LANSAR Mohammed
BADIANE Omar
TRIKI Bilelle

Rapport du Projet : Introduction à l'apprentissage

Introduction et motivation

Afin d'étudier en profondeur l'apprentissage par renforcement et après avoir traité le « cas » de l'$\varepsilon - greedy$ nous nous sommes penchés sur la résolution des équations de Bellman dans un premier temps. L'objectif ici est de calculer exactement de la fonction $v_\pi(s)$ puis d'améliorer au possible la politique π. On calculera ensuite les fonctions $v_\pi(s)$ de manière itérative pour enfin comparer les deux matrices obtenus (pour chaque méthode donc) et vérifier que ces dernières donnent le même résultat après amélioration de la politique. Ensuite dans un second temps, nous passerons sur une partie plus concrète de l'apprentissage par renforcement où ici nous implémenterons dans un jeu Pac-Man les méthodes $Q - Learning$ et $SARSA$. Nous comparerons les résultats obtenus pour ces deux approches avec le comportement aléatoire du Pac-Man afin de montrer l'efficacité de l'apprentissage par renforcement.

Organisation du code

Le projet Java contenu dans le dossier présent est divisé en deux classes :

- Le package **bellman** contient la classe **ResolutionBellman**. Cette dernière est composée des procédures utilisées dans le calcul des fonctions $v_\pi(s)$.
- Le package **rl** contient les classes utilisées pour faire fonctionner le Pac-Man : La majorité on était fournis et non modifié sauf pour les classes **PACMAN**, **Qlearn** et **TraitementDonnee** (que nous avons créé afin de stocker le nombre de mort, de nourriture et le mouvement vers le mur dans un fichier afin de l'exploiter sur Excel).

Pour la partie sur Bellman si **matricielle** de la ligne 22 est à true alors on affiche la politique optimale selon la méthode matricielle et si ce booléen est à false la politique optimale sera affiché de façon itérative. Pour le Pac-Man, si vous souhaitez utiliser la méthode $Q - Learning$ il faut passe le paramètre **use_SARSA** de la ligne 15 sur false ou true si vous désirez utiliser la méthode $SARSA$.

Résultats

I) Recherche de $v_\pi(s)$

Grâce aux travaux réalisé lors de la résolution des équations de Bellman nous avons donc pu nous lancer dans la détermination des fonctions $v_\pi(s)$. Nous avons pu voir que $v_\pi(s)$ peut s'exprimer de deux manières différentes : une matricielle et l'autre itérative. Ainsi nous allons dans un premier temps déterminer la matrice représentant $v_\pi(s)$ grâce à la méthode **SolvingP** qui utilise la formule suivante $V^\pi = (I - \gamma P^\pi)^{-1} r^\pi$. Ainsi grâce à cette dernière on obtient la matrice de départ (cf. Figure 1).

Dans un second temps, nous allons déterminer les valeurs $v_\pi(s)$ grâce à la méthode itérative donné par la méthode suivante : $V_{k+1}(s) = \sum_a \pi(s,a) \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V_k(s')]$. La grille des valeurs $v_\pi(s)$ alors obtenue correspond à la matrice du bas de la figure grâce à la méthode **VIteratif** (cf. Figure 1).

LANSAR Mohammed
BADIANE Omar
TRIKI Bilelle

Dans l'optique d'améliorer nos fonctions $v_\pi(s)$ nous avons donc implémenté la procédure **ImprovePolicy** qui a pour objectif de trouver la politique optimale par notre agent dans notre environnement.

