



Université de la Manouba
École Nationale des Sciences de l'Informatique



RAPPORT DU PROJET DE CONCEPTION ET DE DÉVELOPPEMENT

Sujet :
Une Simulation Robotique pour
l'Expérimentation des Méthodes
d'Apprentissage par Renforcement

Auteurs :

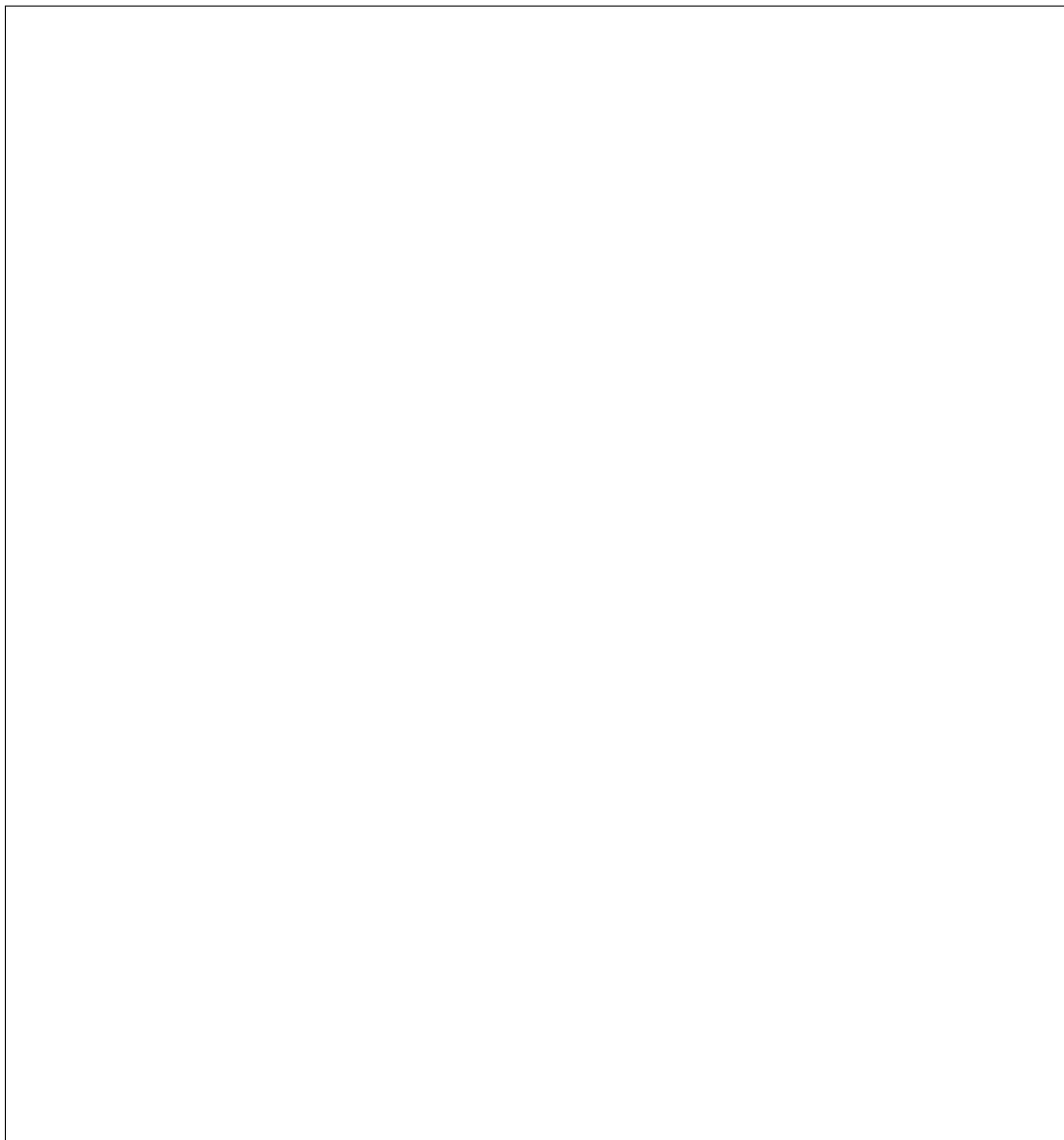
Arij SOULA Hajer MHALLA Moez RAMDHAN

Encadrants :

Wided LEJOUAD CHAARI

Année Universitaire : 2020/2021

Appréciations et signature de l'encadrant

A large, empty rectangular box with a thin black border, intended for the evaluator's appreciations and signature.

Remerciements

Nous tenons à remercier toutes les personnes qui ont contribué au succès de notre projet de conception et de développement et qui nous ont aidés lors de la rédaction de ce rapport.

Nous voudrions principalement remercier notre encadrante Mme Wided Lejouad Chaari pour sa patience, sa disponibilité et surtout ses judicieux conseils, qui ont contribué à alimenter notre réflexion.

Résumé — Ce présent rapport décrit une simulation robotique dans le but d'expérimenter des méthodes d'apprentissage par renforcement, en vue d'une étude comparative de celles-ci dans le problème du labyrinthe, ainsi qu'une proposition d'amélioration de la stratégie de navigation. Deux algorithmes sont présentés et interprétés expérimentalement. Pour différents scénarios, les résultats et les analyses de simulation montrent qu'une grande différence de performance entre les différents algorithmes peut être observée. Nous nous penchons finalement vers l'avenir et illustrons certains défis importants qui peuvent améliorer l'état de l'art sur l'apprentissage par renforcement appliqué au problème de navigation robotique.

Mots clés : expérimentation, apprentissage automatique, apprentissage par renforcement, navigation autonome, agent, simulation robotique.

abstract— This paper describes a robotic simulation for experimenting reinforcement learning methods for a comparative study in the maze problem, as well as a proposal for improving the navigation strategy. Two algorithms are presented and experimentally interpreted. For different scenarios, results and simulation analyses show that a large difference in performance between the two algorithms can be observed. We finally look forward and illustrate some important challenges that can improve the state of the art on reinforcement learning applied to the robotic navigation problem.

Key words : Algorithms, experimentation , machine learning ,reinforcement learning , Autonomous navigation , agent ,robotics simulation

تلخيص:

يصف هذا التقرير محاكاة روبوتية لاختبار أساليب التعلم المعزز بهدف إجراء دراسة مقارنة بين هذه الخوارزميات في مشكلة المتاهة واقتراح تحسين استراتيجية التنقل الذاتية. يتم عرض اثنين من هذه الخوارزميات وتفسيرها تجريبيا وتظهر اختلاف كبير في الأداء بين الخوارزميات. نحن في النهاية نسلط الضوء على النتائج وتحليل المحاكاة في سيناريوهات مختلفة على بعض التحديات المهمة التي من شأنها أن تحسن من طريقة التعلم المعزز.

الكلمات الدالة: خوارزمي، الخوارزميات، التجريب، التعلم الآلي، التعلم المعزز، التنقل المستقل، الوكيل، محاكاة الروبوتات.

Table des matières

1	Étude Préalable	4
1.1	Introduction	4
1.2	L'intelligence Artificielle	4
1.3	Le concept d'Apprentissage Automatique	5
1.4	Les types d'apprentissage	5
1.5	Les Intérêts de l'Apprentissage par renforcement	7
1.6	Description de la problématique	8
1.7	Conclusion	8
2	L'apprentissage par renforcement pour la navigation robotique	9
2.1	Introduction	9
2.2	Définition d'un agent	9
2.3	Les systèmes multi-agents	10
2.4	Les concepts relatifs à l'agent apprenant	11
2.4.1	Processus de Décision Markoviens (MDP)	12
2.4.2	Exploration et Exploitation	13
2.5	Les Algorithmes d'apprentissage par renforcement	14
2.6	Conclusion	15
3	Simulation Robotique	17
3.1	Introduction	17
3.2	Description de l'environnement de navigation	17
3.3	Modélisation du problème de navigation	18
3.4	Environnement de simulation	19
3.4.1	Motivation de choix de NetLogo	20
3.4.2	L'interface graphique et les concepts d'agents dans Netlogo	20
3.5	Modélisation du labyrinthe avec NetLogo	21
3.6	Algorithmes d'apprentissage par renforcement implémentés	25
3.6.1	Q-Learning	25
3.6.2	SARSA	31
3.7	Conclusion	33

4	Résultats des Expérimentations	34
4.1	Introduction	34
4.2	Paramétrage des algorithmes	34
4.3	Résultats expérimentaux, analyse et interprétations	36
4.3.1	Q-Learning	37
4.3.2	SARSA	45
4.4	Comparaison entre Q-Learning et SARSA	46
4.5	Conclusion	48
	Netographie	53

Table des figures

2.1	Architecture d'un agent	10
2.2	Apprentissage par renforcement [?]	11
2.3	Processus de Décision Markoviens	12
2.4	La politique (ou Stratégie)	13
2.5	La subdivision de l'apprentissage par renforcement	14
3.1	L'interface utilisateur de NetLogo	21
3.2	Les fonctions sur NetLogo	22
3.3	Spacing=10	23
3.4	Spacing=20	23
3.5	Une scène de simulation	24
3.6	Processus Q-Learning	26
3.7	Un exemple de mise à jour de la Q-table	27
3.8	Algorithme de ϵ -Greedy	28
3.9	Q-Greedy	29
3.10	Q-Epsilon-Greedy	30
3.11	Q-Epsilon-Decay	31
3.12	: Différence entre les processus SARSA et Q-Learning	32
3.13	L'algorithme de SARSA	33
4.1	Courbe des valeurs du paramètre discount factor γ	35
4.2	Résultat Q-Greedy sur un labyrinthe simple	38
4.3	Résultat Q-Greedy sur un labyrinthe complexe	38
4.4	Résultat Q-Greedy sur un labyrinthe simple avec changement de but	39
4.5	Q-Epsilon-Greedy après amélioration	40
4.6	Comparaison de l'algorithme avec exploration par pas (en rouge) et l'exploration par épisode (en vert)	40
4.7	Résultat d'expérimentation de Q-epsilon-greedy sur un labyrinthe simple statique	41
4.8	Résultat d'expérimentation de Q-epsilon-Greedy sur un labyrinthe complexe statique	42

4.9	Résultat d'expérimentation de Q-epsilon-Greedy sur un labyrinthe simple statique	42
4.10	Comparaison Q-Greedy (en rouge) et Q-epsilon-decay (en vert) sur un labyrinthe simple	43
4.11	Résultat de Q-epsilon-decay sur un labyrinthe complexe	44
4.12	Résultat Q-epsilon-decay sur un labyrinthe complexe avec epsilon déjà annulée	44
4.13	Résultat de Q-epsilon-decay sur un labyrinthe complexe avec epsilon non encore annulée	45
4.14	Résultat de SARSA avec facteur epsilon constant	45
4.15	Résultat de SARSA sur un labyrinthe complexe	46
4.16	Comparaison entre Q-epsilon-decay et SARSA en nombre de pas	46

Liste des tableaux

3.1	Récompenses et Pénalités	19
4.1	Les hyper-paramètres obtenus	36
4.2	Comparaison entre Q-learning et SARSA selon les critères NP/NE	47

Introduction

Au début du 20ème siècle, il n'y avait pas de distinction entre l'Intelligence Artificielle (IA) et l'informatique. C'est à partir de l'année 1950 que l'IA a évolué comme une discipline de l'informatique, en faisant référence à la simulation de l'intelligence humaine dans les machines programmées afin de penser comme des humains et imiter leurs actions et donc faire tout ce qu'un cerveau humain pourra faire .

Cette innovation peut être également appliquée à toute machine programmée présentant des traits associés à un esprit humain tel que l'apprentissage et la résolution des problèmes. Ainsi, l'intelligence artificielle vise l'apprentissage, le raisonnement et la perception.

A quel point l'IA est-elle importante ?

Ces dernières décennies, la technologie de l'intelligence artificielle est devenue plus importante voire essentielle dans notre monde. En effet, c'est un outil et une technique ayant pour but de rendre ce monde meilleur. Vous pouvez simplement regarder autour de vous et vous allez vous rendre compte que la plupart de nos tâches sont facilitées par l'intelligence artificielle.

Par exemple, les réseaux sociaux sont influencés par l'IA, tel que Facebook qui trie les publications, puis les filtre pour mettre en avant celles jugées les plus importantes. Ce site utilise l'IA pour analyser le contenu des publications.

De plus, elle permet de mettre en œuvre des capacités humaines comme la compréhension, le raisonnement, la planification, la communication et la perception dans des applications de plus en plus efficaces et à faible coût. Ce qui permet de faciliter notre vie et réduire l'effort humain.

Quels sont les domaines de l'IA ?

Plusieurs domaines ont été influencés par l'IA cherchant à utiliser l'intelligence artificielle à leur avantage. De plus, elle peut être appliquée à tous les secteurs pour offrir de nouvelles possibilités et des gains d'efficacité.

Prenons comme exemple le secteur de la santé qui est l'un des secteurs les plus consommateurs de l'intelligence artificielle où plusieurs applications basées sur l'IA permettent de diagnostiquer les maladies, par exemple le cancer, même parfois mieux que des spécialistes. L'IA est présente aussi dans le domaine du transport, tel que les fameux véhicules autonomes, ni volant ni pédale, comme les taxis autonomes. On cite aussi aux USA, certains

avions civiles sont équipés de l'IA décollent et atterrissent avec des voyageurs.

Plusieurs constructeurs sont encore en concurrence pour mettre en œuvre cette réussite légendaire. Notons aussi que l'IA prend de plus en plus d'importance dans le domaine des jeux. Ses progrès sont nombreux et différent d'un jeu à un autre. C'est le cas pour les jeux d'Atari et aussi le jeu vidéo de tir à la première personne « Quake III Arena », ou « StarCraft II » le jeu de stratégie en temps réel qui adoptent des stratégies (i.e. algorithmes) pour gagner.

Encore un domaine important appliquant l'intelligence artificielle qui est l'industrie, c'est ce qu'on appelle l'IA Industrielle. Elle est plus concernée par l'application de cette technologie pour résoudre les problèmes industriels pour l'amélioration du rendement, la diminution des coûts, l'optimisation, l'analyse prédictive et la découverte d'informations. Citons particulièrement, le domaine de la robotique qui a un grand potentiel d'applications dans l'industrie des services et médicale. Souvent, les robots sont déployés pour effectuer des tâches qui pourraient être difficiles pour des êtres humains.

Aussi, le secteur de l'agriculture est, actuellement, l'un des secteurs les plus influencés par l'IA. Grâce aux capteurs plantés dans la terre et à des caméras ou encore des drones, on récolte beaucoup d'informations, qui seront ensuite envoyées et analysées par des algorithmes. Ces analyses vont permettre aux « agriculteurs intelligents » de prendre les décisions, par suite faciliter leur travail.

Alors comment s'est passée la transformation de la robotique au fil du temps ?

Dans tous les secteurs, particulièrement l'industrie, les robots doivent faire preuve de flexibilité, d'adaptabilité et, surtout d'autonomie. Il est clair donc que ces facteurs dépendent fortement de la robotique mobile.

En effet, un robot mobile est un robot capable de se déplacer dans un environnement. Il est caractérisé non seulement par son adaptation à la présence des changements dans l'espace d'exploitation mais aussi par sa navigation, planification et action indépendantes. Prenons à titre d'exemple le robot mobile Persévérance qui a été envoyé par la NASA vers la planète MARS le 30 juillet 2020, élaboré pour dégager les traces des anciens microbes qui grouillaient sur la planète il y a des milliards d'années.

