Revue de Littérature

Le génie pour l'industrie

25/10/2022

Léo LAVAL – SYS819

Table des matières

Introduction

Dans le cadre du cours d'apprentissage profond, chaque élève doit produire un projet en lien avec la matière. Dans mon cas, il est question d'un algorithme permettant de trouver le chemin le plus efficace dans un labyrinthe possédant plusieurs chemin vers la sortie. Le principe est de poser aléatoirement le robot dans le labyrinthe et quelque soit sa position initiale il puisse trouver la sortie. Ce projet s'inscrit dans ma formation de roboticien et est une part essentielle du savoir faire d'un ingénieur dans ce domaine. L'objectif n'est pas d'optimiser le temps de calcul mais bien de trouver la solution la plus proche de la perfection. L'optimisation du temps de calcul peut être vu comme un objectif secondaire de mon projet mais n'est pas un critère décisif dans le choix de la méthode que je vais choisir d'utiliser.

Définition de la tâche

Le labyrinthe se compose de plusieurs types de case. Tout d'abord la première qui représente les murs infranchissables que le robot doit éviter. Ensuite, il y a la case de départ du robot et la case de sortie. Enfin toutes les autres cases sont des cases de déplacement. L'entrée du programme est donc la position du robot dans la carte ainsi que son point de sortie. La sortie du programme est la suite des positions que le robot a parcourue et donc le nombre de déplacement de cases en case que celui-ci a réalisé. L'objectif sur une carte donnée est d'attendre le nombre de déplacement le plus bas possible. On pourra alors étudier l'évolution dans le temps du nombre de case parcourue. L'objectif est d'obtenir une courbe qui tends vers une valeur fixe qui sera déterminé comme la valeur minimale de déplacement à faire pour sortir du labyrinthe.

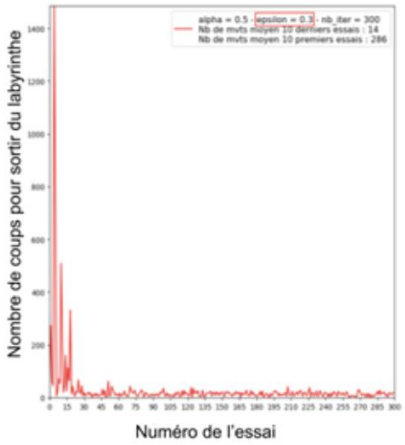

Figure 1 : exemple de courbe[1]

[1] Publicis Sapient, 2019, Johan Jublanc, consulté le 18/10/2022.

Bases de données

Afin de générer suffisamment d'espace de travail pour le robot, il faut posséder un algorithme de génération de labyrinthe avec plusieurs sortie possible. La difficulté de cette tâche réside dans le fait d'avoir plusieurs chemins possibles. En effet il est simple de trouver divers algorithmes sur internet permettant de générer des labyrinthes parfaits, c'est-à-dire des labyrinthes ne possédant aucune boucle fermée avec un seul chemin de résolution possible, mais il est moins aisé de trouver des algorithmes permettant de générer des labyrinthes avec plusieurs chemins possible. De plus la solution de générer aléatoirement un tas d'obstacle sur la ligne du robot peut conduire à un circuit non résoluble. Enfin il faut que l'algorithme puisse générer suffisamment de labyrinthe afin d'avoir des tests probants et ce sans prendre trop de temps.

Plusieurs générations sont possibles :

- La génération d'un labyrinthe parfait duquel on enlève un nombre prédéfini de murs.
- La génération d'une carte vierge sur laquelle on ajoute un nombre défini de murs.

La première méthode a le luxe d'être simple à mettre en œuvre. Il suffit d'implémenter un code de génération de labyrinthe et d'enlever aléatoirement des murs dans le labyrinthe ce qui va créer plusieurs chemins possibles. Plus de murs seront enlevé plus il y aura de chemin possible pour résoudre le labyrinthe car le nombre de chemins évolue exponentiellement par rapports aux murs enlevés. De plus un chemin aussi restrictif qu'un labyrinthe peut poser des problèmes aux algorithmes et le robot se retrouverait coincé à faire des aller-retours sans cesse dans le labyrinthe.