On remarque que : que ce soit la méthode matricielle ou itérative, on obtient exactement la même grille représentant les fonctions $v_\pi(s)$ comme on peut le voir grâce à la figure suivante :

```
Forme matricielle : Calcul du V de départ

-90.1284330584166 -524.0223932073192 -2000.0 -620.796403914387 -527.3835640557259
-117.65567258714734 -479.1335043184303 -618.0731058074563 -2000.0 -1051.7840711737579
-2000.0 -521.5174892286001 -2000.0 -753.5020547413571 -1066.5291434878654
-839.2602366494325 -2000.0 -568.0017304702538 -2000.0 -1088.9162433768925
-2000.0 -1427.0289151832706 -2000.0 -582.4980850036852 -2000.0
-1040.744135650676 -2000.0 -2000.0 -1407.8312260625462 -666.4810345629145
-475.53121971991004 -271.7809774791909 -919.7534419421322 -2000.0 -2000.0
-2000.0

Forme matricielle : Calcul du V après amélioration de politique

200.0 99.0 -2000.0 4.3125 1.15625
-0.421875 200.0 200.0 -2000.0 10.625
-2000.0 -1.2109375 -2000.0 99.0 48.5
23.25 -2000.0 -1.60546875 -2000.0 48.5
-2000.0 10.625 -2000.0 -1.802734375 -2000.0
23.25 -2000.0 -2000.0 -1.95068359375 -1.9013671875
4.3125 10.625 4.3125 -2000.0 -2000.0
-2000.0
```

```
Forme itérative : Calcul du V de départ

-135.99754989329338 -597.2414246265269 -2000.0 -710.1921487719344 -611.6901794521137
-175.32928684497605 -546.7414246265269 -633.933847118922 -2000.0 -1061.847010723362
-2000.0 -605.6148284627186 -2000.0 -755.4217748172443 -1066.8725247283962
-840.4309477383227 -2000.0 -661.5893408567733 -2000.0 -1089.9896015078812
-2000.0 -1427.158994193147 -2000.0 -679.0998983914684 -2000.0
-1048.4846387536872 -2000.0 -2000.0 -1418.5788718083304 -763.2098462749748
-621.196021039329 -340.3721472753037 -1043.296518409413 -2000.0 -2000.0
-2000.0

Forme itérative : Calcul du V après amélioration de politique

200.0 99.0 -2000.0 4.3125 1.15625
-0.421875 200.0 200.0 -2000.0 10.625
-2000.0 -1.2109375 -2000.0 99.0 48.5
23.25 -2000.0 -1.60546875 -2000.0 48.5
-2000.0 10.625 -2000.0 -1.802734375 -2000.0
23.25 -2000.0 -2000.0 -1.95068359375 -1.9013671875
4.3125 10.625 4.3125 -2000.0 -2000.0
-2000.0
```

Figure 1 - A gauche la grille de départ et d'arrivé pour le calcul matriciel tandis qu'à droite nous avons les grilles pour la méthode itérative

II) Q-Learning/SARSA pour Pac-Man

Nous allons désormais nous intéresser à un cas un peu plus concret : l'implémentation de l'apprentissage par renforcement dans un jeu Pac-Man. Ainsi nous allons dans un premier temps justifier pourquoi nous avons modifié ou non certains paramètres du jeu ou de la méthode **Qlearn**. Dans un second temps nous présenterons et décrirons les données brutes tel quel. Enfin nous interprèterons ces résultats.

a) Choix des paramètres

Un point crucial de l'apprentissage (qu'il soit par renforcement ou supervisé) est le choix des paramètres qui peut avoir une grande influence sur les résultats obtenus et au pire peut rendre un programme « non convergent ». Il est important de souligner qu'à des fins de test des deux méthodes d'apprentissages sus cité, lorsque notre Pac-Man mange de la nourriture, ce dernier ne peut pas manger de fantôme : en effet cette règle est modifiée afin que nous puissions nous concentrer seulement sur l'aspect binaire du jeu, c'est-à-dire le nombre de mort et le nombre de nourriture mangé (nous expliciterons dans la partie suivante comment nous allons traiter le nombre de fois où le Pac-Mac se déplace vers un mur). Pour en revenir à nos paramètres, nous avons décidé de ne pas modifier ε, α et γ. Pour ce qui est des récompenses, nous avons testé avec plusieurs valeurs et nous nous sommes rendus compte que la méthode $SARSA$ ne « fonctionne pas très bien » avec les récompenses de base contrairement au Q-Learning qui n'a aucun mal quel que soient les récompenses (tout en restant cohérent). Ainsi nous avons établit que se faire manger par le fantôme entraine une pénalité de -40 et la nourriture une récompense de 80. Puisque $SARSA$ est On-Policy, le Pac-Mac va effectuer énormément d'exploration et va donc avoir de moins bonnes récompenses. Ainsi il faut inévitablement mieux le récompenser pour compenser cette exploration. Enfin nous avons fixé à -25 la pénalité lorsque le Pac-Man se rend vers un mur. Cette pénalité peut sembler élevé (la moitié de celle du fantôme) mais est en réalité justifié de part le fait que partir vers un mur réduit les chances de survie du Pac-Man puisque contre un mur, il est limité en termes d'action à effectuer. Nous ne travaillerons que sur la grille 2 car la grille 1 est limité en termes de déplacement

et d'obstacle et la grille 3 contient de la nourriture dans toutes les cases (cette dernière à été utile seulement pour calibrer et tester nos récompenses et méthodes **qLearning** et **SARSA**).

b) Données obtenues

Nous décidons de faire tourner notre algorithme sur 20000000 d'itérations afin de voir vers quel valeur nos différentes méthodes vont converger et surtout pour laisser le temps à la HashTable représentant le gain moyen de se construire. Nous utilisons notre méthode **impressionClasse** présente dans la classe **TraitementDonnee**.

