RAPPORT APPRENTISSAGE ARTIFICIELLE 2021 – 2022

Nom : TOURE
Prénom : Boubacar
Numéro étudiant : 18 00 94 22
Niveau : Master 1 informatique
Date : 10 Avril 2022 à Angers
Professeur : Monsieur Olivier GOUDET
Cours : Apprentissage Artificielle
Objectif : Implémenter les premières stratégies d'apprentissage pour le *Pacman* afin de lui permettre d'agir de manière précise et réfléchie grâce à l'intégration d'un système d'apprentissage automatique mise en place par des algorithmes d'Intelligence Artificielle.

GitHub :
https://github.com/TBoubacar/ProjetApprentissageArtificiellePacman.git


FACULTÉ
DES SCIENCES
UNIVERSITÉ D'ANGERS

SOMMAIRE

I.Introduction

■ Description

Dans le cadre de la réalisation de notre projet, nous avons pour but d'implémenter des algorithmes d'apprentissages artificielles afin de permettre aux agents ***Pacman*** de pouvoir prendre des décisions et d'apprendre des erreurs qu'ils commentent tout au long de l'avancé du jeu.

Pac-Man initialement intitulé ***Puck-man,*** est un jeu vidéo créé par une entreprise japonaise Namco, sorti au Japon le 22 mai 1980. Le jeu consiste à déplacer l'agent Pac-Man à l'intérieur d'un labyrinthe, afin de lui faire manger toutes les pac-gommes qui s'y trouvent en évitant d'être touché par des fantômes.

Pour la réalisation de cette expérience, nous avions en notre disposition un projet préalablement implémenté. Le rôle qui nous était dédié était lié au faite de faire accroître le système IA en charge du lancement et de l'apprentissage automatique de mouvements des joueurs. De ce fait, notre responsabilité était de faire évoluer la partie du projet concernant l'IA avec pour but de rendre le jeu *Pacman* plus attractif, continue et évolutif.

Pour chaque méthode d'apprentissage implémentée, il nous a été fourni des fichiers sources permettant de lancer l'application via l'interface graphique en mode :

- Debug (**main_debugMode**)
- Standard (**main_standardMode**)
- Haut niveau (**main_batchMode**)

II. Stratégies d’apprentissages

Au cours de la mise en pratique de notre projet, nous avons appliqué les quatre algorithmes qui nous a été demandé. Des séries de tests ont été réalisé sur ces différents algorithmes dont nous vous exposerons les détails ci-dessous. Parmi les différentes algorithmes que nous avons implémenté, nous avons :

- **TabuLar Q-Learning Strategy**

En intelligence artificielle, plus précisément en apprentissage automatique, le Q-learning est une technique d'apprentissage par renforcement. Cette méthode d'apprentissage permet d'apprendre une stratégie, qui indique quelle action effectuer dans chaque état du système. Elle fonctionne par l'apprentissage d'une fonction de valeur état-action notée *Q* qui permet de déterminer le gain potentiel, c'est-à-dire la récompense sur le long terme, *Q(s,a)* , apportée par le choix d'une certaine action *a* dans un certain état *s* en suivant une politique optimale. Lorsque cette fonction de valeur d'action-état est connue ou apprise par l'agent, la stratégie optimale peut être construite en sélectionnant l'action à valeur maximale pour chaque état, c'est-à-dire en sélectionnant l'action *a* qui maximise la valeur *Q(s,a)* quand l'agent se trouve dans l'état *s* (***voir un exemple d’un Q-Table***).

A l’aide de l’algorithme **TabuLarQLearning,** nous avons remarquer que l’agent *Pacman* apprend au fur et à mesure que le jeu avance. Il retient chaque bonne et mauvaise action qu’il a eu à effectuer, ce qui lui donne la possibilité d’éviter certaines erreurs qu’il a commis précédemment.

