IN104 : Résoudre un labyrinthe par de l’Apprentissage par Renforcement

**Introduction**

En intelligence artificielle, l’apprentissage par renforcement fait référence à une classe de problèmes d’apprentissage automatique, dont le but est d’apprendre, à partir d’expériences, ce qu’il convient de faire en différentes situations, de façon à optimiser une récompense quantitative au cours du temps.

Un paradigme classique pour présenter les problèmes d’apprentissage par renforcement consiste à considérer un agent autonome, plongé au sein d’un environnement, et qui doit prendre des décisions en fonction de son état courant. En retour, l’environnement procure à l’agent une récompense, qui peut être positive ou négative.

L’agent cherche, au travers d’expériences itérées, un comportement décisionnel (appelé stratégie ou politique, et qui est une fonction associant à l’état courant l’action à exécuter) optimal, en ce sens qu’il maximise la somme des récompenses au cours du temps.

Q-Learning :

Cette méthode d'apprentissage peut être appliquée pour trouver une suite d'actions associées à des états (politique) d'un [processus de décision markovien](https://fr.wikipedia.org/wiki/Processus_de_d%C3%A9cision_markovien) (fini) quelconque. Cela fonctionne par l'apprentissage d'une [fonction de valeur d'état](https://fr.wikipedia.org/w/index.php?title=Fonction_de_valeur_d%27%C3%A9tat&action=edit&redlink=1) qui permet de déterminer le potentiel bienfait (récompense) de prendre une certaine action dans un certain état en suivant une politique optimale.

La politique est la règle de sélection des actions successives d'un agent dans l'état actuel du système. Lorsque cette fonction de valeur d'action-état est connue, la politique optimale peut être construite en sélectionnant l'action à valeur maximale pour chaque état. Un des points forts du {\displaystyle Q}-learning est qu'il permet de comparer les récompenses probables de prendre les actions accessibles sans avoir de connaissance initiale de l’environnement. Cette notion d’apprentissage par récompense a été introduite à l'origine dans la thèse de Watkins en 1989[2](https://fr.wikipedia.org/wiki/Q-learning#cite_note-2). C'est une variante de l'apprentissage par différence temporelle[3](https://fr.wikipedia.org/wiki/Q-learning#cite_note-3). Par la suite, il a été prouvé, que {\displaystyle Q}-learning converge vers une politique optimale, à comprendre dans le sens de maximiser la récompense totale des étapes successives[4](https://fr.wikipedia.org/wiki/Q-learning#cite_note-4).

**Architecture**

Diagramme de classes

Classe Environnement

Le labyrinthe a été modélisé par une matrice numpy de taille fixe, où les différents états (« state ») des cases sont représentés par des entiers numérotés de 0 à 3,

La bibliothèque numpy a été utilisée afin d’optimiser la puissance de calcul les correspondances étant décrites ci-dessous :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| « State » | 0 | 1 | 2 | 3 |
| Signification | Case disponible | Mur | Départ | Arrivée |

Cette matrice d’états « states » caractérise donc le labyrinthe mais ne possède pas la position de l’agent à chaque étape. Cette dernière est toutefois présente dans la classe par le biais de « current\_position ».

**Vie du projet**

*Activité du 17/04 au 24/04 :*

Début de la programmation de l’environnement.

Création de la classe « state » pour caractériser l’état d’une case dans le labyrinthe. Le labyrinthe sera modélisé par une matrice numpy de taille fixe, où les différents états (« state ») des cases sont représentés par des entiers numérotés de 0 à 3, les correspondances étant décrites ci-dessous :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| « State » | 0 | 1 | 2 | 3 |
| Signification | Case disponible | Mur | Départ | Arrivée |

Importation et adaptation d’une classe AgentRandom en prévision de futurs tests simples de notre environnement.

*Activité du 28/04 au 01/05 :*

Ajout de méthodes dans la classe « state » afin de créer aléatoirement un état pour une case en vue de créer un labyrinthe totalement aléatoire. Certaines précautions (murs sur les bordures sauf départ et arrivée) sont prévues en cas de case en bordure du labyrinthe ou à l’intérieur du labyrinthe.

Ajout de méthodes pour déterminer et aléatoirement des positions possibles de départ sur la bordure puis en sélectionne un au hasard pour le départ et un autre pour l’arrivée. D’autres méthodes permettent de tester si la position est le départ ou l’arrivée.

On réalise que toute la suite de notre projet ne peut avancer sans une méthode « possibleActions » essentielle. On commence alors à implémenter cette méthode et on modélise les actions (Nord, Est, Sud et Ouest) par un couple (i,j) à ajouter à la position courante.

Implémentation de la méthode « show » afin de visualiser le labyrinthe. Détermination de la position de l’agent à l’aide de « current\_position » et on affiche un A pour cet agent à la place.

Ajout de l’architecture de runStep() et des récompenses dans l’environnement.

*Activité du 02/05 au 09/05 :*

Prise de recul sur le projet afin de réaliser sur papier l’architecture des différentes méthodes restantes à implémenter dans l’environnement ainsi que l’interaction avec l’agent.

*Activité du 10/05 au 16/05 :*

Finalisation de create\_random\_environment et implémentation du caractère aléatoire et avec 25% de chances d’avoir un mur sur une case à l’intérieur du labyrinthe.

Finalisation du premier Agent Random et tests sur l’environnement aléatoire.

Debugging et tests de l’environnement et des récompenses.

*Activité du 17/05 au 23/05 :*

Création de l’architecture de l’agent Q-Learning. Le précédent système de récompense n’était pas optimal ni adapté, on a donc revu le système en y ajoutant une matrice de récompense représentant les récompenses de chaque position.

Implémentation des différentes méthodes de l’agent grâce à l’algorithme fourni dans les documents.

*Activité du 24/05 au 30/05 :*

Construction d’une classe Agent en général. Révision de la structure de la matrice de qualité, il s’agit désormais d’une matrice numpy à 3 dimensions afin d’utiliser les performances de calculs de numpy.

Ajout des principales méthodes afin d’implémenter la classe AgentQLearning, puis réalisation de quelques tests unitaires de ces méthodes pour vérifier leur validité.

Correction de quelques bugs apparus lors des tests unitaires et tests globaux de la classe ainsi que de la classe environnement. Initialisation des qualités initiales des murs à 10-6. Commencement des tests globaux sur des labyrinthes aléatoires ainsi qu’un labyrinthe fourni en .txt, en implémentant une fonction qui permet de lire le fichier.

*Activité du 01/06 au 07/06 :*

Derniers tests globaux ainsi que des tests de performances de l’Agent Q-Learning et de l’influence des paramètres sur le ratio de victoires pour un nombre d’épisodes donné. Retour sur la classe d’Agent en général et correction de différents problèmes rencontrés. Avancement majeur sur le rapport final.

*Activité du 08/06 au 12/06 :*

Finalisation du rapport et des différents tests de performances. Préparation des supports pour la soutenance.

**Expérience sur GIT**

Rajouter stats

**Performances**