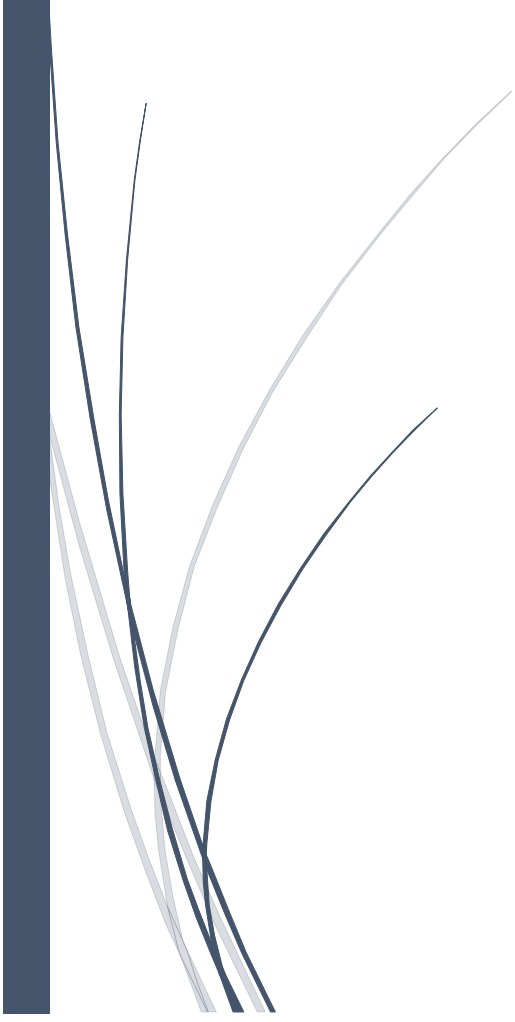
28/01/2019

COMPTE-RENDU DE TP SMA

Planification stochastique & MDP – Apprentissage par renforcement

Mark KPAMY - Anthony CHAPUT – Bastien JACOUD



INTRODUCTION

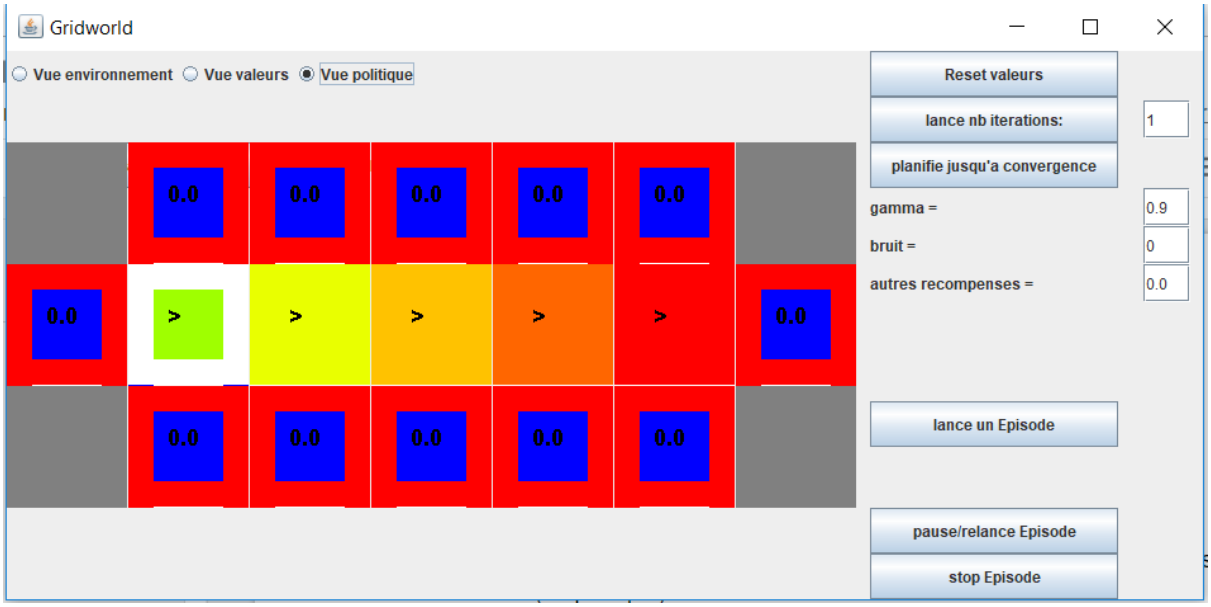
Ce document présente le compte-rendu des TP réalisées dans le cadre du cours de Systèmes multi-agents. Au cours de ces TP, nous avons étudié des algorithmes permettant à un agent donné de mettre en place une stratégie dans son environnement. L'objectif du premier TP a été d'implémenter l'algorithme de planification "Value Iteration" et d'étudier par la suite l'influence des différents paramètres sur la politique optimale obtenue. Le deuxième TP a eu d'objectif d'implémenter et tester dans divers environnements l'algorithme de Q-Learning.