En d'autres termes, la robotique mobile est généralement considérée comme un sous-domaine de la robotique et de l'ingénierie de l'information. Un robot mobile doit pouvoir se déplacer tout en évitant les obstacles. Par conséquent, le robot a la tâche de planifier sa trajectoire dans l'environnement en tenant compte des coûts et de la qualité du chemin choisi. Parmi les critères à minimiser, nous mentionnons la distance parcourue par le robot de la position initiale à la position finale, la trajectoire, l'énergie consommée, le temps mis pour atteindre la position finale, etc.

De ce fait, le problème passe de la planification de la trajectoire vers un problème d'optimisation avec un ensemble de contraintes.

Considérés comme des agents autonomes, les robots ont des capacités qui surpassent celles

des humains pour certaines activités.

Doté d'un algorithme d'apprentissage par renforcement, ces super-spécialistes peuvent explorer un environnement et apprendre comment agir afin d'obtenir la meilleure récompense.

Le présent travail est réalisé dans le contexte d'un Projet de Conception et de Développement (PCD) en deuxième année du cycle de formation d'ingénieur en informatique.

Notre objectif consiste à expérimenter plusieurs méthodes d'apprentissage par renforcement en vue d'une étude comparative de celles-ci.

Le présent rapport est articulé autour de quatre chapitres.

Le premier chapitre présente une étude préalable permettant d'introduire les approches intelligentes, de décrire les concepts clés du projet, la terminologie associée et quelques définitions.

Le second chapitre porte essentiellement sur l'apprentissage par renforcement et ses techniques.

Le troisième chapitre se focalise sur l'environnement de simulation robotique et sa mise en œuvre.

Finalement, le quatrième chapitre décrit l'expérimentation et ses résultats et établit une étude comparative des algorithmes développés.

Chapitre 1

Étude Préalable

1.1 Introduction

La future génération de robots a besoin d'un ensemble de compétences pour effectuer des tâches complexes de manière autonome. La navigation autonome est l'une des capacités fondamentales dont ces robots doivent être dotés. La navigation des robots fait l'objet de la recherche depuis des années. Cet état de l'art nécessite la perception de l'environnement, la planification du chemin, le déplacement entre les cibles et les réactions en temps réel aux événements inattendus.

Les premiers robots autonomes ont été utilisés par la NASA en 1990 dans l'exploration de l'espace. Et plus récemment la chine a réussi à envoyer le robot Zhurong sur la planète MARS. Aujourd'hui, les robots autonomes sont inclus dans de nombreux domaines. Les avantages de la robotique et des systèmes autonomes sont de plus en plus visibles. Leur croissance dans le domaine industriel et commerciale montre l'impact qu'ils auront au cours des prochaines années.

Parmi les exemples d'utilisation, on peut citer la fabrication, les transports (gestion du trafic aérien, les drones, voitures autonomes...), l'énergie (réseaux intelligents), l'agriculture, la médecine, la défense (manutention des matériaux dangereux, élimination des bombes...), et l'espace (inspection et réparation en orbite, l'exploitation minière...). Dans ce chapitre, nous proposons quelques définitions relatives à notre projet dont l'objectif est d'expérimenter certains algorithmes intelligents pour la navigation robotique.

1.2 L'intelligence Artificielle

L'Intelligence Artificielle (IA) est une jeune discipline d'une soixantaine d'années, qui regroupe des sciences, des théories et des techniques y compris la logique mathématique, les statistiques, les probabilités, la neurobiologie computationnelle et l'informatique. Elle repose sur le principe que l'intelligence humaine peut être définie de manière à ce

qu'une machine puisse facilement l'imiter et exécuter des tâches, des plus simples aux plus complexes.

1.3 Le concept d'Apprentissage Automatique

L'apprentissage est l'un des éléments fondamentaux des solutions d'intelligence artificielle (IA). D'un point de vue conceptuel, l'apprentissage est un processus qui améliore la connaissance d'un programme d'IA en faisant des observations sur son environnement.

En termes de technologies avancées, l'un des domaines les plus demandés est l'apprentissage automatique (en anglais machine Learning : ML), il propage rapidement à chaque fois une entreprise introduit un produit intégrant des algorithmes de Machine Learning.

L'apprentissage automatique est la technique qui donne aux ordinateurs la capacité d'apprendre sans être programmé, il est intensément utilisé dans la vie quotidienne. Radicalement, c'est la science qui permet aux machines d'interpréter, d'exécuter et d'analyser des données pour résoudre des problèmes du monde réel.

En effet, au cours des dernières années, l'apprentissage automatique nous a exposé des voitures autonomes, une reconnaissance d'image et de la parole et encore d'autres applications, essentiellement les applications qui s'accommodent à l'apprentissage par l'expérience et développent leurs forces décisionnelles sur une période de temps.

1.4 Les types d'apprentissage

En fonction des types des données disponibles, les chercheurs ou développeurs choisiront des types d'apprentissage automatique (i.e. algorithmes) selon ce qu'ils veulent prédire à partir de ces données, on distingue 4 types :

Apprentissage supervisé [URL13] inventé per Bishop et al.

Dans le domaine de l'apprentissage automatique et de l'intelligence artificielle, l'apprentissage supervisé fait référence à une classe de systèmes et d'algorithmes qui déterminent un modèle prédictif à l'aide de points de données dont les résultats sont connus. Le modèle est appris par formation à l'aide d'un algorithme d'apprentissage approprié (tel que la régression linéaire, les forêts aléatoires ou les réseaux neuronaux, etc.) qui fonctionne généralement à travers une routine d'optimisation pour minimiser une fonction de perte ou d'erreur .

Par exemple, étant donnée l'image d'une personne, nous devons prédire son âge sur la base de l'image donnée, c'est un problème de régression.

Maintenant, étant donné qu'un patient est atteint d'une tumeur, nous devons prédire si la tumeur est maligne ou bénigne, ceci est un problème de classification.

Apprentissage semi supervisé [URL3] inventé par Chapelle et al.

L'apprentissage semi-supervisé (Semi Supervised Learning- SSL en anglais) est une technique d'apprentissage automatique qui étiquette une partie des données. Cette technique peut déduire ou apprendre ce que représentent les données non étiquetées avec une précision bien meilleure que dans l'apprentissage non supervisé (où aucune donnée n'est étiquetée), mais sans le temps et les coûts nécessaires à l'apprentissage supervisé (où toutes les données sont étiquetées).

Les modèles d'apprentissage semi-supervisé s'appliquent dans de nombreux secteurs. Prenons un exemple classique qui est l'analyse de la parole. Le fait de labéliser des fichiers audio est une opération très complexe qui exige un grand nombre de ressources humaines. L'application des techniques de l'apprentissage semi supervisé évoluera les modèles classiques d'analyse de la parole.

Apprentissage non supervisé [URL6] inventé par Barlow

L'apprentissage non supervisé est un type d'apprentissage automatique dans lequel un modèle doit rechercher des motifs (patterns en anglais) dans un ensemble de données sans étiquettes et sans supervision humaine. Dans l'apprentissage non supervisé, seules les entrées sont disponibles, et un modèle doit rechercher des motifs intéressants dans les données.

Un autre nom pour l'apprentissage non supervisé est la découverte de connaissances. Les techniques courantes d'apprentissage non supervisé comprennent le regroupement (clustering en anglais) et la réduction de la dimensionnalité.

Prenons un exemple de regroupement de Gènes. On prend une collection d'un million de gènes différents et on cherche un moyen de rassembler automatiquement ces gènes en groupes qui sont en quelque sorte similaires ou liés par différentes variables, telle que la durée de vie, l'emplacement, les rôles, etc.

Apprentissage par renforcement [URL20] inventé par Sutton et al,

En apprentissage automatique, l'apprentissage par renforcement consiste, pour un agent autonome (robot, etc.), à apprendre les actions à prendre, à partir d'expériences, de façon à optimiser une récompense quantitative au cours du temps.

L'agent est plongé au sein d'un environnement, et prend ses 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.

1.5 Les Intérêts de l'Apprentissage par renforcement

L'apprentissage par renforcement est considéré l'avenir de l'apprentissage automatique. En effet, il existe de beaucoup scénarios où il est irréalisable d'étiqueter les données d'une manière efficace.

Contrairement à l'apprentissage supervisé, l'apprentissage par renforcement ne demande pas beaucoup de données étiquetées. On peut considérer cela comme un point fort et d'intérêt dans le domaine d'apprentissage spécifiquement l'apprentissage automatique, car vu la quantité importante de données dans le monde qui évolue de plus en plus, il devient coûteux de les étiqueter pour toutes les applications.

De plus, l'apprentissage par renforcement est dédié pour résoudre des problèmes très complexes irrésolvable par des techniques classiques. Encore, cette technique innovante est très proche de l'apprentissage des êtres humains, en d'autres termes proche de la perfection.

En outre, la force de l'apprentissage par renforcement réside dans sa capacité à prendre des décisions dans des environnements complexes où la probabilité d'incertitude est élevée et la précision est très faible, en impliquant un nombre infini de combinaisons et une diversité de comportements possibles.

Le besoin de l'utilisation de l'apprentissage par renforcement est de plus en plus excessif. Citons à titre d'exemple les domaines des jeux, le domaine de l'industrie, la conduite autonome et la robotique.

Des études et des expériences ont été faites dans ces différents domaines et ont montrés l'efficacité de cette technique innovante. En effet, la NASA a construit des engins spatiaux qui ont visité tous les principaux objets du système solaire et exploré les nouvelles frontières de l'espace interstellaire. Cependant, dans le passé, l'exploration spatiale n'a pas toujours inclus des machines autonomes. En fait, la mission Apollo, qui a transporté les premiers humains sur la lune, a effectué un atterrissage correct grâce aux capacités de Neil Armstrong.

Bien que cette approche ait fonctionné dans le passé, elle n'est pas évolutive pour les missions futures. Il est de plus en plus essentiel que les futurs engins spatiaux soient autonomes, qu'ils soient capables de prendre seuls des décisions sérieuses. En 2020, l'envoi de « Perseverance » et « Ingenuity » sur Mars a été un succès considérable pour la NASA. Ces robots sont totalement autonomes.

Dans le domaine de la robotique, Google AI ont expérimenté l'efficacité de l'apprentissage par renforcement sur des robots réels où ils ont tourné sur une période de 4 mois. Cette méthode de Google AI avait un taux de réussite de 96 %. Dans le domaine des jeux, grâce à l'apprentissage par renforcement, AlphaGo Zero a pu apprendre le jeu de Go, en jouant contre lui-même. Après 40 jours d'auto-apprentissage, Alpha Go Zero a dépassé la version d'Alpha Go connue sous le nom de Master, qui a battu le numéro un mondial Ke Jie.

1.6 Description de la problématique

Grâce à la popularisation de certains modèles d'apprentissage par renforcement qui ont connu un grand succès essentiellement dans le domaine de Gaming. Mais si nous nous détachons de cette notion, nous trouverons de nombreux cas d'utilisation pratique de l'apprentissage par renforcement. Comme nous l'avons déjà vu dans la section précédente, l'usage de l'apprentissage par renforcement prend de plus en plus d'importance et d'ampleur dans des domaines comme les systèmes de recommandation, énergétique, robotique, etc.

Notre intérêt est pour la robotique. Les robots autonomes sont très utilisés dans l'exploration des planètes, la médecine et l'industrie. L'expérimentation est un élément clé de l'apprentissage par renforcement. Étant donné la complexité de ses méthodes, nous devons apprendre à connaître le comportement des algorithmes sur nos problèmes spécifiques de manière empirique. Pour ce faire, nous avons besoin d'un moyen d'expérimenter les différents algorithmes d'apprentissage par renforcement, d'où la nécessité d'une plateforme de simulation. Dans notre étude, nous allons commencer par les deux algorithmes les plus répandus : Q-Learning et SARSA. Ce simulateur est capable de supporter et d'expérimenter d'autres algorithmes d'apprentissage par renforcement.

1.7 Conclusion

L'apprentissage par renforcement est considéré comme une solution prometteuse qui peut permettre une prise de décision intelligente et réduire la complexité de différents problèmes d'optimisation pour les problèmes de navigation autonomes.

Chapitre 2

L'apprentissage par renforcement pour la navigation robotique

2.1 Introduction

Un agent qui se déplace dans un environnement inconnu pour atteindre un but est couramment considéré un problème de décisions séquentielles. Le but ciblé par un tel agent est fréquemment désigné par le problème alors que la stratégie d'arrivée au but est souvent incertaine. Dans ce cas, lorsque le modèle de l'environnement est indéterminé, il est important d'utiliser les algorithmes d'apprentissage qui vont permettre à l'agent d'arriver au but.

Comme notre projet porte sur la navigation d'un robot, un robot étant un agent autonome, l'objectif de ce chapitre est d'introduire le domaine des agents intelligents dans lequel se positionne notre travail. Nous définissons en premier lieu l'agent et les systèmes multi-agents. En second lieu, nous décrivons l'apprentissage dans les agents. Nous terminons ce chapitre par une description des algorithmes d'apprentissage par renforcement.

2.2 Définition d'un agent

Avant d'étaler les caractéristiques de l'apprentissage chez l'agent, il est crucial de définir la notion d'agent. Il existe plusieurs définitions de l'agent, mais elles diffèrent selon le type d'application.

Nous proposons la définition proposée par Jacques Ferber [URL7] « un agent est une entité logicielle ou matérielle, à laquelle est attribuée une certaine mission qu'elle est capable d'accomplir de manière autonome, disposant d'une connaissance partielle de ce qui l'entoure (son environnement), et agissant par délégation pour le compte d'une personne ou d'une organisation » .

La définition ci-dessus est précise parce qu'elle fixe les caractéristiques des agents suivantes :

- ▶ L'agent est doté d'une mission à accomplir et dispose de techniques pour décrocher son objectif.
- ▶ L'agent est une entité autonome ; son comportement est défini par sa propre expérience pratiquée dans son environnement.
- ▶ L'agent détient une connaissance partielle de son environnement qui lui permet de prendre des décisions.
- ▶ L'agent est caractérisé par un comportement souple, il répond d'une manière propice aux changements qu'il découvre dans son environnement et adopte la décision appropriée pour atteindre le but.

Possédant ces caractéristiques, l'agent est capable d'apprendre et d'évoluer en fonction de l'apprentissage en améliorant son comportement selon ses expériences passées.

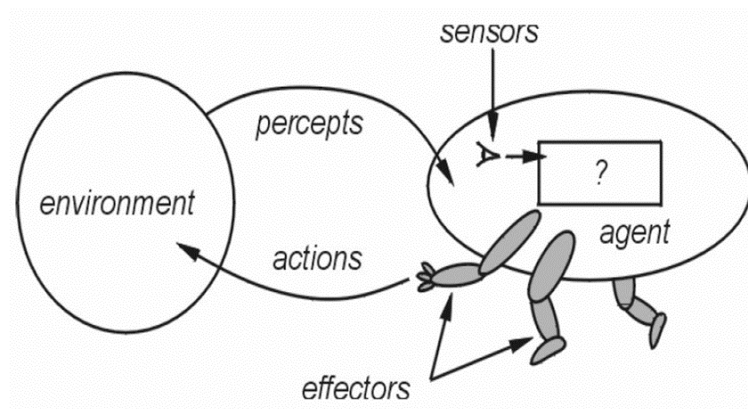


FIGURE 2.1 – Architecture d'un agent
sourced by : [URL22]

2.3 Les systèmes multi-agents

Les chercheurs en Intelligence Artificielle sont orientés vers une vision distribuée des systèmes pour plusieurs motivations. Au lieu de considérer un agent unique, difficile à maintenir et se dévoilant comme étant une ressource critique, ils ont choisi d'appliquer le principe « diviser pour régner ».

