Rapport Final

Le génie pour l'industrie

25/10/2022

Léo Laval-SYS819

Table des matières

Introduction

Dans le cadre de ma formation d'ingénieur robotique, l'apprentissage profond prends une place protubérante. Avec l'essor de la robotique, il est primordial de comprendre et d'appliquer les bases de ce domaine et c'est pourquoi j'ai choisi ce cours. Dans le cadre de ce cours nous avons un projet à rendre : j'ai choisi de m'intéresser à l'apprentissage par renforcement sur lequel j'avais déjà entendu parler. J'ai donc choisi d'étudier le déplacement d'un agent dans un labyrinthe afin de trouver le chemin le plus efficace pour en sortir. Le labyrinthe possède plusieurs chemins qui mène à la sortie ce qui permet de valoriser l'apprentissage du robot. L'objectif est de trouver le chemin le plus proche de la perfection dans un temps raisonnable sans recourir à la force brute qui prends énormément de temps. Ce projet s'inscrit directement dans ma formation car je peux choisir l'itinéraire d'un robot dans une pièce avec des obstacles en réduisant son temps de déplacement.

De plus, j'ai beaucoup entendu parler de l'apprentissage par renforcement dans ma formation sans jamais avoir pu l'appliquer. Il était donc nécessaire que je porte mon attention dessus afin de mettre la lumière sur ce sujet qui m'intriguait.

Rappel de la tâche

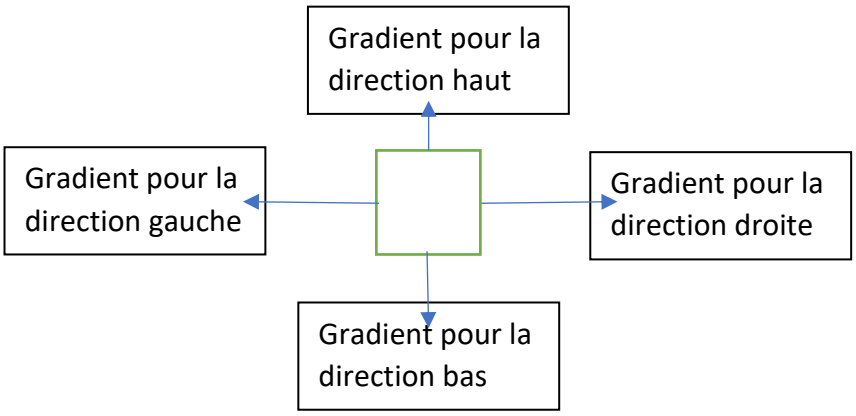
Le labyrinthe se compose de plusieurs types de case. Tout d'abord la première qui représente les murs infranchissables que le robot doit éviter. Ensuite, il y a la case de départ du robot et la case de sortie. Enfin toutes les autres cases sont des cases de déplacement. L'entrée du programme est donc 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.

Résumé de la revue de littérature

La revue de littérature se base sur 3 méthodes intrinsèquement reliées et toute basés sur une méthode : le Q-Learning. Le Q-Learning est une méthode d'apprentissage profond qui se base sur un gradient d'état qui évolue en fonction d'une attribution de récompense. Les deux autres méthodes sont des dérivées de cette méthode.

Le Q-Learning

Le Q-Learning est un échange d'information entre un agent et son environnement. Dans le cadre de notre projet de labyrinthe cela revient à associer à chaque case un gradient pour chaque direction qu'il est possible de prendre.