Cette méthode possède des inconvénients car plus le champ d’action sur le terrain de jeu est grand, plus cet algorithme requiert de la mémoire ainsi que de l’espace sur notre système.

Pour pallier (*un petit peu*) à ce manquement, j’ai eu à implémenter un attribut de type **Hashtable** afin de contenir l’ensemble des états explorés par nos agents *Pacman* durant tout le long de l’exécution de notre programme. Ce qui nous permettrait d’effectuer au minimum un certain de nombre de tour de jeu même sur des terrains de grandes tailles car les variables de types **HashTable** on la faculté de pouvoir stocker en mémoire que les espaces dont ils ont besoin, contrairement au tableau statique. Imaginons l’usage de notre **TabularQLearning** sur un terrain de taille 1 million par exemple. Et à coté de l’agent *Pacman* se trouve une capsule est un seul agent fantôme à l’autre bout du terrain. Grâce au faite que notre **Hashtable** est vide de base et ne requiert aucune donnée de base pour pouvoir être instancier. Notre jeu pourrait être lancer malgré la taille gigantesque du terrain et l’agent pourrait remporter la partie en mangeant la seule ***Capsule*** présente sur le terrain. Ce qui ne

serait pas le cas pour tout ceux qui feront le choix des tableaux dont la mémoire est allouée statiquement dans le constructeur avant le lancement de la partie.

- # Approximate Q-learning Strategy

Tout comme le **TabularQLearning, l'Approximative Q Learning** est aussi une approche d'apprentissage Artificielle permettant de pallier au problème rencontrer dans l'exécution de l'algorithme du **TabularQLearning.** A l'aide de l'algorithme **ApproximateQLearningStrategy,** l'approche est totalement différente. Pour la mise en place de cet algorithme, on procède à une extraction des caractéristiques, appelée « ***features*** ». Contrairement au **TabuLarQLearning**, on utilise un vecteur de poids linéaire permettant de prédire les meilleures actions à effectuer. Cette nouvelle méthode de faire nous permet de gérer au mieux l'espace mémoire allouable lors de l'exécution de notre programme. Ce qui rend cet algorithme plus apte à faire des tests sur des terrains de jeu de grandes envergures (grande taille tel que : « ***mediumClassic.lay*** »).

L'objectif de cette approche est d'ajuster les paramètres W du modèle Qw de façon à minimiser l'erreur(W). De plus, les connaissances qu'on a appris sur des états déjà explorés peuvent être utilisées pour traiter des états non explorés. Le comportement est plus robuste : on fait des choix similaires pour des états similaires.

Le seul inconvénient dans cette méthode est qu'il demande une extraction des features fi (s, a) à chaque itération et pour chaque action possible, ce qui peut être coûteuse. De plus, la qualité du programme dépend des différentes features qui seront implémentées. Plus votre feature sera performant, plus votre algorithme sera puissant et précis.

Parmi les différentes ***features (4 quatre)*** que j'ai eu à implémenter, nous avons :

- **Feature 1 :** Cette feature permet de déterminer si des nourritures sont à un mètre de notre agent ***Pacman.*** Si c'est la cas, alors on envoi 1 sinon 0. Pour améliorer notre programme, nous avons créer une autre fonction permettant de repérer plus de nourriture sur les lignes et colonnes sur laquelle l'agent *Pacman* est situé et nous l'avons utiliser dans le calcul des features.
- **Feature 2 :** Cette feature utilise la fonction calculCoutByRadarPacgum(). Comme son nom l'indique, elle nous permet de renvoyer le poids correspondant au différente P*acgum repérer dans notre champ de visuelle.*
- ***Feature 3 :*** Cette feature nous permet de connaître l'ensemble des mouvements possibles sur notre terrain et de retourner un poids pour chacune des actions appliquées.
- **Feature 4 :** Cette feature nous permet de déterminer si nous avons la possibilité d'éviter à nos P*acmans* de faire des mouvements inutilement. On peut considérer cela comme une petite amélioration de la fonction suivante : **isLegalMove(...)**