Les systèmes multi-agents se fondent sur le principe suivant : au lieu d'avoir un unique agent en charge de la totalité du problème, on considère plusieurs agents, chacun se charge d'une partie du problème. On favorise ainsi la résolution multi-agents à la résolution mono-agent.

Par définition, un système multi-agent est un système composé de plusieurs agents intelligents. Ces systèmes peuvent alors résoudre des problèmes qui sont difficiles ou impossibles à résoudre pour un agent individuel.

Ces dernières années, les systèmes multi-agents sont devenus une approche prometteuse pour résoudre les limitations du domaine médical qui est un vaste secteur qui se distingue par ses caractéristiques décisionnelles communes et distribuées, nécessitant une communication et une gestion complexe entre les différents départements médicaux. L'introduction de systèmes multi-agents dans les domaines médicaux facilite la prise de décision et les traitements, et assure la communication et la coordination en minimisant les erreurs d'analyse et de traitement, et en réduisant le temps nécessaire à la recherche des ressources médicales.

Les agents dans un système multi-agents ont plusieurs caractéristiques importantes. Ils sont autonomes, ils ont une vue locale car un agent n'a pas une vue globale complète et dont le contrôle est distribué.

2.4 Les concepts relatifs à l'agent apprenant

Dans cette section, nous allons introduire plusieurs concepts très importants de l'apprentissage par renforcement tel que les concepts de Markov, les méthodes d'exploration et la fonction de gain, etc.

Le diagramme suivant présente l'interaction entre l'agent et son propre environnement, dans notre cas l'espace de mobilité de l'agent. L'état (state) correspond à la position de l'agent dans l'environnement. La récompense (reward) est une variable qui permet de quantifier la valeur de l'état et l'agent aura comme but de maximiser la récompense cumulative.

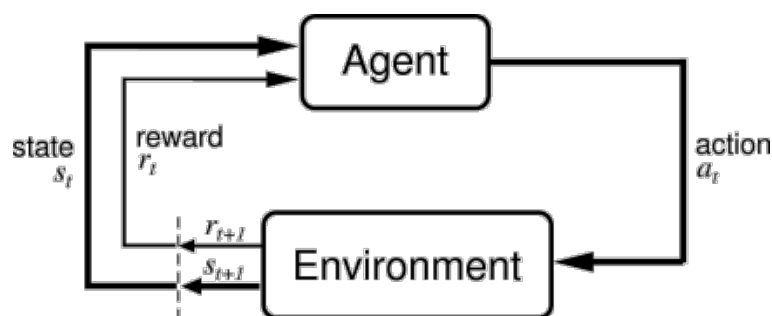


FIGURE 2.2 – Apprentissage par renforcement [?]

2.4.1 Processus de Décision Markoviens (MDP)

Un processus de décision de Markov est un processus de contrôle stochastique discret. À chaque étape, le processus est dans un certain état s et l'agent choisit une action a . La probabilité que le processus arrive à l'état s' est déterminée par l'action choisie. Plus précisément, le processus est décrit par la fonction de transition d'états $T(s, a, s')$. L'état s' est donc déterminé par l'état s actuel et l'action sélectionnée par le décideur. Cependant, pour un s et un a , le prochain état est indépendant des actions et des états précédents.

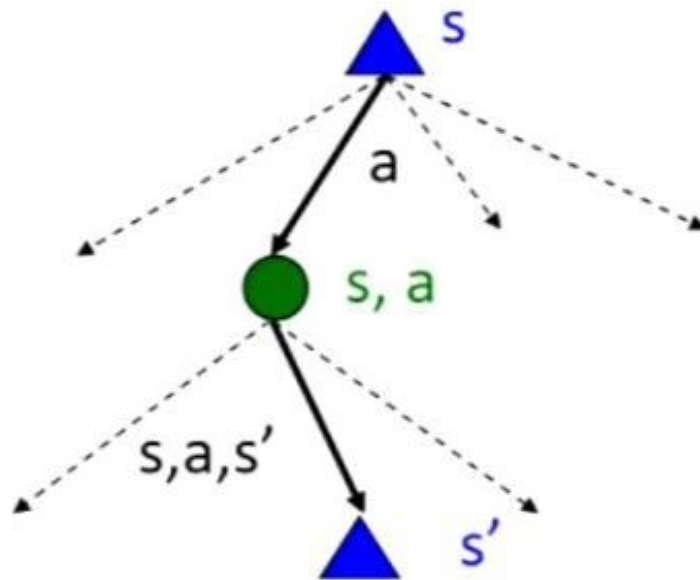


FIGURE 2.3 – Processus de Décision Markoviens
sourced by : [URL1]

Nous allons maintenant parcourir la terminologie utilisée par les méthodes d'apprentissage par renforcement :

► **Agent :**

Un agent est l'entité que nous formons pour prendre des décisions correctes. Par exemple, un robot qui est formé pour se déplacer dans un labyrinthe, tout en évitant les obstacles, pour atteindre la case terminale (la sortie).

► **Environnement :**

C'est l'environnement avec lequel l'agent interagit. L'agent ne peut pas manipuler l'environnement ; il ne peut contrôler que ses propres actions. On distingue deux types d'environnement, épisodique et non épisodique. L'épisode est une séquence d'interaction qui se situe entre un état initial et un état final.

En effet, une tâche non épisodique est une tâche continue dont la séquence d'actions est infinie, tel que le robot personnel assistant. Alors que la tâche épisodique est une séquence d'actions finies qui se termine dans états terminaux.

► **État :**

C'est la position de l'agent dans l'environnement.

► **Action :**

Comme son nom l'indique, une action est le choix de l'agent dans un état pour passer à un nouvel état.

► **Récompense :**

L'agent sera encouragé à trouver le chemin le plus court vers la cellule cible par un simple schéma de récompense. Donc c'est la récompense que nous donnons à l'agent après qu'il effectue son passage d'un état vers un autre.

Une fonction de gain/récompense définira la manière dont la récompense est mise à jour.

Les fonctions de récompense, appelée aussi fonctions de gain, sont utilisées pour les modèles d'apprentissage par renforcement. En effet, elle détermine les récompenses pour les actions. De plus, la fonction de gain a une grande importance parce qu'elle influence énormément l'apprentissage de l'agent.

► **Modèle :**

Le modèle est la vision de l'environnement par l'agent, c'est une fonction qui associe les paires état-action à des distributions de probabilité sur les états.

► **Politique (π) :**

La Politique (en anglais Policy) généralement notée π est la séquence d'actions exécutées par notre agent dans une des positions possibles. Autrement dit, elle définit le comportement d'un agent dans l'environnement.

L'équation ci-dessous montre que la politique est la probabilité que l'agent choisisse d'exécuter a dans l'état s .

$$\pi(a|s) = \mathbb{P}[A_t = a \mid S_t = s]$$

FIGURE 2.4 – La politique (ou Stratégie)

2.4.2 Exploration et Exploitation

Pour améliorer l'apprentissage par renforcement, deux opérations sont nécessaires :

► **Exploration :**

En choisissant une action aléatoire à chaque fois, nous explorons l'environnement. Ceci est fondamental au début de l'entraînement de l'agent parce que l'agent doit essayer d'avoir le plus d'informations possible sur l'environnement et essayer toujours quelque chose de nouveau. C'est ce qu'on appelle l'exploration

► **Exploitation :**

Après quelques épisodes, en explorant l'environnement, il sera temps de sélectionner des actions plutôt que de les choisir au hasard. C'est ce qu'on appelle l'exploitation de ce que nous avons appris.

Il existe pour cela plusieurs stratégies afin de maintenir un équilibre entre exploration et exploitation ; ne pas abandonner ni l'un ou l'autre, tel que la stratégie Epsilon-Greedy, Boltzmann-Exploration, etc.

La stratégie Epsilon-Greedy [URL12] est une méthode permettant l'équilibre entre l'exploration et l'exploitation de manière aléatoire où epsilon fait référence à la probabilité de choisir d'explorer ou d'exploiter.

La stratégie Boltzmann-Exploration [URL2] est une méthode où l'agent tire des actions d'une distribution de Boltzmann (softmax) sur les valeurs Q apprises, régulée par un paramètre de température τ .

2.5 Les Algorithmes d'apprentissage par renforcement

Les algorithmes de l'apprentissage par renforcement sont divisés en deux catégories distinctes : les algorithmes basés sur un modèle (model-based methods) et les algorithmes sans modèle (model-free methods).

Les algorithmes basés sur le modèle, comme son nom l'indique, se concentrent sur le modèle.

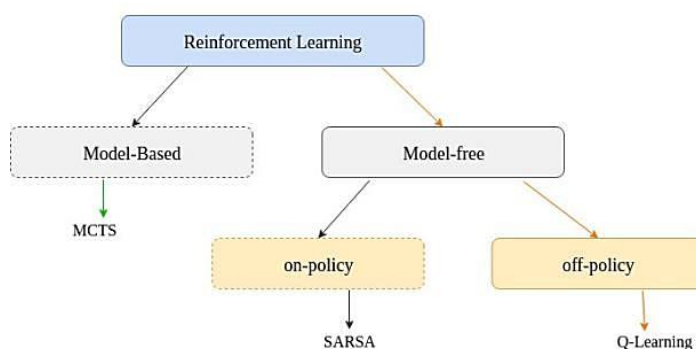


FIGURE 2.5 – La subdivision de l'apprentissage par renforcement
sourced by : [URL11]

En effet, l'agent exploite un modèle appris pour effectuer la tâche à accomplir.

Citons à titre d'exemple la programmation dynamique (en anglais Dynamic Programming) : itération de politique (en anglais Policy Iteration) et Q-Planning.

Tandis que dans les algorithmes sans modèle, l'agent s'appuie sur une expérience d'essais et d'erreurs passée pour réaliser sa tâche. Cette approche peut aboutir au même comportement optimal mais sans utilisation d'un modèle du monde. Parmi les algorithmes sans modèle, on cite le Monte Carlo, Q-Learning, SARSA, Actor Critic, etc. Les algorithmes sans modèle, eux-mêmes, sont divisés en 2 catégories, les méthodes ON-Policy et les méthodes OFF-Policy.

Les algorithmes ON-Policy (i.e. avec politique) cherchent à améliorer la politique (stratégie) que l'agent suit pour prendre des décisions. Tandis que les algorithmes OFF-Policy (i.e. sans politique) améliorent une politique (stratégie) différente de celle que l'agent suit. On peut dire alors que les algorithmes OFF-Policy sont indépendants des actions que prend l'agent et déterminent la politique optimale indépendamment de la motivation de l'agent. En vue d'évaluer la performance de l'agent au sein de son environnement, il est important d'implémenter des algorithmes d'apprentissage par renforcement.

Parmi les algorithmes que nous considérons les plus réputés et classiques dans le domaine de l'apprentissage par renforcement, nous avons choisi les méthodes suivantes :

- **Monte Carlo** est une méthode simple où l'agent apprend les états et les récompenses lorsqu'il interagit avec son environnement. Cette méthode n'a pas besoin d'un modèle, c'est-à-dire que l'agent ne connaît pas les transitions possibles dans l'environnement.
- **Q-Learning** est un algorithme OFF-Policy. Cela veut dire qu'il apprend la valeur de la stratégie optimale de déplacement dans l'environnement, indépendamment des actions de l'agent.
- **SARSA** est un algorithme similaire au Q-Learning mais il est un algorithme ON-Policy, qui cherche à optimiser une et une seule stratégie (politique) d'action.

2.6 Conclusion

Un agent logiciel peut être déployé pour la recherche d'informations pertinentes sur le net. Un agent logiciel peut aussi jouer aux échecs. En revanche, un robot peut être déployé pour ramasser des ordures dans les espaces verts, comme il peut être utilisé dans une chaîne de fabrication d'automobiles. Ainsi, quelle que soit la nature de l'agent, une entité logicielle ou matérielle, il a besoin d'un processus d'apprentissage par renforcement, afin d'être plus puissant et plus performant à résoudre les problèmes qu'il rencontre dans son propre environnement.

Dans ce chapitre, nous avons décrit l'apprentissage par renforcement en présentant ses concepts de base et les méthodes d'apprentissage les plus connues dans le domaine.

Nous allons approfondir cette étude avec plus de détails dans le chapitre suivant, qui décrira l'environnement de simulation permettant à un agent robot de se déplacer dans un labyrinthe avec l'objectif de trouver une sortie.

Chapitre 3

Simulation Robotique

3.1 Introduction

L'apprentissage par renforcement nécessite un très grand nombre d'épisodes de « Trial And Error » et d'interactions avec l'environnement d'où la nécessité de recourir à des simulateurs. De plus, dans un monde virtuel, l'agent peut facilement avoir accès à toutes sortes d'informations sur son simulateur, qui sont difficiles à découvrir dans notre propre monde physique.

Il est également beaucoup plus sûr de confiner ces agents autonomes dans des mondes virtuels pendant leur période d'exploration, car lorsqu'ils commettent une "erreur", celle-ci n'a que des répercussions limitées et peut, au pire, faire planter l'ordinateur sur lequel ils fonctionnent, alors que dans un monde réel, les interactions peuvent être dangereuses, comme dans le cas des voitures autonomes où les essais physiques sur les routes publiques sont peu sûrs, coûteux et pas toujours reproductibles. C'est là que les essais en simulation permettent de combler cette lacune. Si les tâches dans un domaine peuvent être simulées avec précision, l'apprentissage par renforcement améliorera considérablement l'état de l'art dans ce domaine au cours des prochaines années.

Il existe plusieurs plateformes et logiciels permettant de réaliser la modélisation et la simulation à base des agents, prenons à titre d'exemple le NetLogo, GAMA, CARLA [URL5], etc.

3.2 Description de l'environnement de navigation

Dans cette section, nous introduisons, en premier lieu, le problème puis nous présentons l'interface de simulation adaptée à notre application. En second lieu, nous détaillons la modélisation d'un labyrinthe dans le contexte de l'apprentissage par renforcement.

Les labyrinthes traditionnels ont été beaucoup utilisés dans la recherche sur les structures

de données et les algorithmes. En raison de leur familiarité et de leur nature intuitive, ce casse-tête est tout à fait approprié pour démontrer et tester les techniques d'apprentissage par renforcement.

L'environnement de notre problème est un labyrinthe avec une entrée, des murs, des impasses et une seule sortie. Dans un premier temps, un labyrinthe est généré aléatoirement en fonction de l'espacement.

Un agent est placé quelque part dans le labyrinthe. Le but du robot est d'apprendre à atteindre la cible, en optant pour le plus court chemin. Pour y arriver, l'agent parcourt le labyrinthe en une succession d'étapes en colorant le chemin emprunté. Pour chaque étape, l'agent doit décider quelle action entreprendre. A cet effet, l'agent apprend une politique (stratégie de déplacement) qui indique quelle est la meilleure prochaine action à effectuer. D'abord, il faut présenter la modélisation de l'environnement conçu. Les connaissances qui sont essentielles pour cette étude comportent : La modélisation des comportements et des mouvements possibles de l'agent et la modélisation de l'environnement.