Au début de la simulation, l'agent choisi aléatoirement des directions jusqu'à attendre l'arrivée. A l'arrivée, il y a une récompense qui permet de mettre à jour le gradient de la case précédente qui permet de mettre à jour celle d'avant et ainsi de suite. Au fur et à mesure le gradient s'affine afin de dessiner le chemin le plus optimal. Chaque gradient est initialisé à 0 c'est-à-dire qu'il n'apporte aucune récompense. De temps à autre, le système choisi de ne pas suivre le gradient le plus optimal et de prendre une direction aléatoire parmi celle permise afin de découvrir de nouveaux chemins. Le choix aléatoire est défini par la variable epsilon qui décroit au fil du temps pour tendre vers 0 : au début il va y avoir beaucoup de choix aléatoire et à la fin il y en aura très peu. La mise à jour des gradients est effectuée grâce au calcul de l'espérance des gains futurs avec la formule : $Q_\pi(s_t, a_t) = E_\pi[\sum_{i \geq t} \lambda^{i-t} r_i | s_t, a_t]$. Lambda est le taux d'actualisation c'est-à-dire si les valeurs des récompenses changent vite ou non.

La PDMFOD (Processus Décisionnels de Markov factorisés observables dans le désordre)

La PDMFOD est une méthode utilisant un vecteur d'état. En connaissant le labyrinthe à l'avance, il est possible de situer les murs et la récompense, appelés sous-états, et de leur attribuer un vecteur de bonus ou de malus en partant de la position actuelle. Dans cette méthode, le choix de la direction ne dépend pas de la case précédente.

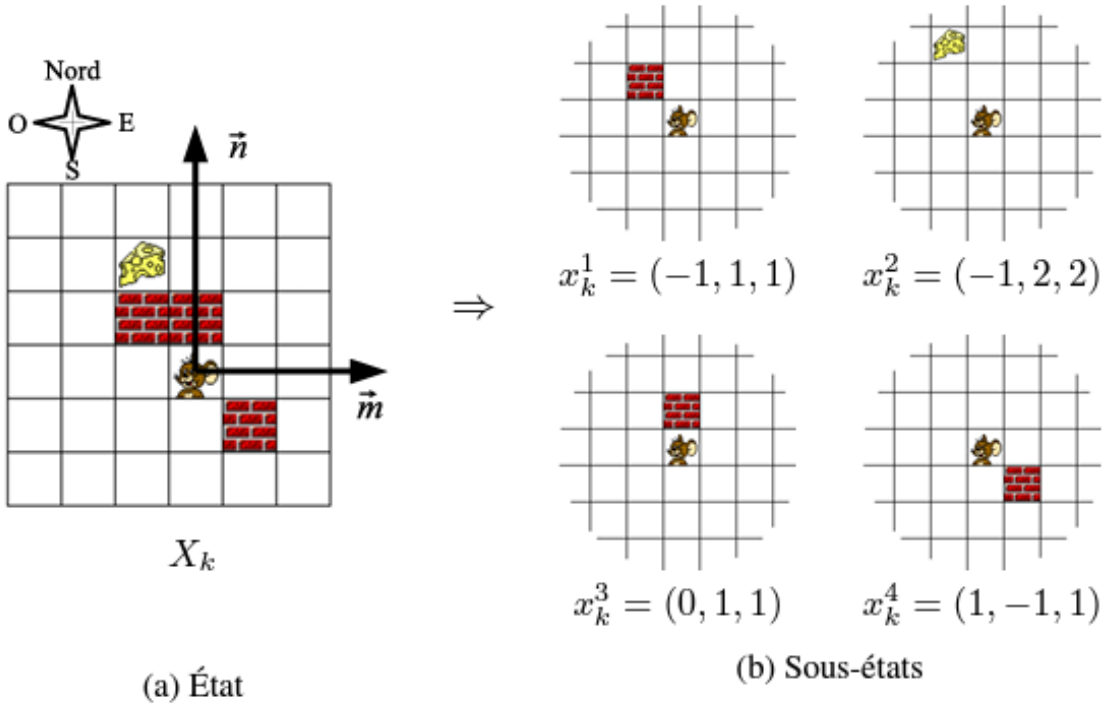


Figure 1:schéma représentant les sous états d'un système[1]

Méthode multiples agents

Cette méthode débute comme un Q-Learning classique. Au bout d'un certain nombre d'itérations, on cherche les cases les plus ambiguës, c'est-à-dire les cases où les gradients sont proches à delta près. Il faut alors enregistrer l'état de ces cases et commencer le traitement à partir de ces cases ci en disposant plusieurs agents qui vont effectuer le trajet afin d'éclaircir les ambiguïtés. Cette méthode repose sur le threading du programme en plusieurs tâches. Cette méthode permet d'obtenir une meilleure appréciation du chemin optimal sur de très grands labyrinthe mais est très gourmande en therme de capacité de calculs.