- **Approximate Q-learning With NN Stratégie**

A l’instar du **Approximate Q-learning, Approximate Q-learning With NN Stratégie** est aussi un algorithme d’apprentissage très avancée mais qui offre plus de libérer que son précédecesseur. Il apporte un remarquable changement dans l’implémentation de son code source notamment avec la suppression du vecteur W. L’avantage de cette approche est qu’on n’aurait plus à recharger des features à chaque appel de la fonction chooseAction. Chose qui pouvait être assez coûteuse car s’il faut créer un tableau de features à chaque fois que notre programme fait appel à la fonction chooseAction, on pourrait se retrouver à créer des tableaux pouvant contenir autant de features qui sera implémentée.

L’usage de cette algorithme se fera dans le scripts « **main_batchMode »**

J’ai aussi essayé d’implémenter le code du DeepQLearning, mais malheureusement je n’ai pas trouvé de solution adaptée pour pouvoir pallier au donnée et information qui me manque pour le réussir. J’avais pensé à modifier la structure de base de certaine de vos classes déjà implémenté afin de pouvoir gérer le cas où plus on considérer plus de deux états et d’actions possible dans notre terrain de jeu. Mais finalement, j’ai décidé de rendre le projet ainsi pour ne pas créer de bug dans certaine partie dont je n’ai aucune maîtrise du code et de l’importance de certaines fonctions.

III. Conclusion

Pour mettre fin à mon projet, je tiens à faire partager quelques petites informations directement liées à mon code. Il arrive des fois que le programme arrête de tourner brusquement sans que je ne comprenne rien malgré le fait que le minuteur de l'interface de commande continue de se mettre à jour.

Je vous affichera en image quelques images afin de vous faire part des résultats que j'ai pu obtenir lors de mes nombreux tests.

