convergent en fin des épisodes, cependant la courbe représentée sur la figure 3 est plus stable par rapport à celle de la figure 2, qui est elle même plus stable que la courbe 1, cela est dû au fait que plus le taux d'apprentissage est élevé plus on prend en compte la nouveauté, et moins on prend en compte l'existant. Pour une valeur α proche de 0, la valur de $(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$ serait négligeable par rapport à $Q(s,a)$, ce qui implique que les valeurs de Q ne seront pas changé énormément à chaque mise à jour, d'où la stabilité dans la convergence.

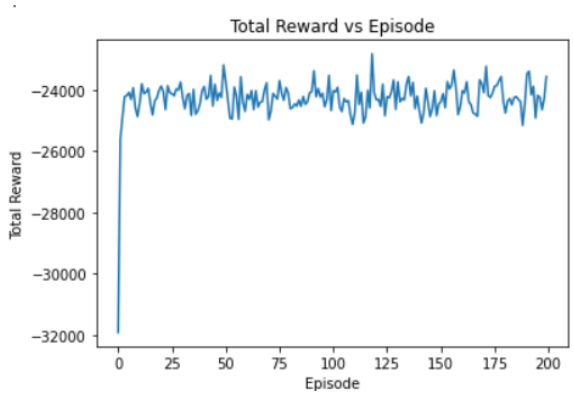


FIGURE 1 – $\alpha = 0.8$, $\gamma = 0.9$

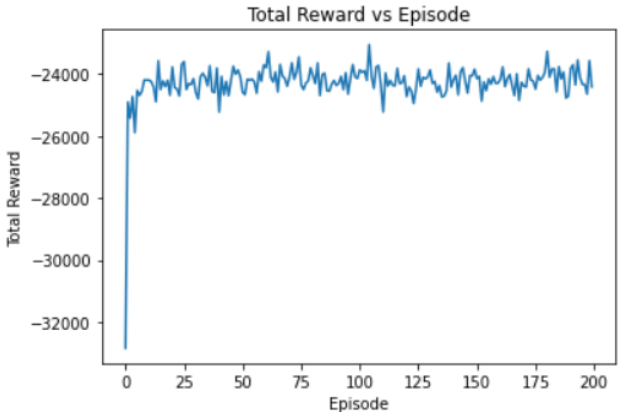


FIGURE 2 – $\alpha = 0.5$, $\gamma = 0.9$

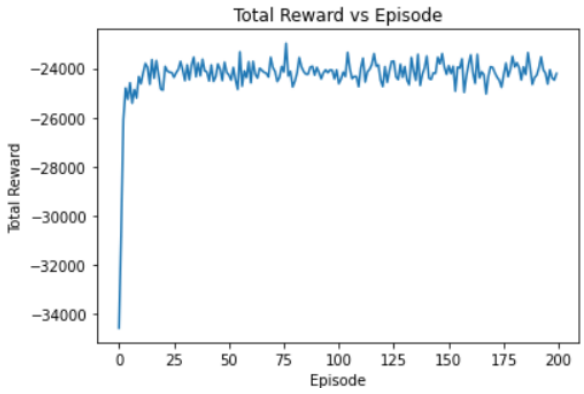


FIGURE 3 – $\alpha = 0.2$, $\gamma = 0.9$

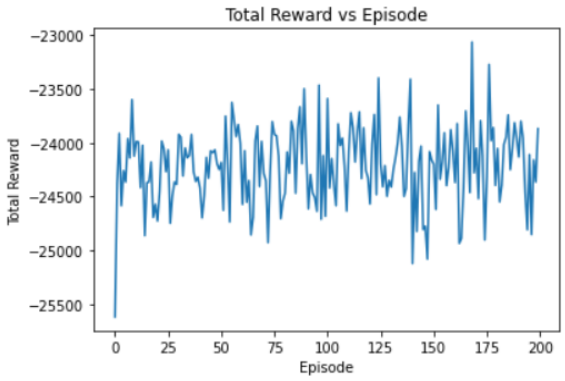


FIGURE 4 – $\alpha = 0.8$, $\gamma = 0.5$

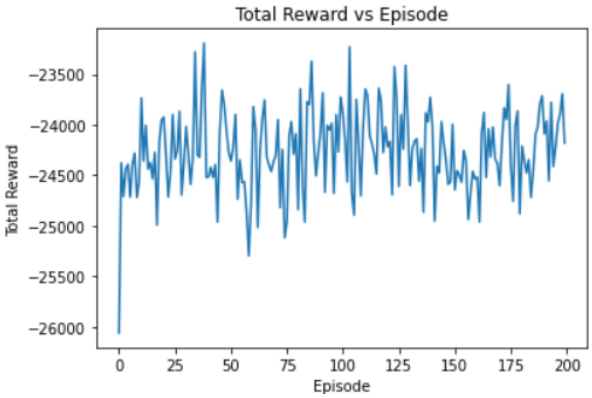


FIGURE 5 – $\alpha = 0.8$, $\gamma = 0.2$

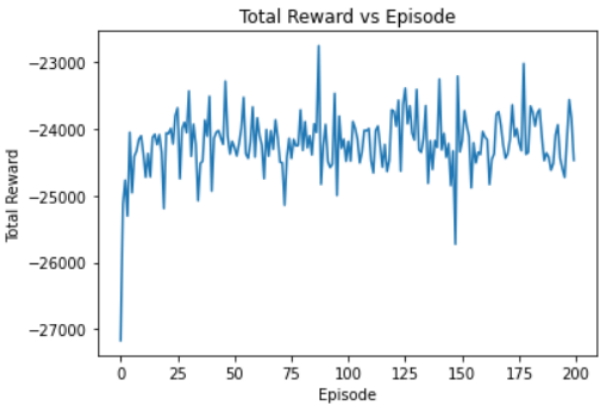


FIGURE 6 – $\alpha = 0.2$, $\gamma = 0.2$

Jouons maintenant sur le paramètre γ : le facteur d'actualisation, il permet d'équilibrer la récompense immédiate et future. Nous fixons ainsi la valeur de α à 0.8 et nous donnons à γ les valeurs : 0.9, 0.5 et 0.2, figure 1, 4 et 5 respectivement. Nous remarquons que plus on diminue le facteur γ, plus on a d'instabilité et de bruit qui s'instaure autour de la limite supposée de convergence, qui est autour des -24000 pour notre cas. En effet, la courbe 1 est bien bornée dans l'intervalle -25000 et -23000, avec quelques variations dûes à l'exploration mais on a bien la convergence. Cependant pour la figure 4 et 5, on a plus de variations, avec plus d'instabilité pour la figure 5 et on peut dire que notre modèle diverge pour ces 2 valeurs de γ. Cela s'explique par le fait qu'une valeur de gamma proche de 0, signifie qu'on accorde une importance minimale aux récompenses futures et on se concentre d'avantage sur la récompense immédiate. Cela est aussi dû à la nature du problème, en effet en voulant maximiser le reward, on doit minimiser le coût, cependant notre coût d'un état s=(m,n) dépend des états et actions précédents, ainsi notre γ doit être proche de 1 afin de prendre en considération cette dépendance et maximiser la récompense future.