[1]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

Implémentation

But du projet

Le but de ce projet est de déterminer le chemin le plus efficace pour rejoindre la sortie du labyrinthe. Il s'inscrit dans une optique de déplacement de robot dans un espace connu dans lequel il se trouve des obstacles gênant la mobilité de celui-ci. Le robot doit être capable de calculer e meilleur chemin possible en un temps correct afin de ne pas rester immobile à son point de départ. Le robot ne débute son déplacement qu'une fois que le chemin lui parait optimal. De plus le robot à surement une puissance de calcul limité à cause de ses composants, il faut donc éviter la force brute pour choisir le chemin. Deux cas sont principalement étudiés : le robot doit atteindre la sortie le plus vite possible, le chemin optimal n'est pas nécessaire ou le robot doit emprunter le chemin optimal, le temps de calcul peut être un peu plus long.

Une fois le but du projet établi, il faut implémenter un environnement simulant le robot et le labyrinthe. Pour cela, j'ai récupéré le code d'un générateur de labyrinthe parfait[2], c'est-à-dire d'un labyrinthe qui ne possède qu'un seul chemin de sortie, duquel j'ai enlevé un certain pourcentage de murs. Ainsi, plusieurs chemins sont créés et l'environnement est bien simulé comme voulu.

Figure 2 : exemple de labyrinthe généré

Après avoir créé l'environnement il faut calculer la trajectoire du robot à l'aide de l'apprentissage par renforcement. Pour cela on crée une fonction de déroulement de chaque tour où l'on se déplace puis où l'on met à jour les gradients jusqu'à que la position soit celle d'arrivée.

Contributions

Sur ce projet, il est intéressant de regarder l'effet de chaque paramètre sur le résultat obtenu. Pour cela nous allons réaliser plusieurs tests en changeant à chaque fois un seul paramètre. Nous pourrons alors regarder la différence sur la vitesse de convergence et la précision du nombre de mouvement minimum à réaliser pour sortir du labyrinthe. La formule de calcul du gradient est la suivante : nouveau gradient=Qt + alpha*(lambda* Qt+1 -Qt) avec Qt le gradient de la case actuelle pour la direction choisie, Qt+1 le gradient le plus haut de la case choisie, alpha le poids de la case suivante dans le calcul et lambda le taux de rafraichissement. Nous allons analyser le nombre d'itération répété, la taille du labyrinthe, le taux de

[2] The Startup, 15/09/2020, Orestis Zekai , Medium : https://medium.com/swlh/fun-with-python-1-maze-generator-931639b4fb7e

rafraichissement, la probabilité d'avoir un choix aléatoire et le poids de la case suivante dans le calcul du gradient.