3.3 Modélisation du problème de navigation

La modélisation est l'abstraction de la réalité afin de mieux comprendre ce qu'on réalise. C'est une étape importante avant d'entamer l'étude. En effet, la modélisation est un outil qui nous permet de maîtriser la complexité ainsi qu'elle offre une vue claire du problème. Dans cette section, nous modélisons le problème de navigation du labyrinthe parce que nous avons besoin d'une présentation claire du labyrinthe. Nous allons définir les états (states), les actions, les récompenses (rewards) que l'agent obtient en changeant d'état. Tout d'abord, le labyrinthe [URL9] est un ensemble de chemins compliqués dans lesquels il est possible de se perdre.

En d'autres termes, le labyrinthe est composé d'un ensemble de cases (des états) reliés par des couloirs.

L'agent, étant dans case du labyrinthe, peut se déplacer de cette case vers une autre qui est voisine. Les directions possibles de l'agent sont : avant, arrière, gauche et droite.

Les états de l'agent sont de 4 types :

- **Etat initial** : l'état initial de l'agent
- **Impasse** : c'est un état où l'agent doit rebrousser chemin.
- **Etat intermédiaire** : état (une case) normal sur le labyrinthe différent des impasses
- **Etat terminal** : c'est l'état cible du labyrinthe (la sortie)

Sans perte de généralité, nous allons travailler dans un environnement discret et déterministe, c'est-à-dire qu'étant donné un état actuel s de l'environnement et une action a , l'agent peut connaître avec certitude le prochain état qu'on note s' de l'environnement. L'état prochain, donc, ne change pas étant donné le même état s et la même action a pour deux instants différents.

► **Définition de la fonction de récompenses et pénalités**

Une des tâches les plus importantes dans la construction d'un modèle d'apprentissage par renforcement est la définition de la fonction de renforcement. Cependant, il peut être difficile de choisir la fonction la plus appropriée, puisque cette fonction détermine le comportement du système. De plus, la littérature aborde rarement le choix de la fonction de récompense.

Pour ce faire nous avons proposé la fonction des récompenses suivante :

Situations/Actions	Récompense
Arriver à une impasse	-20
Avancer vers le mur	-200
Arriver à la case cible	+100
Passer à une case normale	-1

TABLE 3.1 – Récompenses et Pénalités

Nous avons défini une fonction de récompense simple basée sur 4 situations possibles. Ces situations sont : l'agent a atteint le but, l'agent est avancé vers le mur, l'agent est dans une impasse et l'agent est dans case normale.

Nous imposons une pénalisation de -1 à l'agent pour chaque pas afin de l'obliger à suivre la route la plus courte pour atteindre sa destination. Ainsi qu'une autre pénalité de -20 est assigné lorsque l'agent entre dans une impasse afin de défavoriser le backtracking.

En plus, l'agent doit essayer de choisir la série d'actions qui lui permettra d'atteindre sa destination, tout en évitant les actions dangereuses, pour cela l'état mur a pour récompense -200.

En générale, les récompenses positives encouragent les actions associées. Alors que les récompenses négatives incitent l'agent à atteindre le but le plus rapidement possible, tout en évitant les états non privilégiés.

3.4 Environnement de simulation

NetLogo est un environnement de modélisation programmable et générique, dans le sens où il n'a pas été conçu pour un domaine d'application spécifique conçu pour le développement de système multi-agents et développé par l'université Northwestern en 1999.

C'est une plateforme gratuite, qui est entièrement écrite en Java, dédiée principalement aux étudiants pour leur faciliter la conception, le déploiement et la découverte des systèmes multi-agents.

La simulation NetLogo consiste en un monde constitué de rectangles (en 2D) ou de blocs (en 3D) appelés « patches » au sein desquels des agents mobiles appelés « tortues » peuvent se déplacer aléatoirement ou d'une manière intelligente et évoluer en fonction des tortues et des patches autour d'elles et de celles rencontrées sur leur chemin.

Un modèle Netlogo est constitué d'une interface graphique avec laquelle l'utilisateur interagit et suit la simulation.

3.4.1 Motivation de choix de NetLogo

- ▶ Interface utilisateur graphique simple à manipuler, avec une visualisation par défaut attractive et perspicace de modèle. On peut également visualiser le modèle en mode 3D.
- ▶ La possibilité d'adapter l'interface à nos besoins, en ajoutant et implémenter des nouvelles fonctionnalités.
- ▶ Une documentation importante à la fois pour les commandes et pour les modèles prédéfinis.
- ▶ NetLogo est fourni une bibliothèque des modèles préétablies qui peuvent être explorées et modifiées qui sont fournis par la communauté des utilisateurs dans divers domaines comme la biologie, computer science, jeux et mathématique.

3.4.2 L'interface graphique et les concepts d'agents dans Netlogo

Le monde de Netlogo est composé d'agents qui sont des entités qui suivent des instructions. En effet, dans Netlogo, il existe quatre types d'agents : turtles, links, patches, observer.

- ▶ Les « **turtles** » sont les agents qui se déplacent dans le modèle, ils peuvent interagir et prendre des décisions.
- ▶ Les « **links** » sont les agents utilisés pour établir des connexions entre les tortues, chaque extrémité d'un lien est une tortue.
- ▶ Les « **patches** » sont les petits pixels qui constituent le monde dans lequel les « turtles » peuvent se déplacer. Tous les patches ont des coordonnées `pxcor` et `pycor`.
- ▶ L'« **Observer** » : supervise tout ce qui se passe, fait tout ce que les autres agents ne peuvent pas faire pour eux-mêmes et leur donne des ordres spécifiques. Il collecte également des données pour créer des graphiques.

La figure 3.1 illustre l'interface graphique de l'environnement de simulation NetLogo :

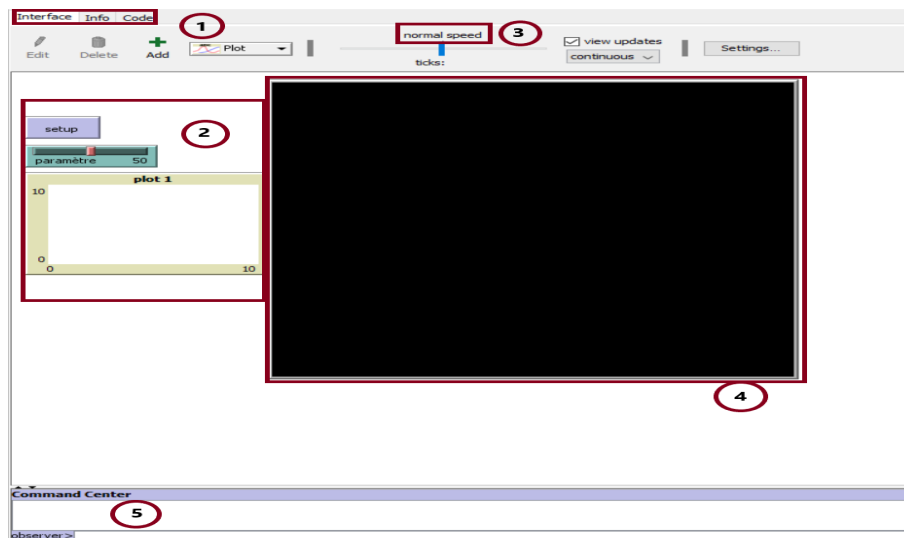


FIGURE 3.1 – L'interface utilisateur de NetLogo

1. ► **Interface** : cet onglet sélectionne l'interface principale. C'est ici que vous pouvez ajouter des éléments (boutons, curseurs, interrupteurs, etc.) qui contrôlent le modèle et vous permettent de visualiser les résultats.
 - **Info** : Cet onglet présente des informations sur le modèle (fonctionnement, réglage, utilisation ...).
 - **Code** : C'est l'endroit où vous pouvez écrire le code qui contrôle le modèle.
2. **Boutons, sliders, graphes** : sont utilisés pour interagir avec le modèle. Ils sont ajoutés en fonction de l'application.
3. **Contrôle de vitesse** : vous permet de contrôler la vitesse d'un modèle, et la fréquence de sa mise à jour.
4. **Modèle** : cette fenêtre affiche les emplacements des agents et l'état des « patches » (essentiellement ce qui se passe pendant l'exécution du modèle). Elle peut être visualisée en 2D ou en 3D.
5. **Le « command center »** : le centre de commande, situé en bas de l'interface, affiche les messages de sortie. Il peut également être utilisé pour interagir avec le modèle en cours d'exécution et le débogage.

3.5 Modélisation du labyrinthe avec NetLogo

Notre interface graphique est composée d'un certain nombre de sections, nous allons détailler les principaux composants ainsi que leurs fonctions comme l'indique la figure 3.2.

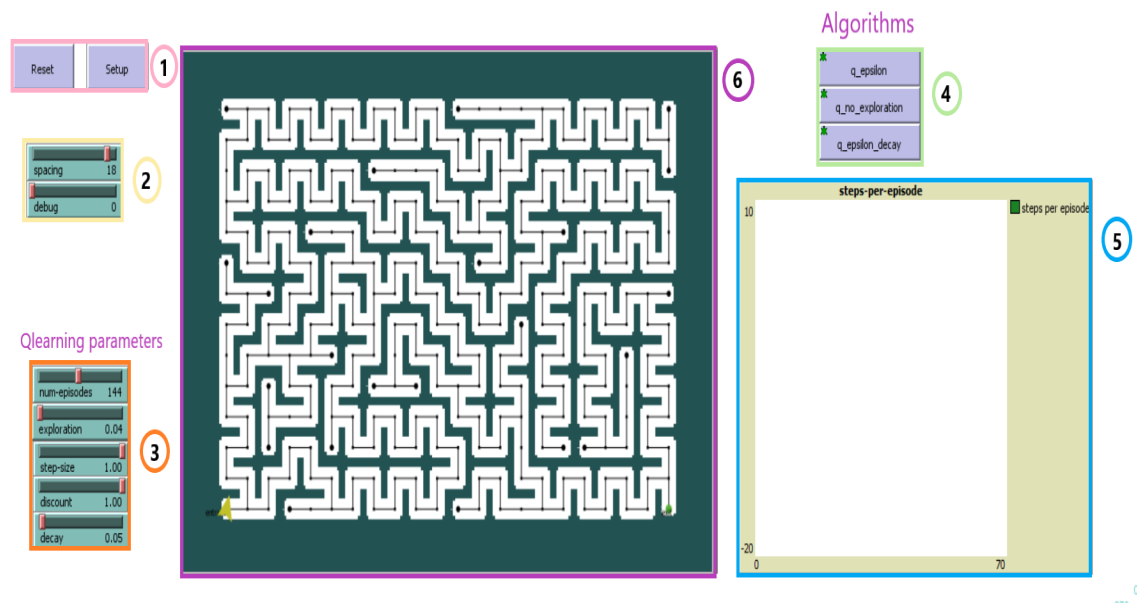


FIGURE 3.2 – Les fonctions sur NetLogo

- **Setup** : Ce bouton réinitialise la simulation. Cela efface la vue, crée un nouveau labyrinthe aléatoirement, initialise l'agent ainsi que la fonction de renforcement.
- **Reset** : Permet de réinitialiser la simulation tout en gardant le même labyrinthe et lancer une nouvelle exécution (utile pour comparer les algorithmes dans les mêmes conditions).
- **Spacing** : c'est un « slider » qui permet d'ajuster l'espacement pour créer un labyrinthe plus ou moins grand. Il varie de 1 à 20.
- **Debug** : peut prendre les valeurs 0,1 ou 2, utile pour le débogage, et contrôle la quantité de messages dans le command centre.
- **Parameters**(num-episodes, exploration, step-size, discount, decay) : ce sont des paramètres utilisés dans le processus d'apprentissage.
- **Algorithms** : Exécute un des algorithmes de l'apprentissage par renforcement implémentés avec le paramétrage indiqué.
- **Plots** : permettent de visualiser la progression de l'apprentissage et d'évaluer les performances de l'algorithme.
- **Environment** : c'est là où l'environnement et les agents sont instanciés

On distingue 2 types de labyrinthe, cela nous permet d'évaluer les performances des algorithmes en augmentant l'espace d'états comme l'indique les deux figures 3.3 et 3.4 .

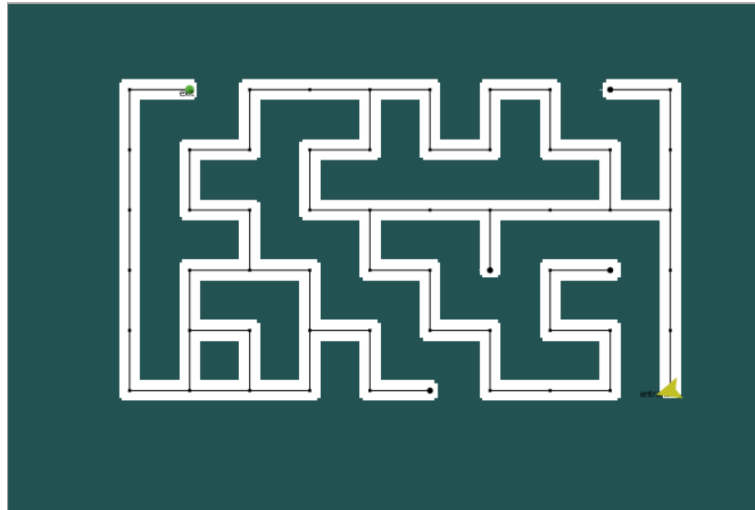


FIGURE 3.3 – Spacing=10

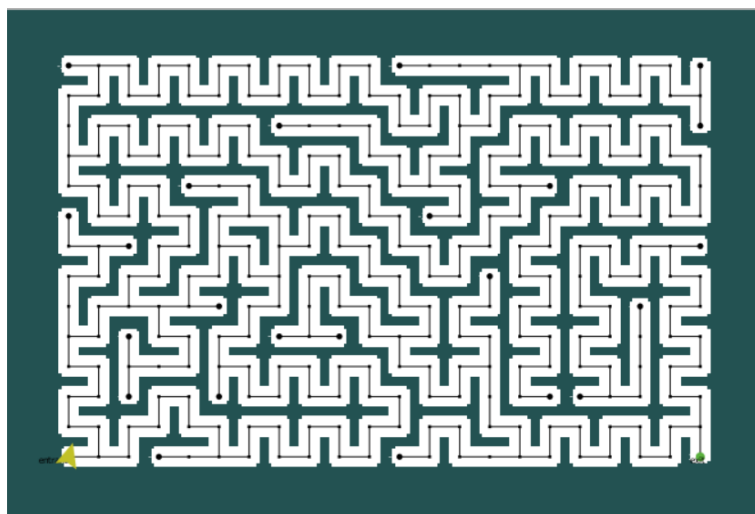


FIGURE 3.4 – Spacing=20

Un des défis de ce projet est de créer notre propre modèle de labyrinthe qui doit reproduire tous les scénarios possibles dans un environnement réel. . Nous considérons que le simulateur doit offrir un large éventail de configurations afin de permettre des expérimentations riches et proches de la réalité.

La génération du labyrinthe est effectuée par une application de l'algorithme de recherche en profondeur d'abord (Depth First Search) aléatoire.