Finalement, pour le cas de la figure 6, nous remarquons qu'en diminuant la valeur de α à 0.2, notre courbe tend à converger et on a plus de stabilité autour de -24000, cependant on a toujours des variations et des pics de temps en temps.

SARSA :

On se propose désormais de résoudre le problème en utilisant l'algorithme SARSA.
SARSA est un algorithme d'apprentissage par renforcement. Son nom est l'acronyme de State-Action-Reward-State-Action. En SARSA, on commence à l'état s_1, on effectue une première action a_1, et on obtient une récompense r_1. Une fois qu'on est dans l'état s_2 on effectue une autre action a_2 avant de mette à jour la valeur de notre Q-table pour l'état s_1 et l'action a_1. La différence avec le Q-learning, c'est qu'à la place de prendre la récompense future pour l'action qui a la plus grande valeur de Q, celle qui maximise $Q(s_2, action)$, on prend $Q(s_2, a_2)$.
On en tire alors, en suivant le même schéma de l'algorithme du Q-learning, l'algorithme suivant :

Algorithm 2 Algorithme SARSA

On initialise la table Q avec des valeurs arbitraires
for each $episode \in \{episodes\}$ **do**
    Initialiser l'état s
    Choisir l'action a avec la plus grande valeur Q dans l'état s ou avec une probabilité ϵ sélectionner une action aléatoire
    for each $pas \in \{episode\}$ **do**
        while Not atteinte d'un état terminal **do**
            Prendre l'action a, observer la récompense r et le nouvel état s'
            Choisir l'action suivante $next_{action}$ avec la plus grande valeur Q dans l'état s' ou avec une probabilité ϵ sélectionner une action aléatoire
            Mettre à jour la valeur Q en utilisant la formule :

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(r + \gamma Q(s', next_{action}) - Q(s,a)\right)$$