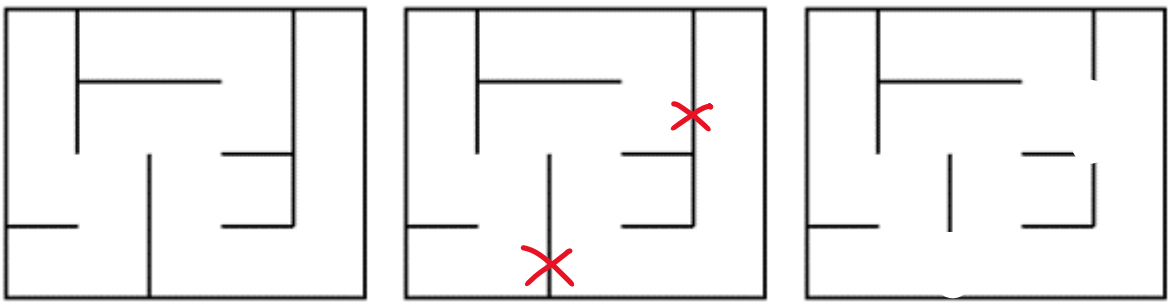

Figure 2 : exemple de labyrinthe parfait modifié [2]

[2] Wiki du Lama – Univ. Savoie, 2016-2017, Xavier PROVENCAL & Candice ROBERT

La deuxième méthode, quant à elle, permet de générer des chemins plus simples à décoder pour les algorithmes mais pourrait entrainer des blocages sur la trajectoire du robot.

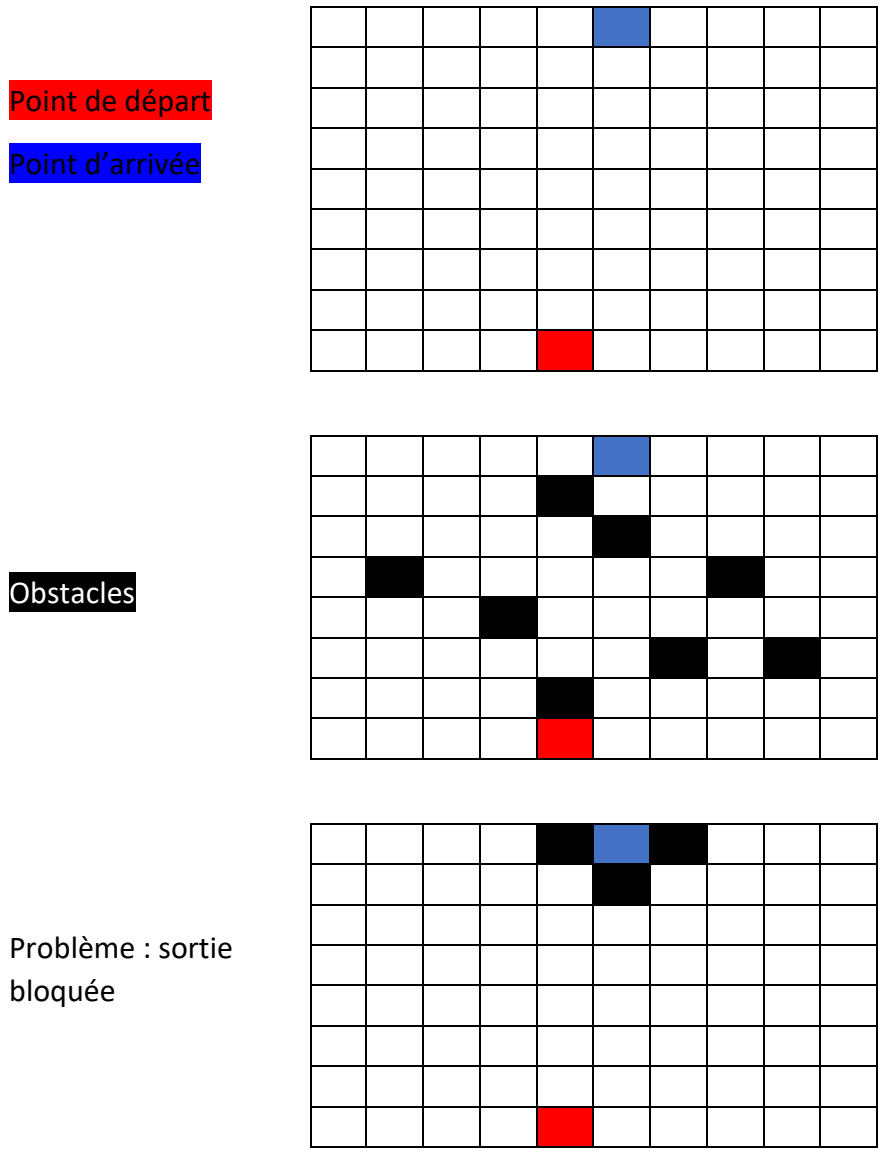

Figure 3 : exemple de la seconde méthode de génération de labyrinthe

La première méthode me semble être la meilleure car elle représente mieux l'objectif que je me suis fixé et permet d'approfondir les travaux sur lesquels je me suis basé pour étudier la méthode à sélectionner pour l'apprentissage, là où la méthode des cases posées a déjà été utilisée.