Chaque nœud du labyrinthe représente un état possible comme l'indique la figure 3.5.

Chaque état est caractérisé par des informations nécessaires à l'agent pour interagir avec l'environnement et prendre des décisions, telles que les récompenses relatives, type de nœud (initial, but, impasse, intermédiaire), et la valeur de l'action relative à chaque paire état-action.

Finalement, deux impasses, choisies aléatoirement, sont sélectionnées pour définir le départ et la sortie du labyrinthe.

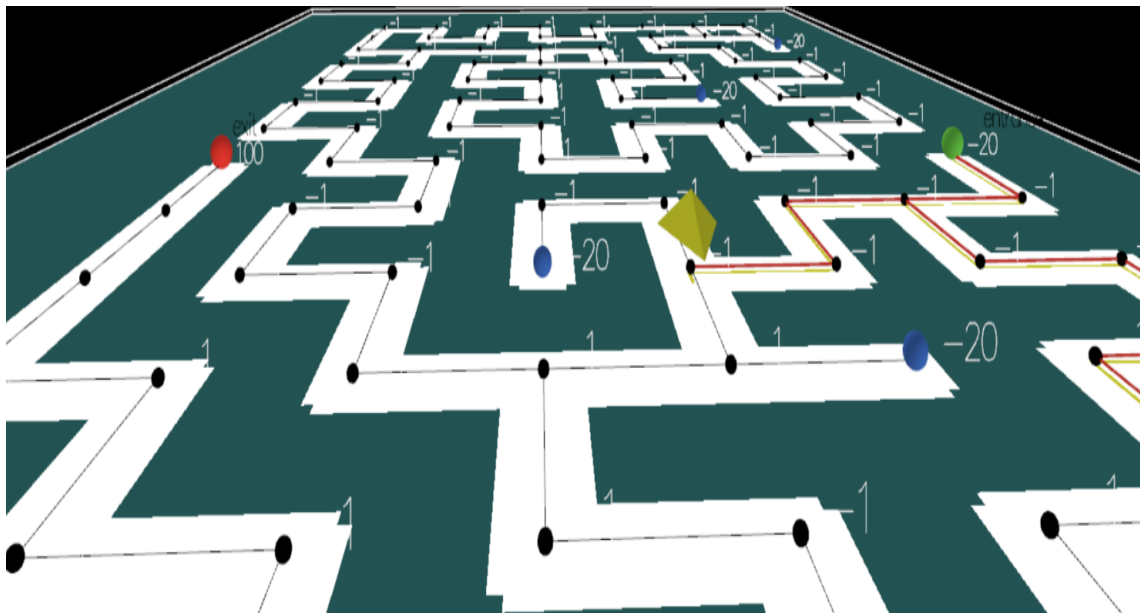


FIGURE 3.5 – Une scène de simulation

Identification des objets du labyrinthe :

- **Triangle jaune** : c'est notre agent apprenant.
- **Petits cercles noirs** : ce sont les états.
- **Grands cercles bleus** : ce sont les « Dead-Ends » ou les impasses.
- **Cercle vert** : C'est l'état initial.
- **Cercle rouge** : C'est l'état cible de notre environnement.
- **Tuiles blanches** : Ce sont les couloirs dans lesquels l'agent est autorisé à se déplacer.
- **Blocs verts** : Ce sont les murs du labyrinthe à travers lesquels l'agent ne peut passer.

Un épisode correspond à une simulation qui commence à l'instant où le système se trouve dans l'état initial et se termine à l'instant où le système se trouve dans l'état cible.

3.6 Algorithmes d'apprentissage par renforcement implémentés

Dans cette partie, nous allons détailler un peu plus les algorithmes implémentés, les adaptations qu'on a dû faire par rapport à la théorie pour pouvoir utiliser le Q-Learning dans sur un labyrinthe ainsi que l'algorithme SARSA.

3.6.1 Q-Learning

Le Q-Learning est l'un des algorithmes d'apprentissage par renforcement sans modèle et OFF-Policy pour l'apprentissage par différence temporelle. [6]

Plusieurs applications de l'apprentissage par renforcement, tel que le Gaming, la robotique, la médecine personnalisée, les systèmes de recommandations ont été construites à partir de cet algorithme.

Essentiellement, le Q-Learning permet à l'agent d'utiliser les récompenses de l'environnement pour apprendre, au fil du temps, la « meilleure » action à entreprendre dans un état donné.

La qualité de notre politique s'améliorera lors de l'apprentissage et continuera à s'améliorer de plus en plus, et pour apprendre cette politique optimale, nous allons utiliser l'équation de Bellman d'optimalité qui est la suivante .

$$Q^*(s, a) \Leftarrow E[R_{t+1} + \gamma \max_{a'} Q^*(s', a')] \quad (1)$$

L'équation (1) indique qu'à un instant t , pour toute paire état-action (s, a) , le retour espéré, partant de l'état s , en prenant l'action a , et avec la politique optimale, sera égal à la récompense attendue R_{t+1} que nous obtenons en choisissant l'action a dans l'état s , en plus du maximum du retour escompté qui est réalisable de toute prochaine paire état-action (s', a') .

La figure 3.6 représente le principe de l'algorithme.

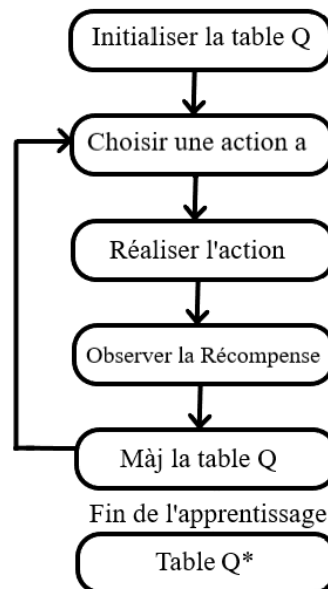


FIGURE 3.6 – Processus Q-Learning
sourced by : [URL15]

► **Q-table**

Q-Table est la structure de données utilisée pour stocker et calculer les récompenses futures maximales attendues pour l'action à chaque état. En d'autres termes, ce tableau nous guidera vers la meilleure action dans chaque état. En effet, il est sous forme de matrice comme l'indique la figure 3.7 où les colonnes présentent les actions sélectionnées et les lignes présentent les états que prend l'agent.

La question qui se pose est comment le Q-learning mettra à jour cette table ?

► **Q-values et leurs mises à jour**

Dans l'algorithme Q-Learning, le but est d'apprendre itérativement la fonction de valeur Q optimale en se basant sur la programmation dynamique et en utilisant l'équation d'optimalité de Bellman.

Pour ce faire, nous stockons toutes les valeurs Q dans un tableau, que nous mettrons à jour à chaque pas de temps. Au départ, toutes les valeurs de la table Q sont initialisées à zéros.

Il existe un processus itératif de mise à jour des valeurs. Lorsque nous explorons l'environnement, la fonction Q nous donne des approximations de plus en plus précises en mettant continuellement à jour les valeurs de Q-table.

Nous mettrons à jour la table Q en se basant sur l'équation (2) suivante :

Q-table		Actions			
		Avant	Arrière	Droite	Gauche
Etats	o	0	0	0	0

	i	0	0	0	0

n
	n	0	0	0	0

Training

Q-table		Actions			
		Avant	Arrière	Droite	Gauche
Etats	o	0	0	0	0

	i	-2.30	-1.96	-0.98	1.34

n	.	9.96	8.65	10.04	9.01
	n	9.96	8.65	10.04	9.01

FIGURE 3.7 – Un exemple de mise à jour de la Q-table
sourced by : [URL12]

$$Q(s, a) \Leftarrow Q(s, a) + \alpha [R + \gamma \max_{a'} Q(s', a') - Q(s, a)] \quad (2)$$

Avec :

- s = un état particulier
- a = une action
- s' = l'état suivant
- R = récompense de l'état s'
- γ = gamma ou « Discount factor » il indique à quel point les futures récompenses sont importantes pour l'état actuel. C'est une valeur comprise entre 0 et 1.
- α = taux d'apprentissage ou « Learning Rate » définit la vitesse d'apprentissage de l'agent.

Dilemme d'exploration exploitation Notre agent choisit l'action la plus « greedy » pour obtenir la meilleure récompense. L'agent sélectionne toujours l'action ayant la plus grande valeur Q . Ce comportement s'appelle exploitation [15]. Cette méthode présente un inconvénient majeur. L'agent n'agit que sur la base de sa connaissance de l'environnement, qui peut être incomplète. En agissant de manière gourmande sur la base de sa connaissance

limitée de l'environnement, l'agent a très peu de chances d'apprendre le comportement optimal dans l'environnement.

C'est là que l'exploration [URL17] entre en jeu. Lorsqu'un agent explore, il n'utilise pas nécessairement ses meilleures actions à chaque état, mais il explore les différentes options disponibles, guidé par une stratégie d'exploration. Donc l'exploration peut être décrite comme une tentative pour découvrir de nouveaux chemins de l'environnement en effectuant des actions sous-optimales, ou aléatoire. Ainsi l'exploration aboutit soit à une diminution de la récompense, soit à une augmentation de la valeur de la récompense. Notre agent se trouve face au dilemme suivant : explorer des actions non découvertes ou exploiter mes connaissances ?

Ce compromis entre l'exploration et l'exploitation est l'un des défis de l'apprentissage par renforcement car les performances optimales nécessitent généralement un certain équilibre entre les comportements d'exploration et d'exploitation. Pour ce fait nous allons recourir à une stratégie d'exploration appelé Epsilon-Greedy.

La dernière pièce du puzzle : Epsilon Greedy

La méthode Epsilon-Greedy [URL23] ou glouton est une méthode simple pour équilibrer l'exploration et l'exploitation en choisissant entre l'exploration et l'exploitation de manière aléatoire.

La plupart du temps l'agent sélectionne la meilleure l'action possible. Mais pour explorer plus d'options et potentiellement trouver une récompense plus élevée, on introduit le facteur epsilon. Cet epsilon fait référence à la probabilité de choisir d'explorer des nouvelles actions. La figure 3.8 présente l'algorithme de Epsilon-Greedy.

$$a(t) = \begin{cases} \max Q_t(a) & \text{avec probabilité } 1 - \textit{epsilon} \\ \textit{random}(a) & \text{avec probabilité } \textit{epsilon} \end{cases}$$

FIGURE 3.8 – Algorithme de ϵ -Greedy
sourced by : [URL10]

En résumé, nous allons comparer 3 versions de Q-learning en se basant sur la méthode d'exploration : Q-greedy, Q-epsilon-greedy et Q-epsilon-decay afin de pouvoir étudier l'effet du paramètre epsilon sur les performances.

3.6.1.1 Q-Greedy

Q-greedy choisit toujours l'action que la politique (stratégie) pense être la plus performante, même si cette politique n'a pas encore convergé.

Par conséquent, cette méthode représente l'exploitation pure, c'est à dire le facteur d'exploration ϵ est nul. Cela signifie qu'elle a de très fortes chances de choisir une action dont les performances ne sont pas optimales et de s'y tenir, ce qui empêche l'amélioration de la politique.

La sélection d'action peut être décrite par :

$$a(t) = \underset{a'}{\operatorname{argmax}}(Q(s, a')) \quad (3)$$

Avec :

- $a(t)$: l'action que prend l'agent un instant t
- s : l'état courant où se positionne l'agent
- $\underset{a'}{\operatorname{argmax}}$: une fonction retourne l'action a' qui maximise la valeur de Q dans l'état s .

Algorithme1: Q-Greedy

Données: α : taux d'apprentissage, γ : discount factor
Résultats: table Q , la politique π^* estimée
 $Q(s,a) \leftarrow 0$ pour tout paire (s,a) ;
 Pour toute épisode faire:
 Initialiser l'état s
 Pour tout pas de l'épisode faire
 Faire:
 $a \leftarrow \underset{a'}{\operatorname{argmax}}(Q(s, a'))$;
 prendre l'action a ;
 Observer la récompense et l'état s' ;
 /*équation de mise à jour
 $Q(s, a) \leftarrow Q(s, a) + \alpha [R + \gamma \underset{a'}{\operatorname{argmax}} Q(s', a') - Q(s, a)]$;
 $s \leftarrow s'$;
 Tant que s est non terminale ;
 Fin
 Fin

FIGURE 3.9 – Q-Greedy

Le problème ici est que cette approche ne fait que l'exploitation, car elle choisit toujours les mêmes actions sans se soucier d'explorer d'autres actions qui pourraient renvoyer une meilleure récompense. Une certaine exploration est nécessaire pour trouver réellement un chemin optimal.

3.6.1.2 Q-Epsilon-Greedy

L'une des techniques d'exploration les plus simples est l'exploration Epsilon-Greedy (Sutton et Barto, 2018) [URL4]. L'idée est d'exécuter la meilleure action possible selon la politique la plupart du temps mais avec une probabilité ϵ de choisir une action aléatoirement.

Afin de stimuler constamment l'exploration, le principe est de choisir une valeur de ϵ constante. Cependant, comme il s'agit d'une constante, même si l'algorithme converge, l'agent continuera à explorer et à choisir des actions sous-optimales.

Comme expliqué précédemment, l'agent choisit une approche « Greedy » avec une proba-

Algorithme2: Q-Epsilon-Greedy

Données: α : taux d'apprentissage, γ : discount factor
Resultats: table Q, la politique π^* estimée
 $Q(s,a) \leftarrow 0$ pour tout paire (s,a) ;
 Pour toute épisode faire:
 Initialiser l'état s
 Pour tout pas de l'épisode faire
 Faire:
 $n \leftarrow$ nombre aléatoire entre 0 et 1
 Si $(n < \epsilon)$ alors
 $a \leftarrow$ action aléatoire de l'espace d'état
 Sinon
 $a \leftarrow \operatorname{argmax}_{a'} (Q(s, a'))$;
 Prendre l'action a ;
 Observer la récompense et l'état s' ;
 /*équation de mise à jour
 $Q(s, a) \leftarrow Q(s, a) + \alpha [R + \gamma \max_{a'} Q(s', a') - Q(s, a)]$;
 $s \leftarrow s'$;
 Tant que s est non terminale ;
 Fin
 Fin

FIGURE 3.10 – Q-Epsilon-Greedy

bilité de $1-\epsilon$ et choisit une approche exploratoire avec une probabilité ϵ .

3.6.1.3 Q-Epsilon-Decay

Cette stratégie est basée sur Epsilon-Greedy, mais la valeur de ϵ décroît avec le temps. En pratique, cela signifie que la stratégie commence avec une valeur de ϵ élevée, et donc un taux d'exploration élevé. Au fur et à mesure de l'apprentissage, le facteur ϵ doit diminuer afin que l'agent puisse exploiter les valeurs Q les plus optimales.