Partie 1 : Planification stochastique & MDP

Influence des paramètres

Bridge Grid :

Il faut modifier le bruit en prenant 0 ou une très petite valeur, proche de 0. Ainsi, le robot sera certain (ou presque) d'atteindre sa destination sans tomber et donc sans obtenir un score négatif. Donc il sait qu'il peut se permettre de traverser le pont vu que le risque est très faible voir négligeable et atteindra l'autre bout du pont de manière certaine.



Discount Grid :

En partant de valeurs initiales lambda = 0.9, bruit = 0.2, Rother = 0.0, nous allons essayer d'obtenir une politique optimale qui suit un chemin sûr pour l'état absorbant de récompense +10.

Cas 1 - Politique optimale qui suit un chemin risqué pour atteindre l'état absorbant de récompense +1 :

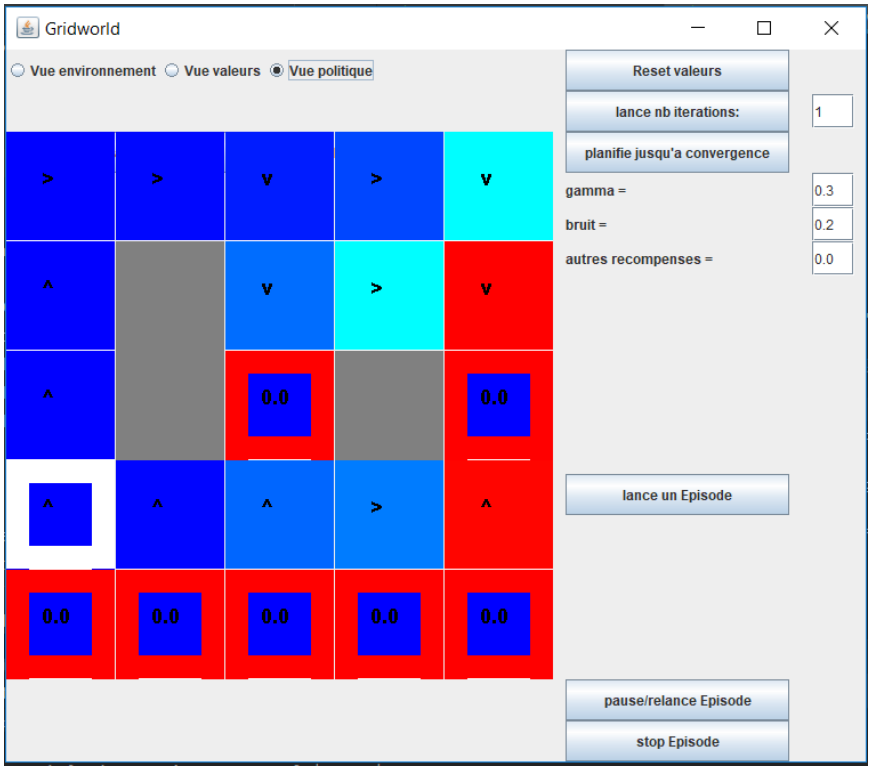
On choisit 0,1 (petite valeur) pour gamma et 0 (valeur négligeable) pour le bruit. Ainsi, on il peut utiliser un chemin risqué sans soucis (pas de bruit), et avec la valeur de gamma à 0,1 on va seulement atteindre la valeur +1 et ne pas être trop gourmand pour vouloir la valeur 10.

Cas 2 : Politique qui suit un chemin risqué pour atteindre l'état absorbant de récompense +10 :

On garde 0,9 (grande valeur) pour gamma et on choisit 0 (valeur négligeable) pour le bruit. Ainsi, on va pouvoir utiliser un chemin risqué sans soucis. Comme gamma vaut 0,9 on va vouloir obtenir la meilleure récompense (ex : la valeur plus élevée, en pratique comme elle est pondérée par gamma qui est plus proche de 1, son impact est plus important) possible donc aller obtenir le +10.

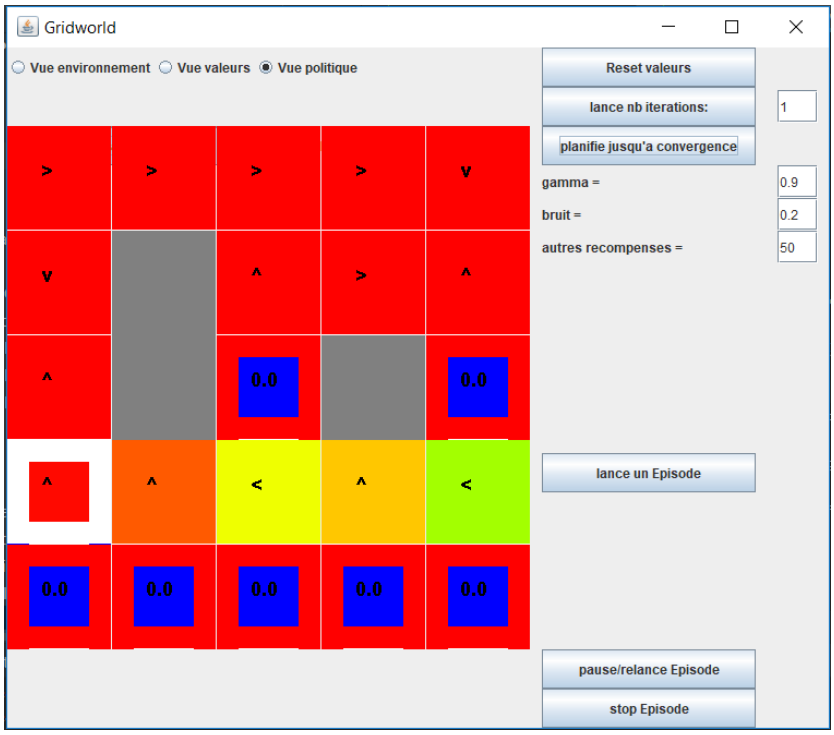
Cas 3 : Politique qui suit un chemin sûr pour atteindre l'état absorbant de récompense +1 :