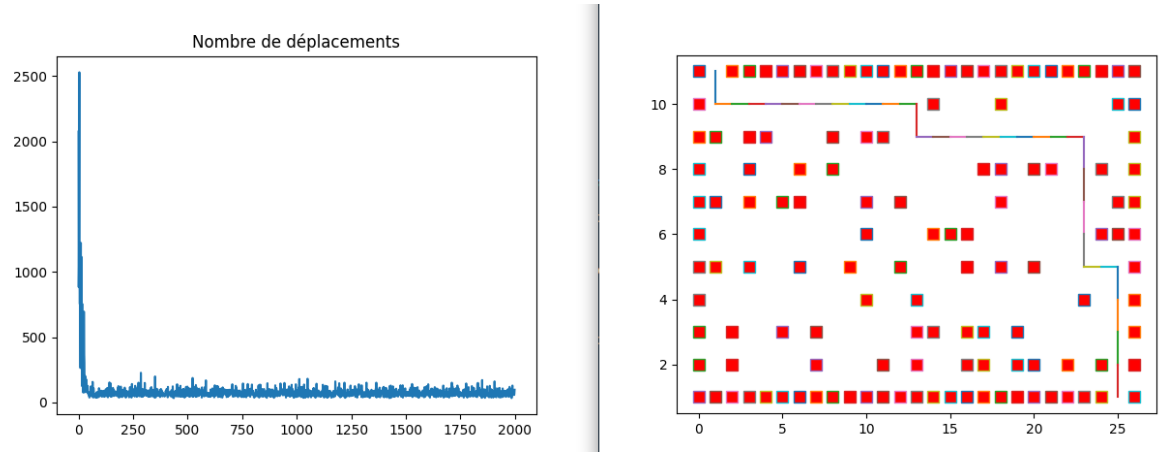
Expériences

Méthode utilisée

Pour le projet j'ai choisi d'utiliser le Q-Learning simple. Pour cela on associe à chaque cellule 4 valeurs de gradient. Chaque expérience est répétée plusieurs fois et le résultat expliqué est le condensé de ce qui ressort le plus pour chaque paramètre et est illustré avec un exemple représentatif de toutes ces expériences.

Expérience 1 : expérience test

Cette expérience est l'expérience test avec laquelle on pourra comparer tous les résultats. Elle se déroule dans un labyrinthe de taille moyenne de 27 par 11 sur 2000 itérations. Nous avons une réduction d'epsilon de 0.95, alpha et lambda sont à 0.5.

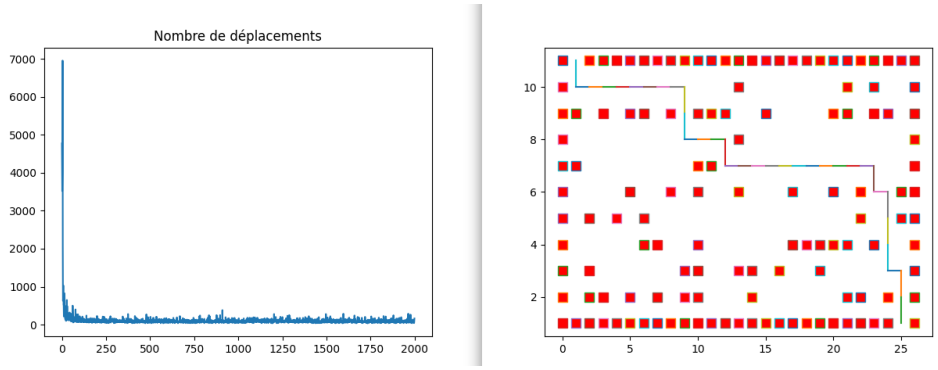


On remarque dans l'expérience que les valeurs convergent, et on retrouve le chemin le plus efficace.

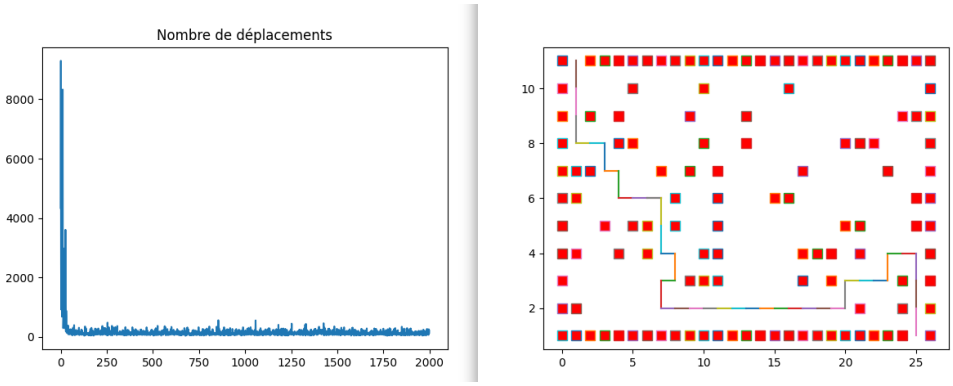
Expérience 2 : réduction de la probabilité de choix aléatoire