Evaluation de la performance

J'ai trouvé plusieurs méthodes pour la résolution de ce problème. La première méthode consiste à utiliser les processus décisionnels de Markov factorisés observable dans le désordre (PDMFOD) afin de former un Q-Learning parallèle. La deuxième est d'utiliser directement le Q-Learning. La troisième est l'apprentissage multi agent. Ces méthodes seront explicitées dans la rubrique « Explication en détail de chaque méthodes »

La première est très performante pour trouver rapidement un chemin efficace afin de trouver la sortie mais demande quand même de nombreuses itérations avant de trouver un résultat acceptable. Dans l'étude cette méthode est utilisée sur un labyrinthe avec 5 récompenses à aller chercher et utilise la deuxième méthode de génération d'obstacles.

La seconde est plus simple et demande plus d'itérations pour trouver un résultat exploitable. Dans l'étude que j'ai trouvée environ 400 itérations étaient nécessaires afin de trouver un résultat exploitable sur une labyrinthe très simple. Mon objectif étant de trouver le chemin sur un labyrinthe assez complexe, il peut alors y avoir des milliers d'itérations nécessaire.

La dernière méthode se base sur la communication entre plusieurs agents et demande un nombre d'itération très réduit pour trouver une convergence au niveau de la récompense moyenne obtenue, soit 250 sur l'étude.

Taxonomie de méthodes

Les trois méthodes se basent sur un principe commun qu'est le Q-Learning. Le Q-Learning est un des principes fondamentaux de l'apprentissage par renforcement. Cependant la méthode PDMFOD diffère de la première en poussant plus loin ce principe. La dernière méthode se base aussi sur le Q-Learning mais en ajoutant une mémoire sur les espaces où il y a le plus d'ambiguïté. Sur ces espaces ambigus, il y aura une mise à jour plus régulière des valeurs de gradient.

Il est donc important dans mon projet de comprendre le système de Q-Learning car il est la base de toutes les techniques utilisées dans la résolution de labyrinthe tout en étant plus ou moins poussé.

Explication en détail de chaque méthode

Tout d'abord la méthode du Q-Learning est une méthode qui se base sur des transferts d'information entre l'agent (le robot) et son environnement. En effet le principe est d'affecter des récompenses quand l'agent prends une bonne décision et des malus quand il en prend de mauvaises. L'agent a un panel d'action possible A (par exemple les déplacements qui lui sont permit d'effectuer) et une récompense associée à chaque action R (par exemple une grosse récompense si on se rapproche de l'objectif et un malus si on s'en éloigne). Le concept de bonne ou de mauvaises décisions est défini par la politique que le robot suit. La politique est la détermination du choix des actions. La politique n'est pas fixe, en effet c'est l'évolution dans le temps de la politique qui permet l'apprentissage.

On associe à des formules mathématiques chacun de ces termes. On note S l'état de l'environnement, A l'agent et R les récompenses. Chacun est un ensemble qui dépend du temps. La politique est une fonction qui permet de définir l'action choisi au temps t en fonction de l'état actuel : $\pi(S_t) = a_t$. Enfin la méthode Q-Learning est une fonction calculant l'espérance des gains futurs actualisés $Q_\pi(s_t, a_t) = E_\pi[\sum_{i \geq t} \lambda^{i-t} r_i | s_t, a_t]$. Lambda est le taux d'actualisation c'est-à-dire si les valeurs des récompenses changent vite ou non.

Dans le cadre d'un labyrinthe cela revient à associer à chaque case dans les 4 directions une récompense. Ces récompenses sont mises à jour en prenant en compte leurs état précédant et le bénéfice qu'elles ont apportés dans la simulation présente. Dans un premier temps l'agent explore au hasard la carte afin d'arriver à la sortie. Ensuite, il faut répéter le test plusieurs centaines de fois afin d'obtenir un résultat.