Algorithme3: Q-Epsilon-Decay

Données: α : taux d'apprentissage, γ : discount factor, ε : epsilon, μ : facteur de diminution
Resultats: table Q, la politique π^* estimée
 $Q(s,a) \leftarrow 0$ pour tout paire (s, a) ;
 Pour toute épisode faire:
 Initialiser l'état s
 Pour tout pas de l'épisode faire
 Faire:
 $n \leftarrow$ nombre aléatoire entre 0 et 1
 si $(n < \varepsilon)$ alors
 $a \leftarrow$ action aléatoire de l'espace d'état
 Sinon
 $a \leftarrow \operatorname{argmax}_{a'} (Q(s, a'))$;
 Prendre l'action a ;
 Observer la récompense et l'état s' ;
 /*équation de mise à jour
 $Q(s, a) \leftarrow Q(s, a) + \alpha [R + \gamma \max_{a'} Q(s', a') - Q(s, a)]$;
 $s \leftarrow s'$;
 Tant que s est non terminale ;
 $\varepsilon \leftarrow \varepsilon - \mu$;
 Fin
 Fin

FIGURE 3.11 – Q-Epsilon-Decay

3.6.2 SARSA

SARSA [URL20] introduite par Rummery and Niranjan est l'un des algorithmes d'apprentissage par renforcement sans modèle et ON-Policy pour l'apprentissage par différence temporelle. Nous rappelons que ON-Policy veut dire que la stratégie de navigation apprise est la même que celle utilisée pour faire les choix des actions à exécuter. SARSA désigne État-action-récompense-État-Action (en anglais State-Action-Reward-State-Action).

La figure 3.12 suivante donne une représentation du déroulement du SARSA.

► Q-values et leurs mises à jour

Dans l'algorithme SARSA, identiquement au Q-learning, le but est d'apprendre la fonction de valeur Q optimale en se basant sur la programmation dynamique et en utilisant l'équation d'optimalité de Bellman.

Comme déjà expliqué, toutes les valeurs Q seront stockées dans un tableau qui sera mis à jour à chaque pas de temps. Initialement, toutes les valeurs de la table Q seront mises à zéros.

La mise à jour de la table Q se base sur l'équation suivante :

$$Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma Q(s', a') - Q(s, a)] \quad (4)$$

Avec :

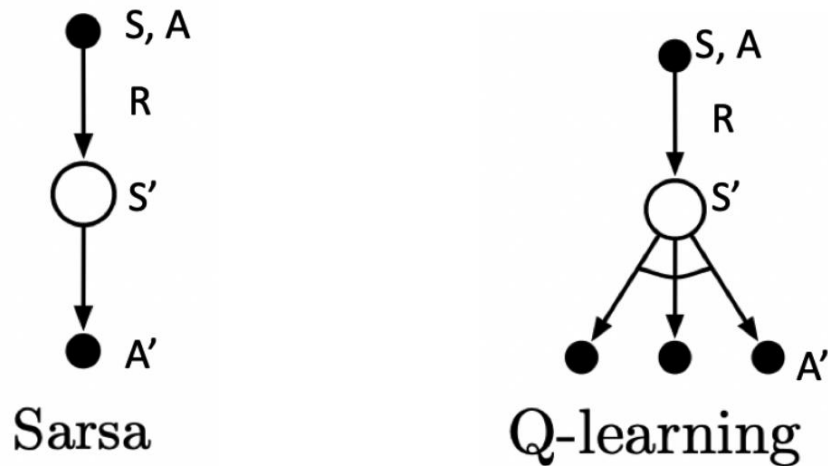


FIGURE 3.12 – : Différence entre les processus SARSA et Q-Learning
sourced by : [URL24]

- s = un état particulier
- a = une action effectuée par l'agent à l'état
- r = récompense suite au passage d'un état s vers l'état s'
- s' = l'état suivant
- a' = une action effectuée par l'agent à l'état s'
- γ = gamma ou « Discount factor » il indique à quel point les futures récompenses sont importantes pour l'état actuel. C'est une valeur comprise entre 0 et 1.
- α = taux d'apprentissage ou « Learning Rate » définit la vitesse d'apprentissage de l'agent.

Avec SARSA, l'agent prend place à un état s , exécute une action a , et obtient une récompense r . Actuellement, il est dans un état s' et exécute une autre action a' et met à jour la valeur de l'action a qui est effectuée dans l'état s .

La figure 3.13 présente l'algorithme de SARSA :

Comme remarqué dans le pseudo-code, la méthode d'apprentissage SARSA prend une deuxième action a' , qui est l'action réalisée par l'agent dans l'état s' .

Contrairement au Q-learning qui considère le meilleur cas possible si on arrive à l'état suivant, SARSA considère la récompense si on suit la stratégie actuelle vers l'état suivant.

Algorithme 4: SARSA

Données: α : taux d'apprentissage, γ : discount factor, ϵ : epsilon
Resultats: table Q, la politique π^* estimée
 $Q(s,a) \leftarrow 0$ pour tout paire (s,a) ;
 Pour toute épisode faire:
 Initialiser l'état s
 $n \leftarrow$ nombre aléatoire entre 0 et 1
 si $(n < \epsilon)$ alors
 $a \leftarrow$ action aléatoire de l'espace d'état
 Sinon
 $a \leftarrow \operatorname{argmax}_{a'} (Q(s, a'))$;
 Pour tout pas de l'épisode faire:
 Faire
 Prendre l'action a ; Observer la récompense et l'état s' ;
 /*choisir une action a' en utilisant la stratégie epsilon greedy */
 $n \leftarrow$ nombre aléatoire entre 0 et 1
 Si $(n < \epsilon)$ alors
 $a' \leftarrow$ action aléatoire de l'espace d'état
 Sinon
 $a' \leftarrow \operatorname{argmax}_{a'} (Q(s, a'))$;
 /*equation de mise a jour
 $Q(s, a) \leftarrow Q(s, a) + \alpha [R + \gamma Q(s', a') - Q(s, a)]$;
 $s \leftarrow s'; a \leftarrow a'$
 tant que s est non terminale;
 Fin
 Fin

FIGURE 3.13 – L'algorithme de SARSA

3.7 Conclusion

Après avoir présenté la modélisation du problème et l'interface de notre environnement, nous avons passé en revue les stratégies différentes sur l'exploitation-exploration que nous avons choisis pour comparer les performances sur le problème du labyrinthe.

Nous avons ainsi présenté le dilemme exploration/exploitation et les différentes versions à expérimenter par la suite. La qualité de ces différentes stratégies dépend fortement des choix des hyper-paramètres.

Pour évaluer la performance des algorithmes que nous avons décrits, nous allons présenter dans le chapitre suivant le processus d'optimisation des paramètres et les différentes mesures obtenues.

Chapitre 4

Résultats des Expérimentations

4.1 Introduction

En vue d’une étude comparative des différents algorithmes implémentés, nous passons à la phase d’expérimentation de ces algorithmes pour le déplacement du Robot dans le labyrinthe et l’évaluation de la qualité du chemin obtenu pour atteindre la sortie. Cette phase expérimentale nécessite l’ajustement et l’optimisation des paramètres utilisés et leur impact sur la qualité de la solution, dans notre cas il s’agit du plus court chemin.

Nous rappelons que l’expérimentation sera faite sur la plateforme de simulation robotique NetLogo.

Dans ce chapitre, nous présentons en premier le paramétrage utilisé dans nos expérimentations. Nous décrivons ensuite les résultats obtenus et leurs interprétations. En dernier lieu, nous finirons par présenter la comparaison entre ces algorithmes.

4.2 Paramétrage des algorithmes

D’après Henderson al. [URL16], la performance des algorithmes d’apprentissage par renforcement dépend essentiellement de la définition des hyper-paramètres.

Le problème de l’optimisation des hyper-paramètres apparaît lorsqu’un algorithme est dirigé par des hyper-paramètres, c’est-à-dire des paramètres qui ne sont pas appris mais qui doivent être choisis par l’utilisateur.

Le réglage des hyper-paramètres est une opération qui consiste à déterminer la direction du réglage afin de minimiser une fonction objective. L’ajustement manuel de ces paramètres de contrôle est une tâche pénible et consomme trop de temps et d’efforts, et devient parfois peu pratique si la solution a plus de paramètres et nécessite un réglage précis. Pour aborder ce problème, une méthode de sélection consiste à automatiser la recherche des paramètres. Donc une fois l’espace de recherche des paramètres est défini, un optimiseur ou un algo-

l'algorithme de recherche est nécessaire pour trouver le meilleur paramétrage.

La solution la plus intuitive consiste à tester toutes les combinaisons possibles en discrétisant l'espace des hyper-paramètres et tirer les meilleures valeurs. Cette technique est appelée brute-force ou recherche exhaustive [URL19].

Le but de l'optimisation des hyper-paramètres est de trouver la combinaison y^* minimisant la fonction objective x .

$$y^* = \underset{y \in Y}{\operatorname{argmin}} f(y)$$

Avec

- $x=f(y)$ représentant le nombre de pas effectué et c'est la fonction objectif à minimiser.
- y^* est la combinaison optimale de paramètres

Les résultats seront sous forme d'une courbe ayant comme abscisse le paramètre en question, et comme ordonnée le nombre de pas effectués sur n épisodes d'apprentissage.

La figure 4.1 montre un exemple de résultat de ce processus pour optimiser le paramètre discount factor γ .

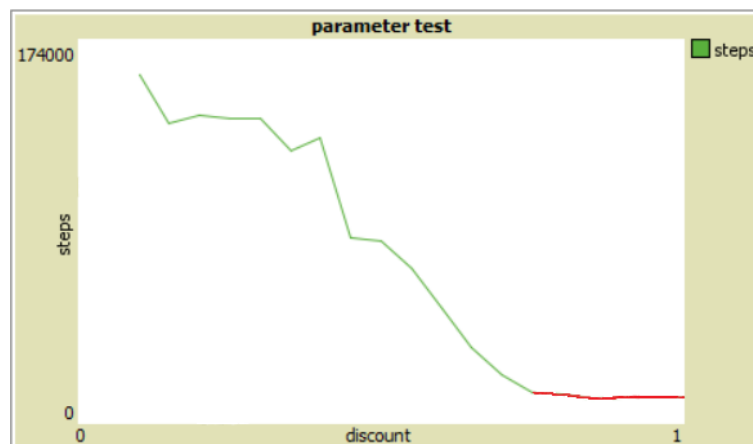


FIGURE 4.1 – Courbe des valeurs du paramètre discount factor γ

Le palier en rouge correspond aux valeurs du paramètre qui optimisent la fonction objective (nombre de pas effectués).

Ces résultats peuvent varier en raison de la nature stochastique de l'algorithme ou de l'environnement. Pour cela il faut exécuter l'exemple 10 fois pour prendre les résultats moyens.

Tous les paramètres seront étudiés selon le même principe.

Rappelons que les hyper-paramètres à considérer sont :

- α (**step-size**) : Permet de contrôler la vitesse d'apprentissage de l'agent.
- γ (**discount**) : est un coefficient qui est relatif à aux récompenses futures et nous rappelle à quel point elles sont importantes pour l'état actuel de l'agent.
- ϵ (**epsilon**) : est le facteur d'exploration qui mesure le taux d'exploration.
- λ (**decay**) : le taux d'exploration fixé au début de l'expérimentation diminue à chaque fin d'épisode avec un coefficient nommé facteur de dégradation d'epsilon.

Tous ces paramètres ont des valeurs réelles comprises entre 0 et 1.

Le tableau 4.1 montre les hyper-paramètres et résultats obtenus pour les différents algorithmes et pour les différents espacements du labyrinthe en prenant comme métrique le nombre de pas effectués.

Algorithme	Version	Taille du labyrinthe	α (step-size)	γ (discount)	ϵ (epsilon)	λ (decay)
Q-learning	Q-Greedy	20	0.9	0.9		
		10	0.825	0.95		
	Q-epsilon-Greedy	20	0.85	0.85	0.02	
		10	0.825	0.9	0.03	
	Q-epsilon-Decay	20	0.95	0.95	1 (initial)	0.05
		20	0.95	0.95	1 (initial)	0.05
SARSA	SARSA	20	0.9	0.875	0.02	0.05
		10	0.825	0.85	0.03	0.02

TABLE 4.1 – Les hyper-paramètres obtenus

Dans la littérature il existe une série de solutions pour ajuster les hyper-paramètres telles que les algorithmes génétiques [URL18], les méthodes évolutionnaires [URL8] et optimisation par essaims particuliers (en anglais particle swarm optimization) [URL14] qui sont un peu plus sophistiqués. Ces stratégies de recherche de paramètres améliorent significativement les performances de l'algorithme associé.

4.3 Résultats expérimentaux, analyse et interprétations

Dans cette section, nous allons présenter les résultats des expérimentations ainsi que les interprétations de chacun des algorithmes de la famille Q-Learning et ensuite de l'algorithme SARSA.

Dans nos expériences, nous avons laissé l'agent apprendre un nombre total de 200 épisodes (peut être modifié par l'utilisateur) pour conduire à la mise à jour complète de notre table q avec les meilleures actions qui peuvent être prises dans chaque état de l'environnement. Chaque expérience est répétée pendant 10 tours pour tester la cohérence de nos résultats.

4.3.1 Q-Learning

Les différentes versions de Q Learning ont été conçues pour une évaluation et une analyse approfondie de leurs performances. Vu que la distinction entre les trois versions réside dans le choix du facteur d'exploration, l'expérimentation des diverses versions de l'algorithme Q-Learning servira à étudier l'effet de l'exploration sur l'apprentissage.

Pour évaluer les performances des algorithmes, nous allons proposer une étude basée essentiellement sur les deux axes suivants :

- Le comportement par rapport à l'espace d'état (la complexité du labyrinthe)
- Le comportement dans un labyrinthe avec un état but statique, c'est-à-dire l'état cible reste inchangeable durant toute la phase d'apprentissage, et dynamique, c'est-à-dire un état cible plus proche s'ajoute au cours de l'apprentissage.

Les 2 critères principaux d'évaluation sont :

- Le plus court chemin (nombre de pas effectué par l'agent)
- La rapidité d'apprentissage (nombre d'épisodes effectué)

Dans cette partie nous allons présenter les résultats expérimentaux, les scénarios de test, et notre proposition pour améliorer la stratégie d'exploration dans l'application du labyrinthe.

Pour finir, les résultats de l'expérimentation seront des courbes représentant le nombre de pas effectués par épisode.

4.3.1.1 Q-Greedy

Rappelons que cette méthode se base sur l'exploitation pure. C'est-à-dire l'agent choisit toujours l'action ayant la meilleure valeur Q.

A. Expérimentation sur un labyrinthe simple statique

La courbe ci-dessous représente le nombre de pas cumulé pour chaque épisode. Le chemin obtenu est coloré en rouge dans le labyrinthe.

Malgré l'absence du facteur d'exploration epsilon, les résultats de l'expérimentation montrent que l'algorithme converge toujours vers la solution optimale. Cela peut être expliqué par le fait qu'aux premiers épisodes, le choix des actions était aléatoire, vu que la table Q est initialisée à zéro.

Donc l'exploration non-intentionnelle du labyrinthe a pu en quelque sorte solliciter l'agent à visiter tout l'espace d'états en un nombre important de fois et de converger vers la solution optimale dans un labyrinthe de petite taille.