Dans cette expérience on cherche à tester l'effet de la réduction plus ou moins rapide de la probabilité de choisir le chemin optimal ou de choisir un chemin aléatoire. Epsilon se réduit comme suit : epsilon = epsilon * pourcentage de réduction

Avec un pourcentage de réduction d'Epsilon de 99% :



Avec un pourcentage de réduction d'Epsilon de 50% :



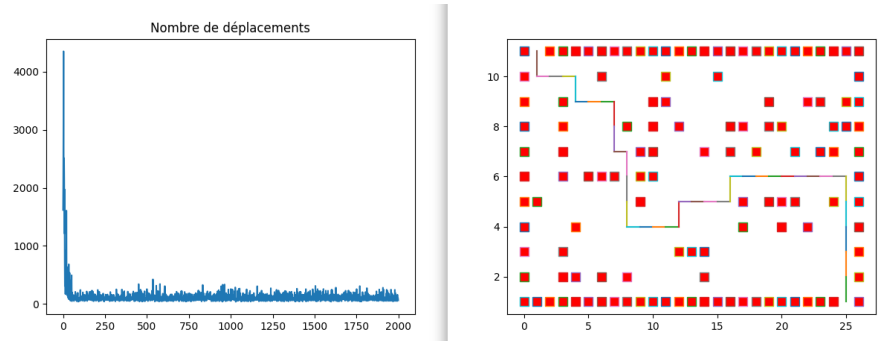
On remarque dans cette simulation que lorsque la quantité d'exploration quand est trop basse, la convergence du nombre de déplacement est plus lente que dans l'expérience test. On remarque par contre que le chemin est le plus optimal.

Dans l'autre cas, quand la réduction est trop forte, le chemin optimal n'est pas trouvé ce qui est normal car le programme va vouloir se concentrer sur son premier chemin et explorer le moins possible. On remarque par contre que le nombre de déplacement converge plus vite et c'est explicable car le chemin se modifie moins au fil du temps.

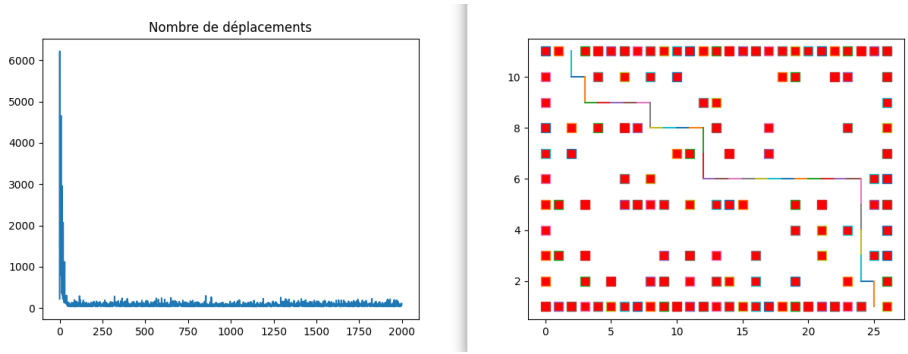
Expérience 3 : taux de rafraichissement

Cette expérience teste le taux de rafraichissement des gradients

Pour un taux haut avec lambda qui vaut 0.9



Pour un taux bas avec lambda qui vaut 0.1

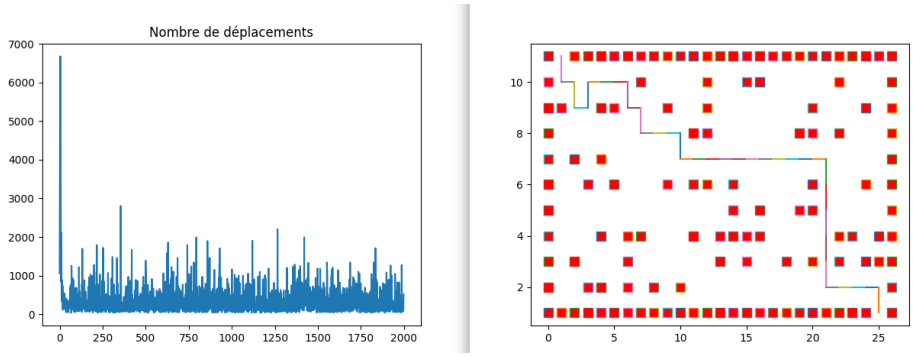


On remarque que le taux de rafraichissement influe sur le nombre de déplacement : en effet il converge moins bien avec un taux trop élevé. Cependant cette variable n'influe que très peu le résultat.