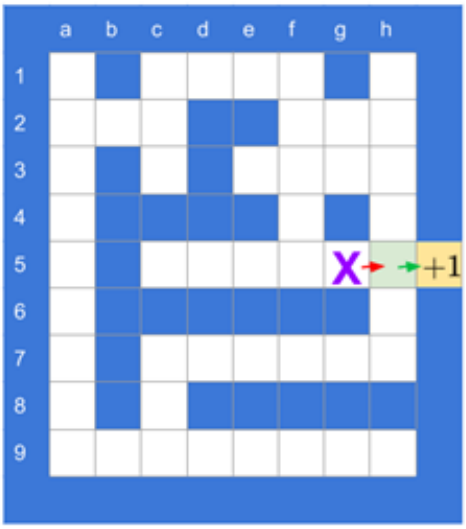

Figure 4 : schéma du fonctionnement du Q-Learning[3]

[3] Publicis Sapient, 2019, Johan Jublanc, Partie 1 : Un algorithme de base : le Q-Learning

La PDMFOD (Processus Décisionnels de Markov factorisés observables dans le désordre) est une méthode utilisant un vecteur d'état. Dans cette méthode, l'agent associe à la case sur laquelle il se situe à un instant t des sous états par rapports au reste des cases du labyrinthe. C'est une méthode qui se base sur la méthode du Q-Learning en l'approfondissant énormément. On part toujours du principe que l'agent connait la totalité du labyrinthe et peut donc connaitre l'emplacement de tous les murs et de la case de sortie. A Chaque états (donc chaque case) est donc associé un vecteur d'états prenant en compte les récompenses (la case de sortie) et les mallus (les murs) appelés sous-états. Cette méthode dispose de contraintes : en effet chaque case est indépendante de ses cases voisine car le calcul du vecteur ne dépend pas de la case précédente. Il faut donc ajouter en plus un « outil pour apparier les sous-états».

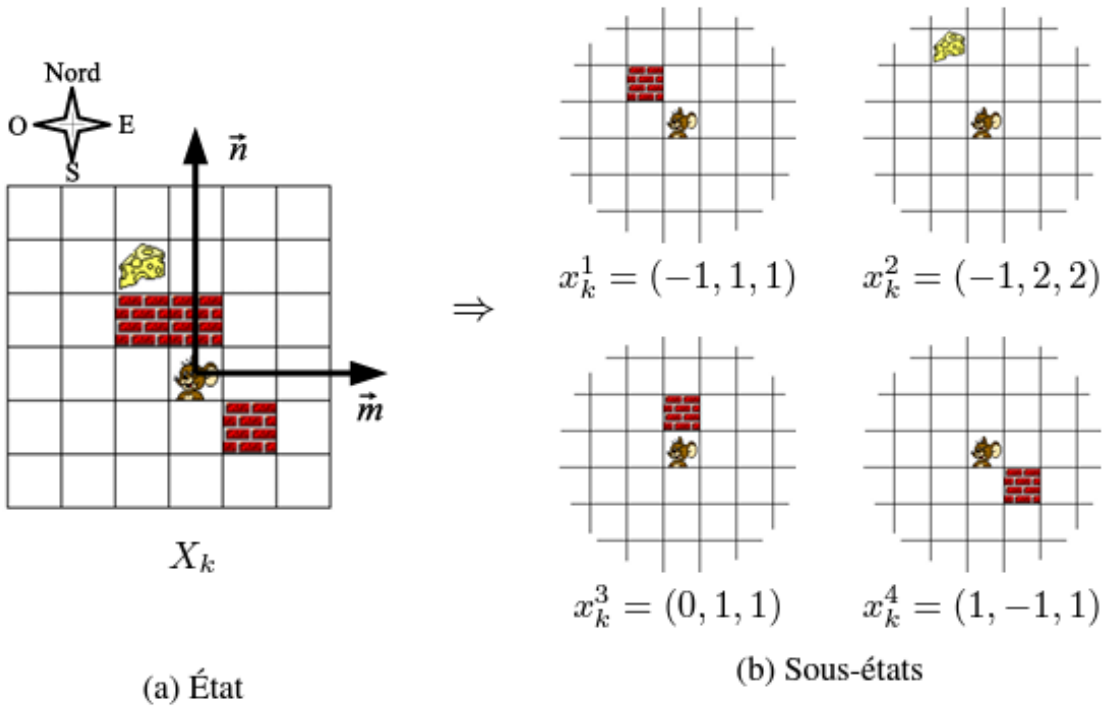

Figure 5 : schéma représentant les sous-états d'un système[4]