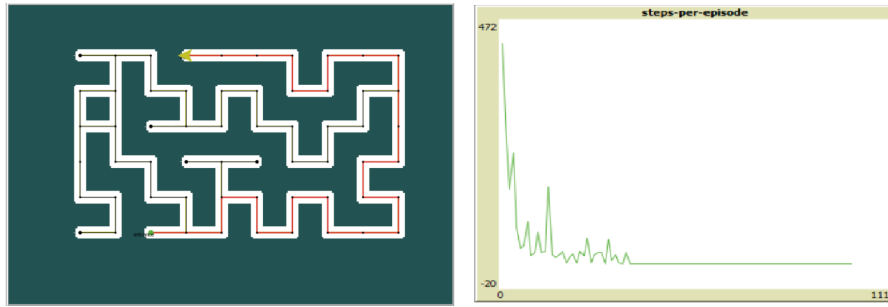


FIGURE 4.2 – Résultat Q-Greedy sur un labyrinthe simple

B. Expérimentation sur un labyrinthe complexe statique

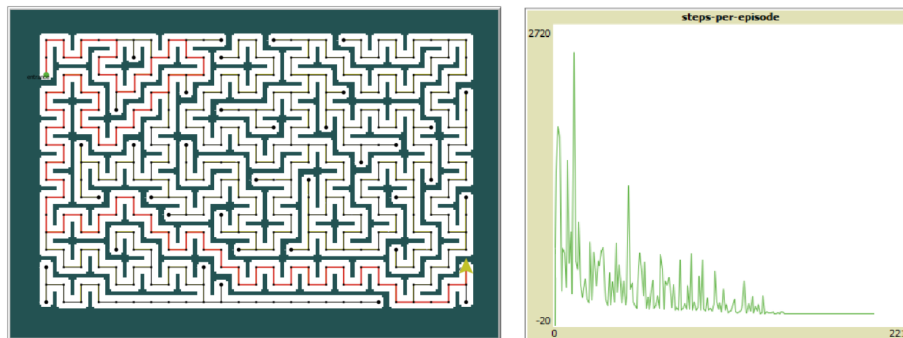


FIGURE 4.3 – Résultat Q-Greedy sur un labyrinthe complexe

A partir de ce résultat et des expérimentations exhaustives que nous avons faites, nous pouvons extraire deux problèmes principaux :

- Dans 10% des cas l'algorithme converge vers la solution sous-optimale (comme le montre la figure 5.3). Dans cette situation, l'aspect aléatoire temporel, dont nous avons parlé dans la partie (A), est insuffisant pour explorer tout l'espace d'états.
- L'agent peut se bloquer dans une boucle infinie entre deux états A et B, c'est-à-dire la meilleure action à prendre dans l'état A amène l'agent à l'état B et la meilleure action dans l'état B le ramène à l'état A.

C. Expérimentation sur un labyrinthe avec changement de but

Le but de cette expérience est de mettre en valeur l'importance de l'exploration et tester l'aptitude de l'agent à s'adapter aux changements d'objectifs. Une des solutions que nous avons envisagées est de créer un nouvel état but : si l'agent a appris un état but A1, est-il capable d'apprendre à atteindre le nouvel état but A2 ?

Dans cette expérience, nous utilisons la même taille du labyrinthe que la première expérimentation (A). Nos expérimentations montrent que la taille de l'espace d'états n'a pas d'influence sur la détection d'un nouveau cible. La position de départ est marquée en vert, et il n'y a qu'une seule sortie. Après un nombre important d'épisodes assurant la convergence vers la solution optimale, une nouvelle sortie plus proche de l'entrée marquée en rouge est rajoutée.

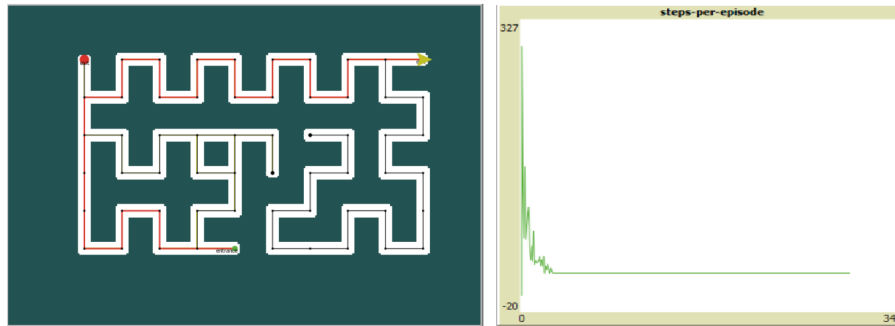


FIGURE 4.4 – Résultat Q-Greedy sur un labyrinthe simple avec changement de but

Les résultats ne sont pas surprenants. En effet, l'exploitation pure ne peut pas détecter les nouvelles opportunités, même dans un labyrinthe de petite taille et avec une centaine d'épisodes de navigation.

4.3.1.2 Q-epsilon-Greedy

Dans cette stratégie, l'agent utilise à la fois l'exploitation pour tirer parti des connaissances acquises et l'exploration avec un facteur fixe. Cela permet à l'algorithme d'explorer et de trouver éventuellement de meilleures solutions au problème posé.

Après avoir expérimenté l'algorithme Q-epsilon-Greedy, nous avons détecté quelques lacunes. Des améliorations sont alors nécessaires pour le rendre plus efficace.

► Première amélioration : exploration par épisode

Rappelons que la stratégie d'exploration choisie, au début, consiste à choisir entre l'exploration et l'exploitation à chaque pas de la navigation. Mais il s'est avéré, après plusieurs expérimentations, que lorsque l'agent a atteint sa cible, même en utilisant le facteur epsilon, son exploration est limitée aux cases adjacentes.

En effet, pour améliorer le comportement de l'agent, notre solution consiste à choisir d'explorer ou d'exploiter tout un épisode comme décrit dans l'algorithme 4, et non plus à chaque pas de l'épisode.

Algorithme 5: Q-Epsilon-Greedy

Données: α : taux d'apprentissage, γ : discount factor, ϵ : epsilon
Résultats: table Q, la politique π^* estimée
 $Q(s,a) \leftarrow 0$ pour tout paire (s,a) ;
 Pour toute épisode faire:
 Initialiser l'état s
 $n \leftarrow$ nombre aléatoire entre 0 et 1
 Si $(n < \epsilon)$ alors
 Pour tout pas de l'épisode faire:
 Faire:
 $a' \leftarrow$ action aléatoire de l'espace d'état
 Prendre l'action a ;
 Observer la récompense et l'état s' ;
 /*équation de mise à jour
 $Q(s,a) \leftarrow Q(s,a) + \alpha [R + \gamma \max_{a'} Q(s',a') - Q(s,a)]$;
 $s \leftarrow s'$;
 Tant que s est non terminale;
 Fin
 Sinon:
 Pour tout pas de l'épisode faire:
 Faire:
 $a \leftarrow \operatorname{argmax}_{a'} (Q(s,a))$;
 Prendre l'action a ;
 Observer la récompense et l'état s' ;
 /*équation de mise à jour
 $Q(s,a) \leftarrow Q(s,a) + \alpha [R + \gamma \max_{a'} Q(s',a') - Q(s,a)]$;
 $s \leftarrow s'$;
 Tant que s est non terminale;
 End
 End
 End
 End
 End

FIGURE 4.5 – Q-Epsilon-Greedy après amélioration

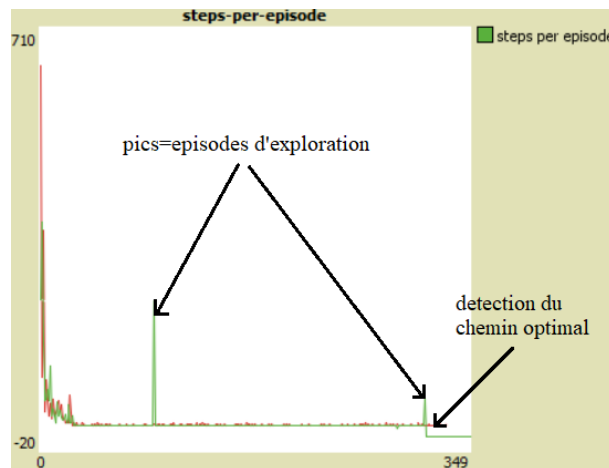


FIGURE 4.6 – Comparaison de l'algorithme avec exploration par pas (en rouge) et l'exploration par épisode (en vert)

Nous avons remarqué que l'agent explore mieux ce qui est prouvé par les 2 pics présentés par la figure 4.6. En effet, ces deux pics, correspondant aux pas cumulés pendant les épisodes d'explorations, montrent que l'agent, même après un bon apprentissage, a continué d'explorer son environnement et il a convergé vers la solution optimale.

Avec ces résultats, nous pouvons conclure que notre solution est efficace et améliore la stratégie d'exploration/exploitation.

► **Deuxième amélioration : Exploration guidée**

Comme expliqué précédemment, une exploration intensive de l'environnement est nécessaire, ce qui peut s'avérer irréalisable dans des environnements réels où l'exploration peut être dangereuse ou nécessite des ressources coûteuses, et même lorsqu'il est possible d'accéder à un simulateur et que l'exploration est sûre, la quantité d'interaction nécessaire pour trouver une politique raisonnable peut être trop coûteuse.

Une deuxième amélioration de la stratégie est de guider l'exploration pour la rendre plus efficace. Il est inutile, par exemple, de choisir l'action : tourner vers le mur, ou de choisir une nouvelle action quand il s'agit d'un chemin sans divergence c'est à dire l'agent n'est pas dans une intersection des chemins. L'intérêt est de diminuer l'espace de recherche et donc résoudre le problème plus rapidement.

A. Expérimentation sur un labyrinthe simple

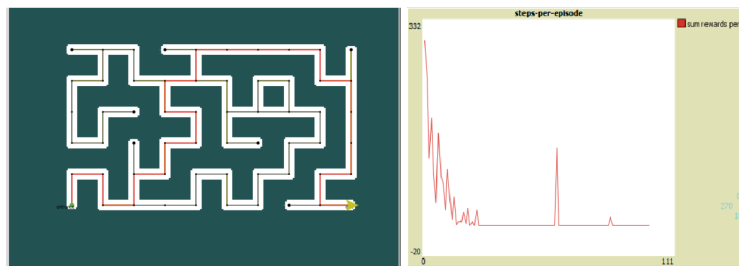


FIGURE 4.7 – Résultat d'expérimentation de Q-epsilon-greedy sur un labyrinthe simple statique

Comme prouvé dans la section précédente, l'algorithme Q-greedy converge toujours vers la solution optimale dans ce scénario particulier, et donc une exploration continue est inutile et peut dégrader les performances.

B. Expérimentation sur un labyrinthe complexe

Comme nous l'espérons, l'algorithme est capable de converger toujours vers la solution la plus optimale tout en gardant un taux d'exploration fixe. Cependant, cet algorithme donne de meilleures performances que le Q-Greedy.

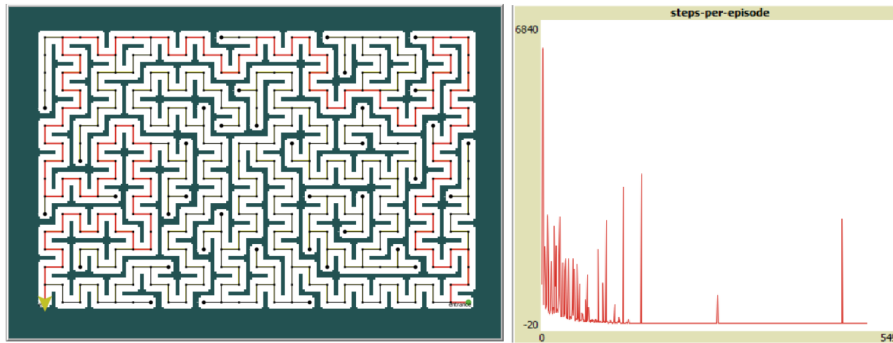


FIGURE 4.8 – Résultat d’expérimentation de Q-epsilon-Greedy sur un labyrinthe complexe statique

C. Expérimentation sur un labyrinthe avec changement de but

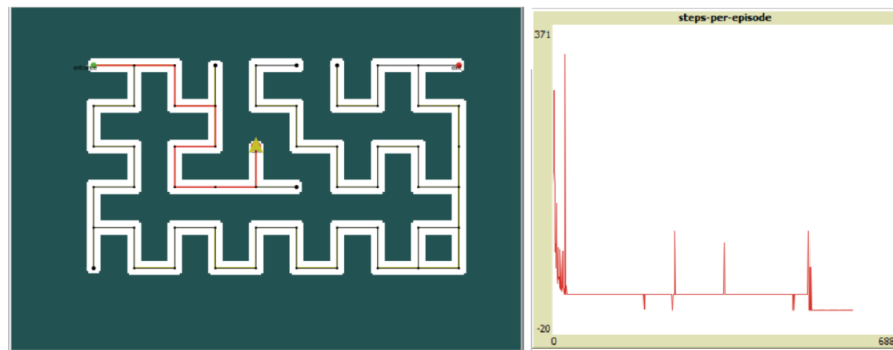


FIGURE 4.9 – Résultat d’expérimentation de Q-epsilon-Greedy sur un labyrinthe simple statique

Ces résultats démontrent que, pour le cas d’un labyrinthe avec changement de but, il est effectivement bénéfique d’utiliser la méthode Epsilon-Greedy. A priori c’est la seule parmi les versions de Q-learning proposés qui garantit la détection des changements de l’environnement.

4.3.1.3 Q-epsilon-decay

Cette stratégie commence à explorer de manière intensive dès le début, mais décroît jusqu’à ce qu’elle devienne une stratégie « greedy ».

Si l’algorithme Q-Greedy offre aussi cette possibilité d’explorer aux premiers épisodes, alors quelle est l’utilité de cette méthode ? Est-ce que le comportement sera pareil ? Pour répondre à cette question nous proposons une comparaison entre Q-Greedy et Q-Epsilon-Decay.

A. Expérimentation sur un labyrinthe simple

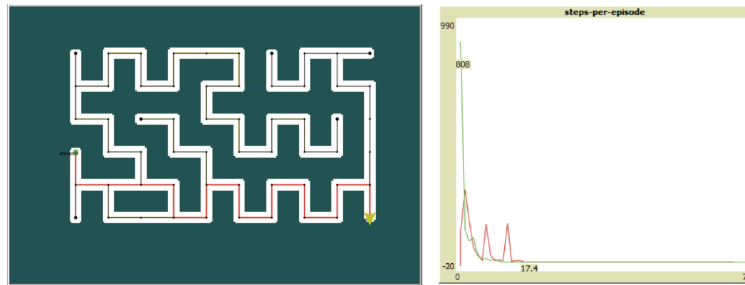


FIGURE 4.10 – Comparaison Q-Greedy (en rouge) et Q-epsilon-decay (en vert) sur un labyrinthe simple

Les deux algorithmes convergent vers le chemin optimal. Alors, afin de faire la comparaison, nous avons utilisé trois métriques : le nombre d'épisodes pour converger, le nombre de pas total et le temps nécessaire pour compléter tous les épisodes.

1. Le nombre d'épisodes nécessaire pour converger : la différence réside dans les premiers épisodes, nous pouvons observer clairement que le Q-epsilon-decay converge plus rapidement en nombre d'épisodes, mais le premier épisode est toujours trop lent par rapport au Q-greedy, cela est dû à l'exploration intense imposée par epsilon, ce qui permet à l'agent de découvrir une grande partie de l'espace d'état et de converger plus rapidement.
2. Le nombre de pas total pour chaque algorithme : les expérimentations montrent que les deux algorithmes requièrent presque le même nombre de pas moyen pour converger.
3. Le temps nécessaire pour converger : Le Q-greedy complète la tâche beaucoup plus rapidement par rapport à Q-epsilon-decay (le temps est suivi dans le centre de commande).