            Définir $s \leftarrow s'$
            Définir $a \leftarrow next_{action}$
        end while
    end for
end for

Après avoir codé cet algorithme sous python, nous avons tracé la courbe du reward total de chaque épisode en fonction du nombre d'épisodes. Nous prenons les mêmes paramètres fixés pour le Q-learning, et nous obtenons les figures suivantes :

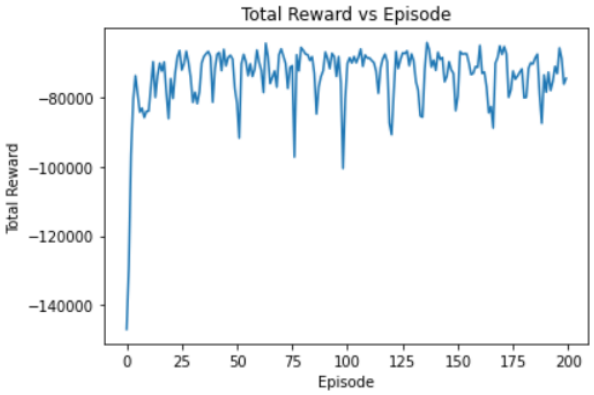


FIGURE 7 – $\alpha = 0.2$, $\gamma = 0.9$

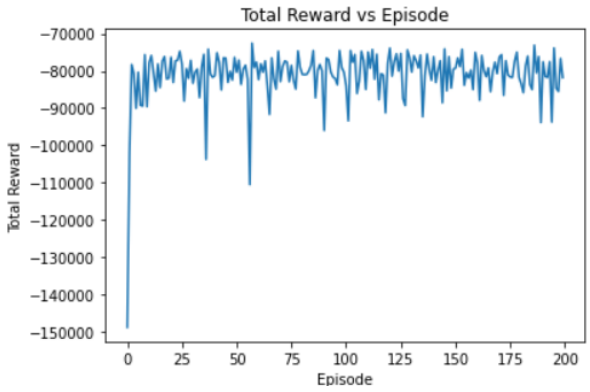


FIGURE 8 – $\alpha = 0.5$, $\gamma = 0.9$

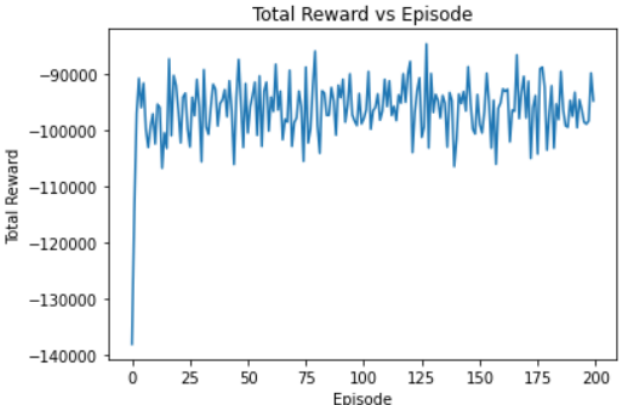


FIGURE 9 – $\alpha = 0.8$, $\gamma = 0.9$

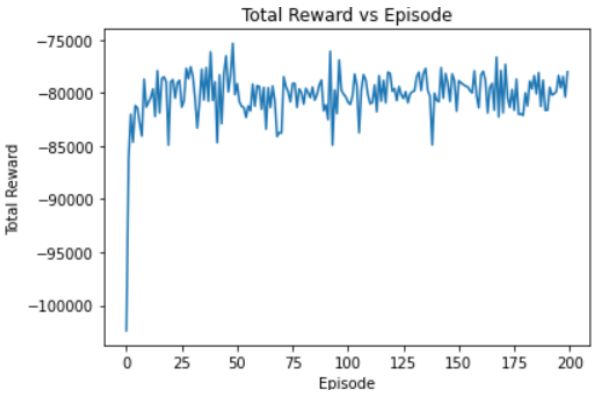


FIGURE 10 – $\alpha = 0.8$, $\gamma = 0.5$

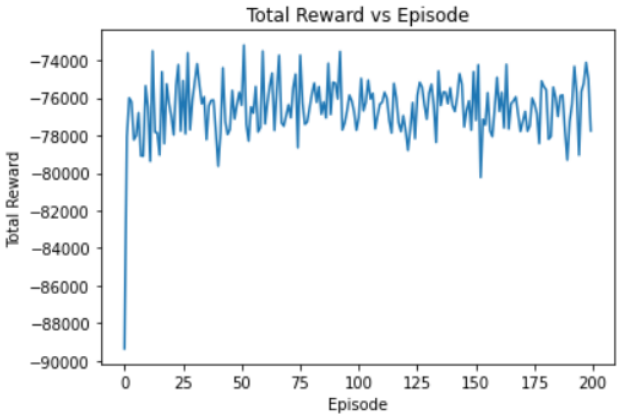


FIGURE 11 – $\alpha = 0.8$, $\gamma = 0.2$

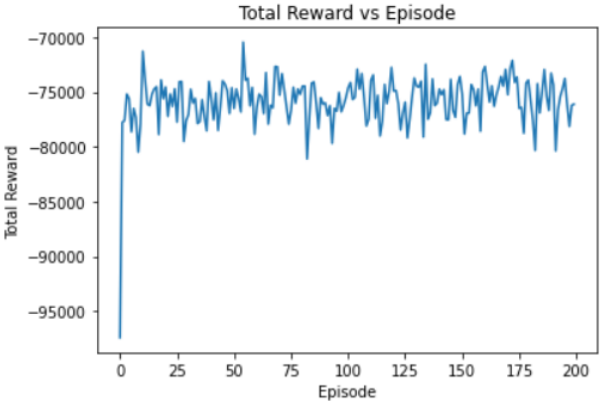


FIGURE 12 – $\alpha = 0.5$, $\gamma = 0.5$

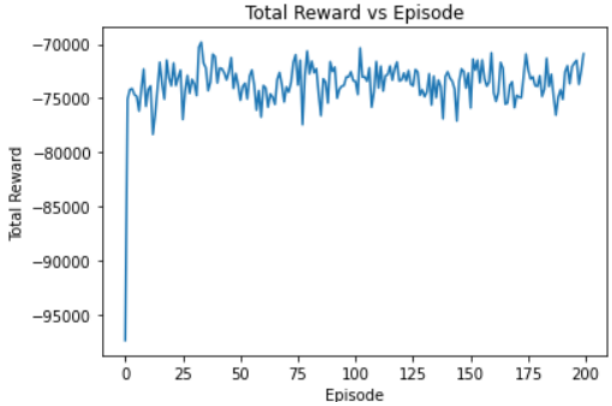


FIGURE 13 – $\alpha = 0.5$, $\gamma = 0.2$

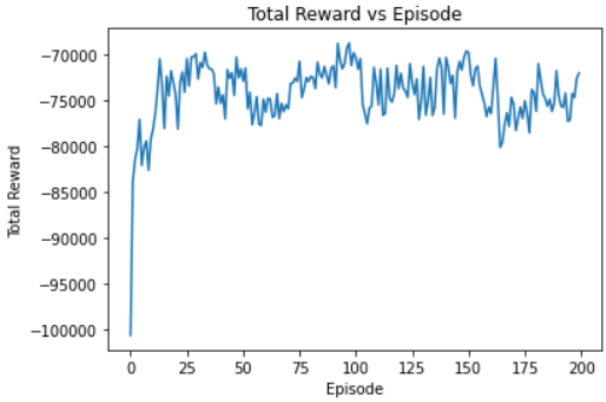


FIGURE 14 – $\alpha = 0.2$, $\gamma = 0.2$

Remarquons d'abord que notre modèle converge pour presque toutes les valeurs qu'on a prise de α et γ.