Voici maintenant les courbes représentants le nombre de mort/nourriture/bloqué associés à chaque méthode et où les courbes de nuance rouges représentes les données lissés (grâce à une approximation polynomiale). On précise donc que l'axe des abscisses correspond aux nombres d'époques et l'axe des ordonnées le nombre soit mort, nourriture ou bloqué :

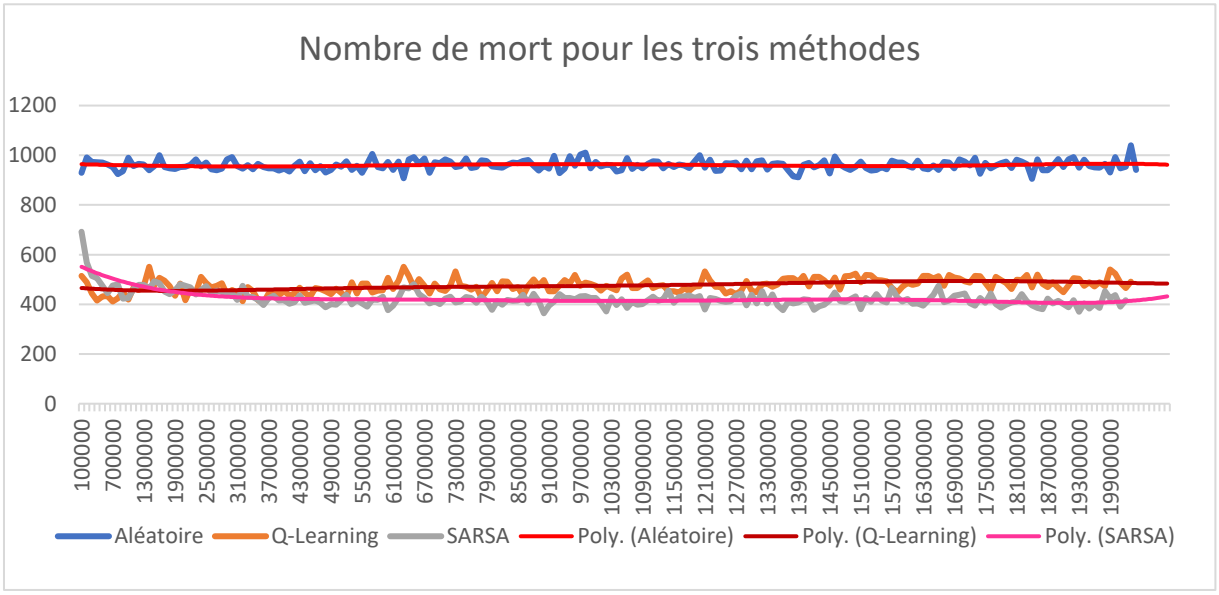

Figure 2 - Graphe représentant le nombre de mort en fonction des époques

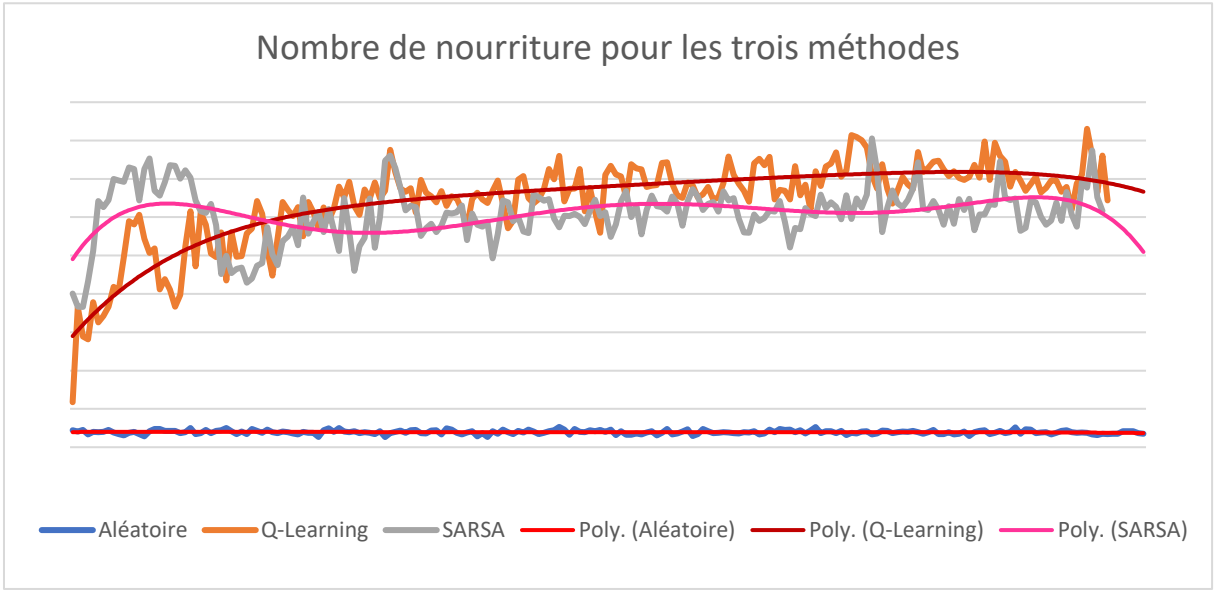

Figure 3 - Graphe représentant le nombre de nourriture en fonction des époques

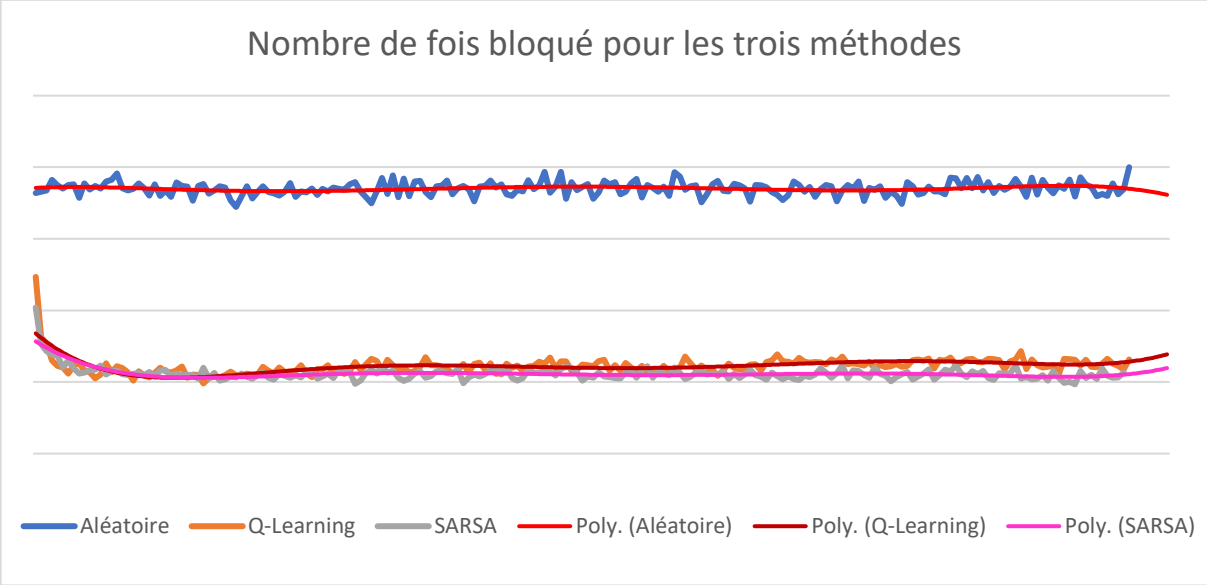

Figure 4 - Graphe représentant le nombre de fois bloqué en fonction des époques

c) Interprétation et conclusion

Grâce aux graphes si dessus, nous pouvons facilement comparer notre intelligence artificielle selon les améliorations implémentées. Tout d'abord nous pouvons observer que l'approche aléatoire donne des résultats très mauvais pour les trois types de récompenses. Comme nous pouvons le voir grâce aux courbes polynomiales qui approchent nos courbes de donnée, la méthode $SARSA$ donne un nombre de mort moins important que le $Q-Learning$ (en excluant les 100000 premières itérations où la méthode $Q-Learning$ est plus efficace. Ensuite pour le nombre de nourriture, la méthode $SARSA$ donne plus de nourriture jusqu'à la 2600000 itération et au-delà c'est la méthode du $Q-Learning$ qui est beaucoup plus efficace. Enfin le nombre de fois où le Pac-Man est bloqué ne varie pas trop entre ces deux méthodes. Ainsi comme on peut le voir, en prenant nos paramètres et nos récompenses, la méthode $SARSA$ est plus efficace que le $Q-Learning$ pour un nombre d'itération inférieur à 2600000 itérations mais ensuite c'est le $Q-Learning$ qui est meilleur en tout point. La méthode aléatoire quant à elle est inefficace en tout point. Ainsi on peut conclure que la méthode $SARSA$ est plus efficace que le $Q-Learning$ mais pour un très grand nombre d'itération la méthode $SARSA$ devient un peu moins efficace pour la nourriture : Ce phénomène étant surement dû à nos récompenses qui ont été choisis selon nos critères ou de part la disparité entre les méthodes de mise à jour qui est soit On-Policy soit Off-Policy.