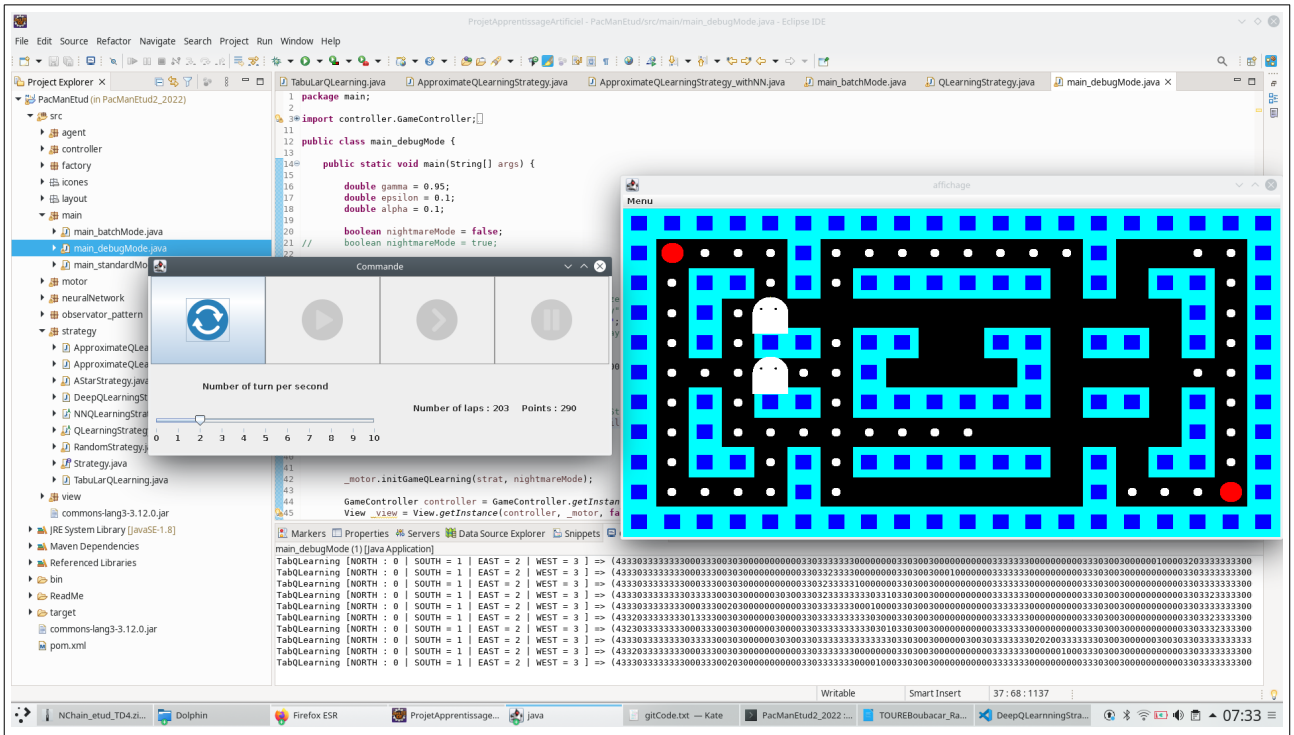

Figure : Tabular Qlearning

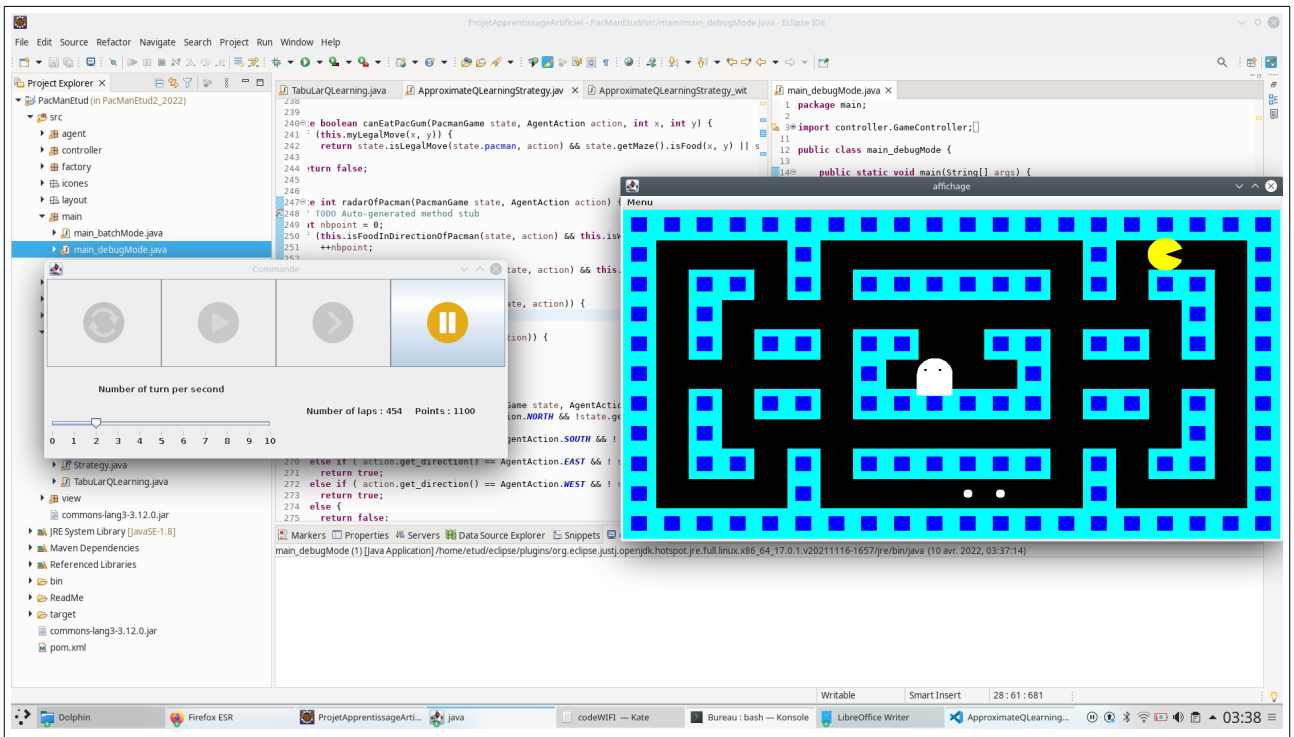

Figure : Approximative Qlearning

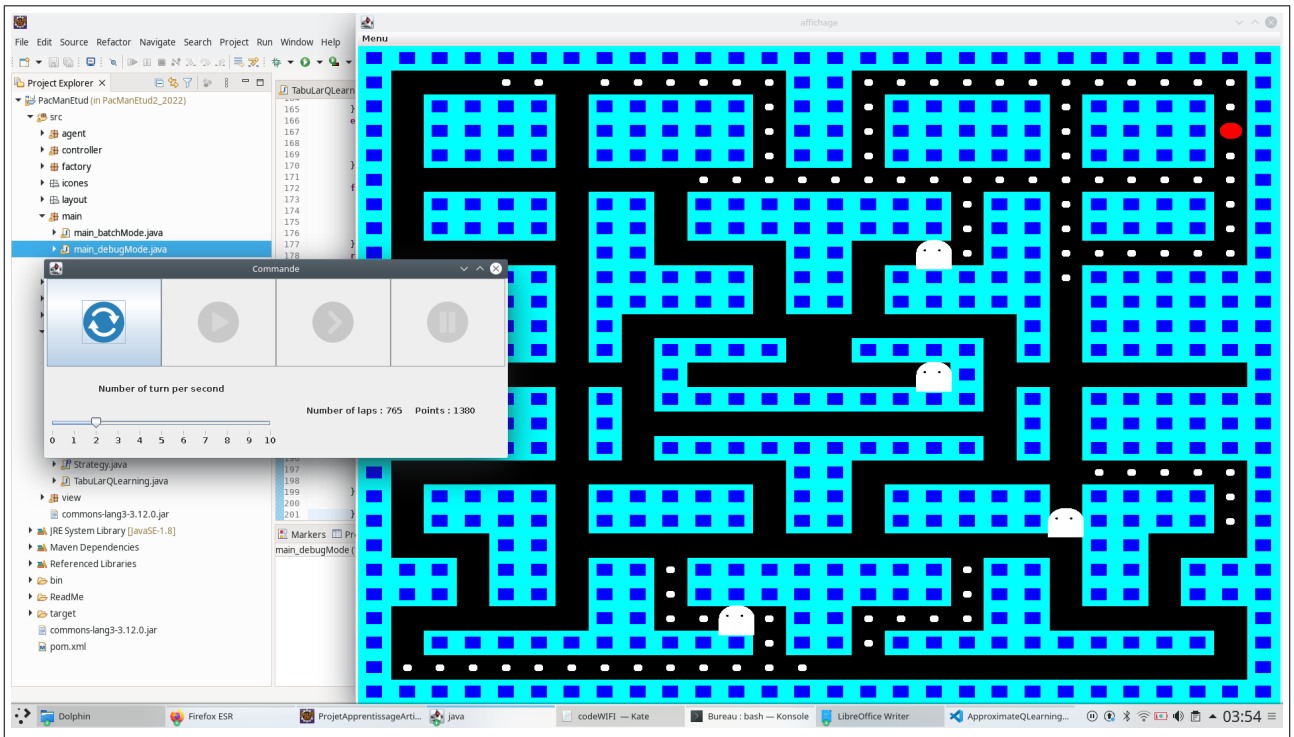

Figure : Approximative Qlearning With NN

IV. Annexes

Initialized

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	327	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	499	0	0	0	0	0	0

Training

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603

Figure : Exemple d'un Q-Table