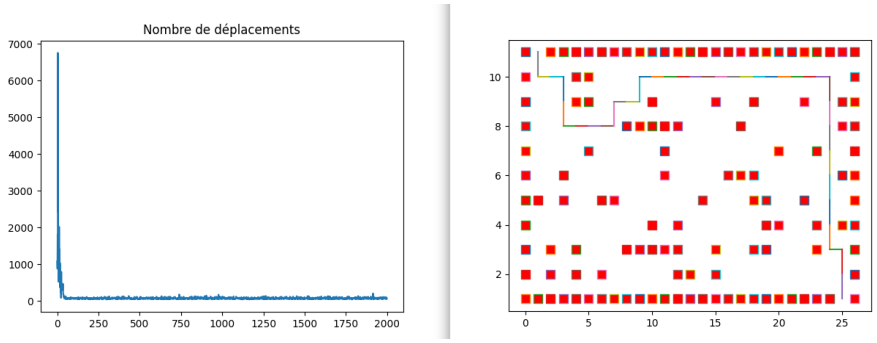
Expérience 4 : poids de la case suivante dans le calcul du gradient

Dans cette expérience on teste le poids de la case choisie dans la mise à jour du gradient

Avec un poids élevé où alpha vaut 0.9



Avec un poids faible où alpha vaut 0.1

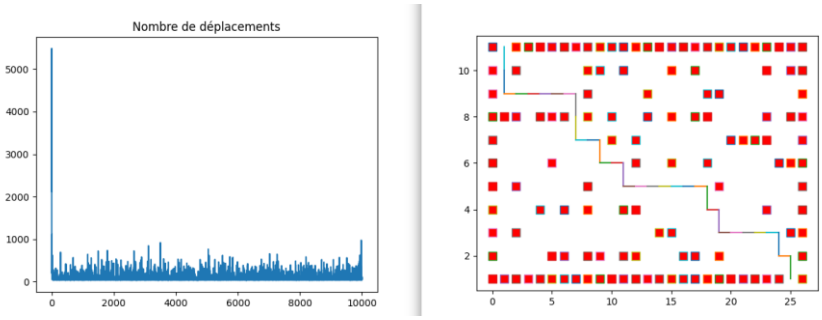


On remarque distinctement que cette variable affecte énormément la convergence du nombre de déplacement. Si le poids est faible le nombre de déplacement est très stable mais ne converge pas très vite. Dans le cas contraire le nombre de déplacement varie beaucoup mais converge plus vite vers une valeur « acceptable » sans avoir à revenir pleins de fois sur ses pas.

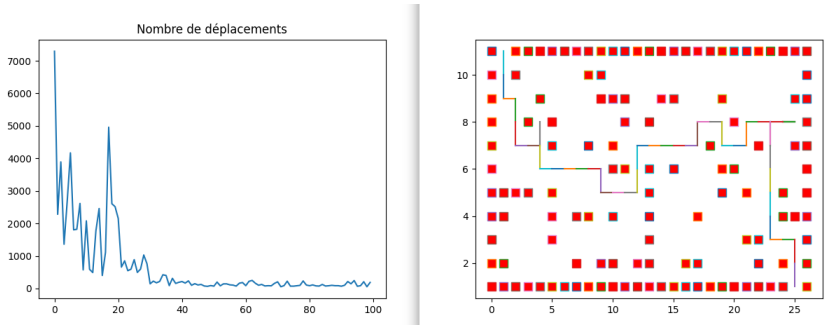
Expérience 5 : Nombre d'itération

On cherche l'effet du nombre d'itération sur le chemin optimal et sur le temps de calcul.