Pour γ fixée à 0.9, nous remarquons sur les 3 premières figures, que plus α est grand plus le reward

total est petit, et moins on a de variations entre 2 états successifs de plus que l'intervalle des valeurs du reward total est petit.

Fixons maintenant la valeur de α à 0.8 puis à 0.5 et à 0.2, et pour chacune de ces valeurs traçons les courbes associées à $\gamma = 0.9$, 0.5 et 0.2. Nous remarquons que plus γ est petit, moins on a de variance pour le reward totale, en effet, prenons le cas de $\alpha = 0.5$ ce qui correspond au figures 8, 12 et 13, on a pour la figure 8 où $\gamma = 0.9$ l'intervalle contenant la majorité des valeurs du reward total est [-90 000, -75 000], cependant en diminuant la valeur de γ, cet intervalle devient pour la figure 12 [-80 000, -76 000] pour une valeur de $\gamma = 0.5$, et se rétrécit encore plus pour $\gamma = 0.2$, figure 13, où il devient [-76 000, -72 000].
Regardons maintenant les figures 11, 13 et 14, où $\gamma = 0.2$ et α prend les valeurs 0.8, 0.5 et 0.2 successivement.
Nous remarquons encore que plus alpha est grand, moins on a de variations entre les valeurs du reward total, en général.