On choisit 0,3 pour gamma et on garde 0,2 pour le bruit. Il faut que le bruit ne soit pas négligeable pour que le robot n'emprunte pas le chemin risqué mais le chemin sûr, contrairement aux cas 1 et 2. De plus, en mettant une valeur plutôt faible à gamma (0,3 au lieu de 0,9), on lui indique d'atteindre la valeur 1 plutôt que la valeur 10. Attention tout de même à choisir une valeur de gamma supérieur ou égale à celle du bruit, sinon le robot revient à sa position initiale.



Cas 4 : Politique optimale qui évite les états absorbants

On modifie les autres récompenses en lui mettant une valeur élevée (50 dans notre cas). Ainsi, les états initiaux auront une récompense petite par rapport aux autres cases. Notre robot ne cherchera donc pas à rejoindre un état absorbant. Pour cela, il va réaliser une boucle dans son chemin afin de ne jamais rejoindre un état absorbant.



Partie 2 : Apprentissage par renforcement

Dans cette partie du TP, nous avons implémenté l'algorithme du Q-learning. L'objectif est d'implémenter et de tester dans différents environnements cet algorithme. Nous avons donc en premier repris l'environnement du TP 1. Nous allons principalement expliquer dans cette partie les éléments utilisés pour définir un état MDP pour le jeu du Pacman et les fonctions caractéristiques choisies pour la classe FeatureFunctionPacman.

Changements apportés

Critères retenus pour discriminer chaque état :

- *canEscape* : Paramètre booléen indiquant si le Pacman possède une issue en cas de présence proche d'un fantôme
  - Vérification sur un rayon de deux cases de la présence d'un fantôme et d'une case libre
- *isDotAccessible* : Paramètre booléen indiquant la présence d'une gomme et si elle est accessible
  - Vérification sur deux cases de distance
  - Directement faux si le Pacman est dans un cul-de-sac
- *onDanger* : Paramètre booléen indiquant si el Pacman est en danger
  - Fantôme adjacent à un Pacman
  - Pacman dans un cul-de-sac
- *isDeadEnd* : Paramètre booléen indiquant que le Pacman est dans un cul-de-sac
  - Détection du nombre de murs (si 3 cases mur, présence d'un cul-de-sac)

- *isLose*, *isWin* : Paramètres booléens pour la victoire ou la défaite de la partie

Construction de *FeatureFunctionIdentity*

Nous construisons un tableau de (nombre d'états * nombres d'actions) pour pouvoir utiliser cette classe. En l'occurrence, *EtatPacmanMDPClassic* peut prendre la forme de 24 états différents (4 paramètres cumulatifs = factorielle de 4) et de 4 actions.

Pour pouvoir attribuer une valeur à 1 unique en fonction de l'état et de l'action, nous allons passer en binaire en devinant le numéro de la case selon les paramètres :

- Si le paramètre 1 activé : premier bit à 1
- Si le paramètre 2 activé : deuxième bit à 1
- …

Pour faire varier selon l'action, nous allons mettre la valeur de l'état dans les bits les plus à gauche vacants (bits pas encore utilisés). Enfin, ce nombre binaire nous donnera la case à modifier selon l'état et l'action, ce qui nous garantit de toujours tomber sur la même case.

Fonctions caractéristiques choisies

Au niveau du choix des fonctions caractéristiques pour la classe *FeatureFunctionPacman*, nous avions d'abord testé les fonctions caractéristiques données en exemple dans les documents, à savoir :

- Le biais
- Le nombre de fantômes pouvant atteindre le Pacman à un pas
- La présence d'une gomme à la position future d'un Pacman
- La distance à la plus proche gomme depuis la position future du Pacman

Nous n'avons malheureusement pas eu le temps d'effectuer des tests sur d'autres fonctions caractéristiques.

Conclusion

Ces deux TP nous ont permis d'implémenter et de comprendre le fonctionnement d'algorithmes d'apprentissage notamment le Q-learning et MDP. Nous avons pu observer le comportement d'un agent suivant les différents algorithmes et analyser la courbe d'apprentissage. Néanmoins, nous avons rencontré des difficultés sur la dernière partie du TP 2 en ce qui concerne la définition de features et l'implémentation de la classe FeatureFunctionIdentity.