Avec un nombre d'itération de 10000 :



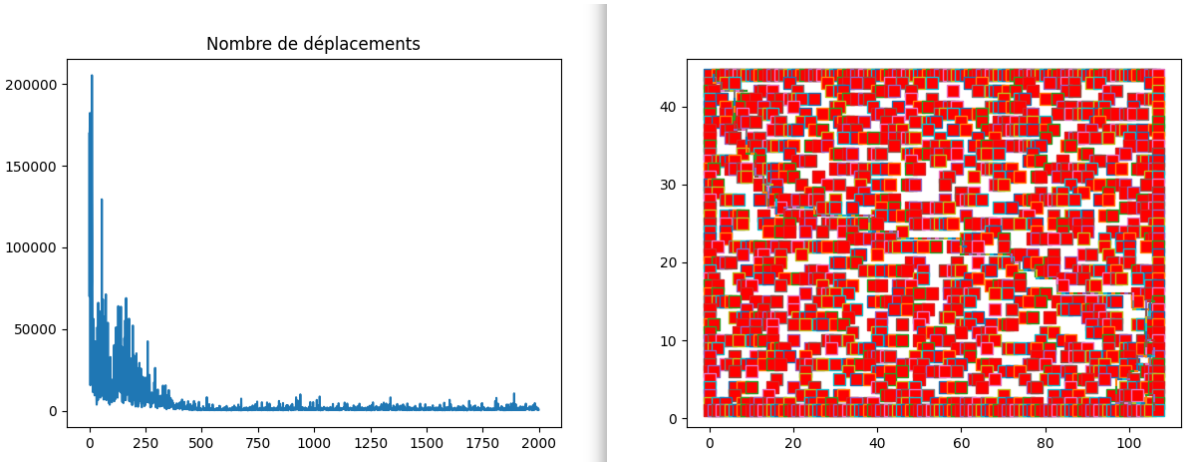
Avec un nombre d'itération de 100 :



Lorsque l'on augmente le nombre d'itérations, on garantit le meilleur chemin mais au coût d'un temps de calcul énorme. Un petit nombre d'itération permet d'avoir un chemin efficace mais pas optimal.

Expérience 6 : Taille du labyrinthe

On cherche l'effet de la taille du labyrinthe sur le programme. Ici, le labyrinthe est 4 fois plus grand.



Sur cette expérience le temps de calcul est beaucoup plus long et est d'environ 10 minutes. Sur des labyrinthes compliqués il ne faut pas compter sur uniquement le nombre d'itération pour trouver le bon chemin car il faudrait trop de temps de calcul. Il faut donc compenser en jouant avec les variables préalablement étudiées afin d'obtenir un résultat satisfaisant dans un temps correct.

Conclusion

Dans le cadre plus concret de l'application à un robot ces expériences nous permettent de définir des paramètre en fonction du besoin : S'il est nécessaire pour le robot d'avoir un itinéraire les plus rapidement possible il faut privilégier un petit nombre d'itérations, afin d'avoir un temps de calcul réduit, un poids haut afin, un taux élevé une probabilité de choix aléatoire basse afin de se concentrer sur le premier chemin trouvé et de l'optimiser même si ce n'est pas le chemin optimal absolu. On pourrait retrouver ce cas dans un terrain accidenté où le robot doit prodiguer les premiers soins à des personnes blessées. D'autre part, s'il est nécessaire de trouver le chemin le plus court il faudra alors privilégier un grand nombre d'itération, un poids bas, un taux moyen et une probabilité de choix aléatoire haute. Cette situation pourrait se présenter sous la forme d'un manque de batterie de la part du robot qui doit retourner à sa borne de rechargement de la manière la plus efficace mais qui n'est pas dépêché par le temps.

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

- The Startup, 15/09/2020, Orestis Zekai , Medium : https://medium.com/swlh/fun-with-python-1-maze-generator-931639b4fb7e