Après cette analyse, nous fixons α à 0.8, et γ à 0.2, nous remarquons (figure 11) qu'on a bien une convergence stable, où les valeur du reward total convergent vers une limite de -77 000, d'autant plus que la majorité des valeurs du reward total sont dans [-78 000, -75 000].

Comparaison entre les deux algorithmes :

Tout d'abord, la différence entre les deux algorithmes réside dans la manière dont ils mettent à jour leur table de Q-value. Dans le premier algorithme (Q-learning), la table de Q-value est mise à jour en utilisant la valeur maximale attendue des Q-values à l'état suivant. Dans le second algorithme (SARSA), la table de Q-value est mise à jour en utilisant la valeur Q-value de la prochaine action choisie plutôt que la valeur maximale attendue des Q-values.

Comparons maintenant les deux algorithmes en terme d'efficacité. D'après les différentes courbes que nous avons tracés, nous remarquons bien que le reward total obtenue grâce au Q-learning est bien supérieur à celui obtenu par l'algorithme SARSA.
En guise de comparaison, prenons maintenant la courbe 1 de l'algorithme Q-learning et la courbe 11 de l'algorithme SARSA. Remarquons que le reward total maximal pour la courbe de Q-learning est de -23 000, tandis que celui de SARSA est de -74 000. Nous remarquons aussi que la courbe obtenue grâce au Q-learning est plus stable et a moins de variance que celle de SARSA.

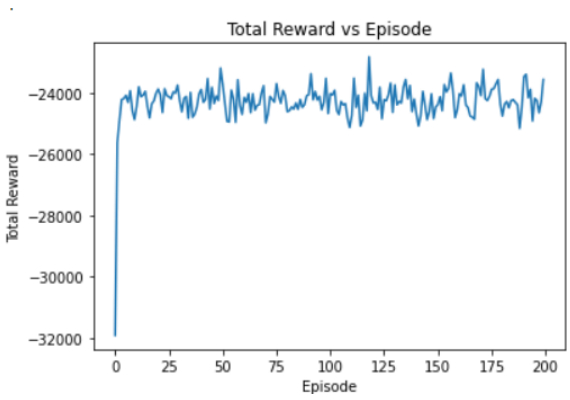


FIGURE 15 – Q-learning : $\alpha = 0.8$, $\gamma = 0.9$

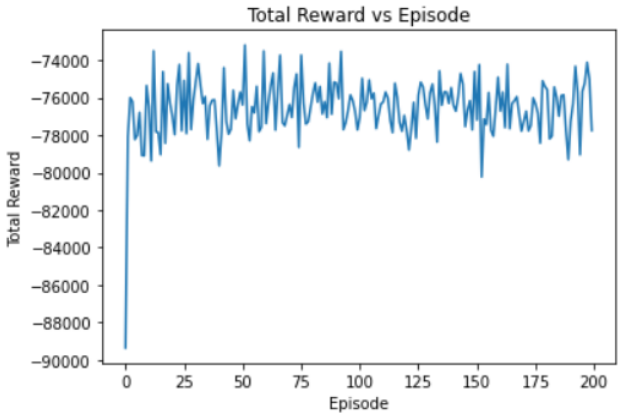


FIGURE 16 – SARSA : $\alpha = 0.8$, $\gamma = 0.2$

Explication possible :

L'une des explication possible est que le Q-learning est plus agressif en termes de mise à jour de la fonction Q. Il essaie de maximiser la récompense en choisissant toujours la meilleure action possible, tandis que SARSA est plus prudent et n'ajuste la fonction Q que pour l'action prévue. Cela entraîne une mise à jour plus lente de la fonction Q, ce qui peut expliquer pourquoi les valeurs du reward total pour SARSA sont plus faibles que celles du Q-learning. On peut aussi l'expliquer par le fait que notre système d'étude est incertain ou change rapidement, vu que les arrivés i suivent une distribution de probabilité (uniforme dans notre cas), dans ce cas, le Q-Learning peut rapidement s'adapter en explorant de nouvelles actions pour maximiser les récompenses futures, ce qui peut entraîner des valeurs de récompense plus élevées. SARSA, d'autre part, se concentre sur la mise à jour des valeurs en fonction des actions prévues, ce qui peut rendre la convergence plus lente.

On conclut que pour notre système d'étude, le Q-learning s'est avéré l'algorithme le plus efficace, vu qu'il maximise le reward et que c'est le plus rapide à converger.