Il faut alors attribuer la récompense à la case en trouvant les sous-états qui permettent de trouver un score de récompense le plus élevé possible. Ensuite le calcul des gradients de récompense est basé sur le Q-Learning à la différence qu'il est possible que l'agent choisisse de se déplacer contre un mur dans quel cas celui-ci ne se déplace pas et sa case précédente

- [4] Guillaume J. Laurent, Emmanuel Piat. Apprentissage par renforcement dans le cadre des processus décisionnels de Markov factorisés observables dans le désordre. Etude expérimentale du Q-Learning parallèle appliqué aux problèmes du labyrinthe et du New York Driving. Revue des Sciences et Technologies de l'Information

est donc sa case actuelle. Dans cette méthode la sortie recueillie est le gain moyen final que l'agent aura accumulé à chaque déplacement.

La dernière méthode est une méthode fonctionnant à l'aide de mémorisation. Il faut tout d'abord débuter par un Q-Learning classique, et au bout de plusieurs itérations déterminer les cases qui sont les plus ambigus c'est-à-dire où deux gradients ou plus sont très proche. L'algorithme va alors enregistrer l'état de ces cases et tester à partir de celles-ci les chemins afin de clarifier les gradients. Il est possible d'affubler plusieurs agents sur le labyrinthe pour déterminer ces nouveaux gradient et d'ensuite faire une moyenne avec les gradients précédemment trouvés.

Choix de la méthode

Pour le projet il semble correct de s'orienter vers la méthode du Q-Learning classique car elle est plus simple à programmer, à s'approprier et répond à toutes les exigences fixées qui sont de trouver le chemin le plus efficace sans prendre en compte le temps de calcul. De plus avec cette méthode il peut être intéressant de regarder si lorsque l'on change d'emplacement le point de départ, l'algorithme se retrouve sans trop de difficulté. Bien que celui-ci demande un grand nombre d'itération sur des labyrinthe qui commence à devenir complexe, la longueur du temps de calcul ne m'importe pas et n'est pas un de mes critères. Cette méthode peut aussi être représentées facilement sur l'interface de 'utilisateurs là où les autres méthodes plus complexes le seront moins.

En outre, les autres méthodes sont trop complexes pour un simple labyrinthe surtout qu'elles ont plus de chance de se coincer dans des espaces restreints comme par exemple celle du PDMFOD. Cette méthode est plus utile dans des espaces plus ouverts comme la seconde génération de labyrinthe que j'ai énoncé.

Enfin la dernière méthode demande de savoir coder plusieurs agents en même temps et d'ajouter un moyen de mémoire, qui permet une efficacité accrue mais une complexité exponentielle.

En conclusion, il est préférable de prendre le Q-Learning simple car il répond le mieux à mes exigences énoncées dans l'introduction.

Bibliographie

- Publicis sapient, 2019, Johan Jublanc, Reinforcement learning, partie 1 : introduction, consulté le 18/10/2022

https://blog.engineering.publicissapient.fr/2020/02/12/reinforcement-learning-introduction/

- Guillaume J. Laurent, Emmanuel Piat. Apprentissage par renforcement dans le cadre des processus décisionnels de Markov factorisés observables dans le désordre. Etude expérimentale du Q-Learning parallèle appliqué aux problèmes du labyrinthe et du New York Driving. Revue des Sciences et Technologies de l'Information- Série RIA: Revue d'Intelligence Artificielle, Lavoisier, 2006, 20, pp.275309. Hal-00342330

https://hal.archives-ouvertes.fr/hal-00342330

- Shirley HOET & Nicolas SABOURET. Utiliser des protocoles d'interaction de la mémoire pour l'apprentissage par renforcement d'actes de communication. Laboratoire d'informatique de Paris 6.

chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://www.info.fundp.ac.be/mfi09/wp-content/uploads/2009/04/short-HoetSabouret.pdf

- Justin FU, John D. Co-Reyes & Serguey LEVINE, EX² : Exploration with Exemplar Models for Deep Reinforcement Learning. University of California Berkeley.

chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://proceedings.neurips.cc/paper/2017/file/1baff70e2669e8376347efd3a874a341-Paper.pdf