En se basant sur les trois mesures précédentes, nous pouvons dire dans cette configuration que le Q Greedy est plus performant que Q-epsilon-decay.

B. Expérimentation sur un labyrinthe complexe

L'expérimentation de Q-Greedy dans un labyrinthe complexe a montré que dans certains cas l'algorithme peut converger vers une solution sous-optimale. Contrairement, d'après les simulations Q-greedy-decay détecte toujours le chemin le plus court. D'où le

raisonnement sur les métriques précédentes n'est plus significatif.

Nous avons remarqué que le facteur lambda (decay rate) joue un rôle considérable, un bon

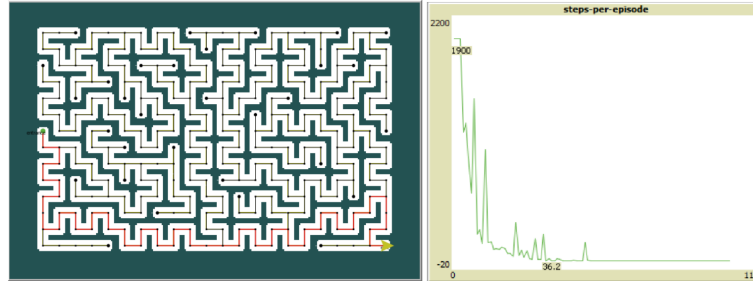


FIGURE 4.11 – Résultat de Q-epsilon-decay sur un labyrinthe complexe

facteur de dégradation est celui qui assure la convergence vers la solution optimale avant l'annulation du facteur d'exploration.

C. Expérimentation sur labyrinthe avec changement de but

Suite à nos expérimentations, nous avons constaté que cet algorithme peut se comporter différemment en fonction du facteur epsilon :

- Si le facteur epsilon est totalement annulé l'algorithme se comporte comme le Q-greedy comme l'indique la figure 4.12.

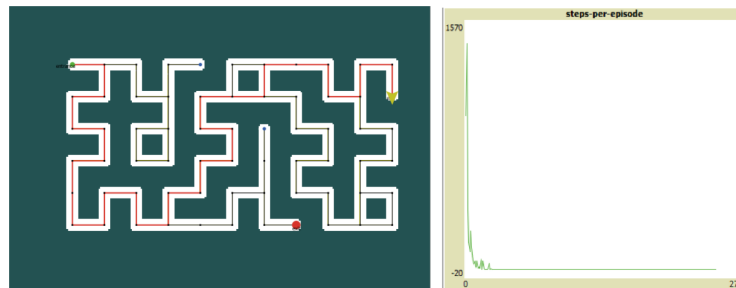


FIGURE 4.12 – Résultat Q-epsilon-decay sur un labyrinthe complexe avec epsilon déjà annulée

- Sinon l'algorithme peut converger vers la solution optimale si le facteur epsilon est capable de découvrir ce nouveau but avant son annulation comme l'indique la figure 4.13.

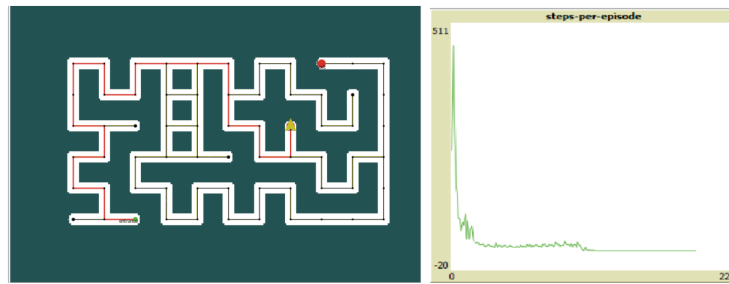


FIGURE 4.13 – Résultat de Q-epsilon-decay sur un labyrinthe complexe avec epsilon non encore annulée

4.3.2 SARSA

Dans cette partie, nous allons présenter les résultats expérimentaux de l'algorithme SARSA sur le problème de navigation d'un robot dans un labyrinthe. Nous avons choisi de se limiter à une expérimentation sur un labyrinthe complexe statique.

Les résultats de l'expérimentation seront des courbes représentant le nombre de pas effectué par épisode.

Dans un premier temps nous avons décidé d'expérimenter l'algorithme SARSA en utilisant comme stratégie d'exploration Epsilon-Greedy avec facteur epsilon constant. Les résultats n'étaient pas satisfaisants, l'expérimentation a montré que la convergence est relativement lente par rapport à Q-Learning. D'autre part nous avons remarqué que SARSA échoue, dans la plupart du temps, à converger vers la solution optimale. La figure 4.14 montre que l'algorithme SARSA est facilement bloqué dans le minimum local dans 15% des cas, il peut créer des épisodes infiniment longs.

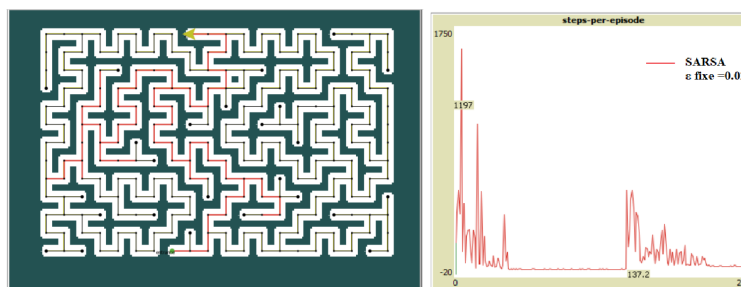


FIGURE 4.14 – Résultat de SARSA avec facteur epsilon constant

Pour remédier à ce problème, la solution était d'opter pour le epsilon decay où epsilon sera décrémenté au fil du temps jusqu'à atteindre 0. Nous avons gardé le même facteur de dégradation (decaying rate) trouvé dans le Q-learning.

Le résultat de l'expérimentation est décrit par la courbe illustré sur la figure 4.15 :

L'algorithme a donné des résultats meilleurs et il a pu converger vers la solution optimale.

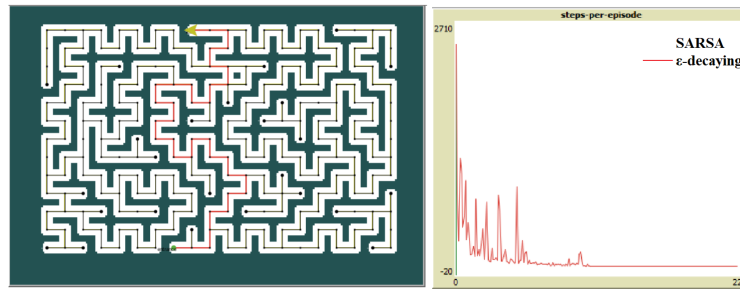


FIGURE 4.15 – Résultat de SARSA sur un labyrinthe complexe

L'interprétation à tirer de cette expérimentation est que SARSA qui est un algorithme ON-Policy, ne convergera pas vers les valeurs optimales de Q tant que l'exploration se poursuit. Cependant, en diminuant l'exploration au fil du temps, SARSA peut converger vers les valeurs optimales de Q , tout comme Q-learning.

4.4 Comparaison entre Q-Learning et SARSA

La différence entre l'algorithme Q-learning et SARSA est l'omission du terme **max** dans l'équation de mise à jour. Le Q-learning choisit la direction de la maximisation de Q à chaque mise à jour, et re-sélectionne l'action lors de la prochaine mise à jour. Pour trouver le plus court chemin dans un labyrinthe, une exploration poussée de tous les chemins possibles est obligatoire. Cependant, SARSA est plus réaliste que Q-learning. Il est préférable d'apprendre une fonction de valeur Q pour ce qui se passera réellement plutôt que ce que l'agent aimerait qu'il se passe. SARSA se base alors sur cette philosophie et planifie l'action pour le futur même avant la mise à jour de la valeur Q .

La figure 4.16 montre l'exécution des deux algorithmes :

La comparaison entre l'agent Q-epsilon-decay et l'agent SARSA était basée sur 2 critères :

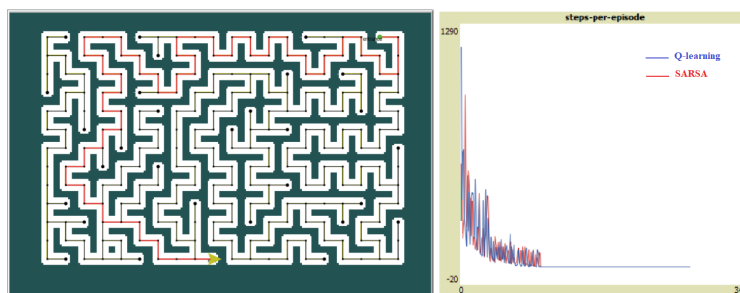


FIGURE 4.16 – Comparaison entre Q-epsilon-decay et SARSA en nombre de pas

le nombre de pas cumulés pendant l'apprentissage (NP) et le nombre d'épisodes nécessaires pour obtenir la meilleure récompense NE.

NP : nous l'avons obtenu en additionnant tous les pas de chaque épisode que l'agent termine. L'agent qui converge en un minimum d'effort sera le meilleur.

NE : avec cette métrique, nous voulons savoir quel agent est le plus rapide pour atteindre la meilleure récompense possible.

Les comparaisons sont faites sur un labyrinthe complexe statique en gardant la même configuration du labyrinthe. 10 exécutions sont simulées pour faire sortir les résultats moyens de chaque algorithme comme le montre la table 4.2.

	NP	NE
Q-epsilon-Greedy	25116	43
SARSA	26772	46

TABLE 4.2 – Comparaison entre Q-learning et SARSA selon les critères NP/NE

D'après les résultats, nous pouvons constater que l'algorithme SARSA a convergé moins rapidement vers la solution. De plus Q-epsilon-decay converge en effectuant un nombre de pas inférieur à celui de SARSA.

Lorsque la politique de l'agent est simplement gloutonne où epsilon decay, nous avons remarqué que Q-learning et SARSA produisent presque les mêmes résultats. Mais en introduisant le facteur epsilon fixe, le comportement change. Plusieurs expérimentations ont engendré ces différentes constatations :

- SARSA apprendra la politique sous optimale, le Q-epsilon-greedy, quant à lui, convergera vers la politique q^* optimale, d'où la différence en nombre de pas trouvés. Cette conclusion est déjà prouvée dans la littérature [URL25].
- SARSA s'approchera de la convergence en tenant compte des pénalités possibles des mouvements exploratoires, alors que Q-learning les ignorera. Cela rend SARSA plus conservateur, s'il y a un risque d'une grande récompense négative près du chemin optimal, Q-learning aura tendance à tomber dans ce piège lors de l'exploration, alors que SARSA aura tendance à éviter un chemin optimal dangereux [URL26].

En se basant sur ces résultats nous pouvons dire que dans la pratique, deux cas se présentent :

- Si les erreurs sont coûteuses, nous préférons un algorithme d'apprentissage plus sûr qui évite les risques élevés. Par exemple si l'agent est formé dans le monde réel et non pas en simulation, il est préférable de converger vers une solution sous optimale avec moins de risque que de converger vers une solution optimale dangereuse. Alors si des récompenses obtenues pendant l'apprentissage peuvent être coûteuse, alors SARSA peut être un meilleur choix.
- Si l'objectif est de former un agent optimal en simulation, ou dans un environnement à faible coût et à itération rapide, alors l'apprentissage Q est le meilleur choix, en raison du premier point (apprentissage direct de la politique optimale).

4.5 Conclusion

Ce chapitre détaille les expérimentations des deux algorithmes proposés Q-Learning et SARSA, nous avons montré l'importance du choix des hyper-paramètres et leur effet sur les performances de l'algorithme, puis nous avons décrit le processus d'optimisation de ces paramètres. Ensuite, une expérimentation exhaustive dans différents scénarios a été lancée afin d'évaluer les algorithmes et les modifications apportées en vue d'améliorer leurs performances. Enfin à partir des résultats et des expériences menées nous avons pu comparer les deux algorithmes. Nous avons aussi montré l'effet de la stratégie d'exploration sur les performances de SARSA.

Conclusion et perspectives

Ce présent rapport entre dans le cadre du Projet de Conception et Développement, ayant pour objectif l'expérimentation des méthodes d'apprentissage par renforcement, qui est l'une des approches de l'apprentissage automatique dans le domaine de l'Intelligence Artificielle (IA), en vue d'une étude comparative de celles-ci.

Tout au long de notre projet, nous avons présenté l'apprentissage en IA, pour cela nous avons présenté en premier lieu quelques définitions de base associées à l'IA, l'apprentissage automatique, ses caractéristiques et ses différents types.

En deuxième lieu, nous avons focalisé notre étude sur l'apprentissage par renforcement et ses techniques. En effet, nous avons introduit la notion d'agent, les systèmes multi-agents ainsi que les principaux concepts de l'apprentissage chez l'agent et les algorithmes d'apprentissage par renforcement les plus connus dans le domaine.

En troisième lieu, nous avons présenté l'importance de la simulation en vue d'expérimenter, tester et améliorer ces méthodes d'apprentissage automatique. Aussi, nous avons introduit l'environnement de simulation utilisé pour modéliser le problème de navigation robotique et le simuler sous NetLogo. Ensuite, nous avons décrit notre problématique ainsi que la mise en œuvre des algorithmes Q-Learning et SARSA. En dernier lieu, nous avons présenté les résultats et nous avons établi une étude comparative des algorithmes développés.

Cependant, nous avons rencontré quelques difficultés, au début du travail, pour construire l'environnement de simulation d'un labyrinthe. En effet, la plateforme de développement et de modélisation NetLogo est un environnement ayant un langage de programmation particulier. Des efforts étaient fournis pour apprendre à programmer en NetLogo et se familiariser avec l'environnement lui-même.

Ce projet constitue une base expérimentale à partir de laquelle, de nouvelles activités de recherche peuvent être lancées pour améliorer le travail présenté. Les perspectives que nous envisageons peuvent donc s'orienter vers les directions suivantes :

- Dans notre travail, nous avons traité l'apprentissage mono-agent. Afin d'améliorer le travail et aller encore plus loin, nous pouvons introduire d'autres agents.
 - Un passage du monde virtuel (le labyrinthe en tant que modèle) vers le monde réel est possible et, parfois, il est essentiel d'appliquer l'apprentissage par renforcement à des problèmes du monde réel où l'espace d'état/action est continu.
- Par conséquent, lorsque la taille du problème augmente, les besoins confronte une

croissance exponentielle en calcul.

Le deep reinforcement learning s'attaque au problème des grands espaces d'état en utilisant un réseau neuronal profond (DNN) afin d'approximer la fonction de valeur ou la politique.

- Utiliser l'apprentissage par transfert dans les algorithmes d'apprentissage par renforcement profond (Deep Reinforcement Learning).

Par conséquent, lorsque la taille du problème augmente, les besoins confronte une croissance exponentielle en calcul.

En effet, L'apprentissage par transfert [URL21] (transfer learning en anglais) est l'un des domaines de recherche de l'apprentissage automatique qui consiste à transférer les connaissances des tâches sources vers des tâches cibles. Il représente la capacité d'un système à appliquer des connaissances et des compétences apprises, à de nouvelles tâches partageant des ressemblances

Netographie

- [URL1] Stéphane Airiau. **Processus de décision markovien**. <https://www.lamsade.dauphine.fr/~airiau/Teaching/M1-DecUncertain/dui-10.pdf>, page5, Consulté le 02 avril 2021.
- [URL2] Nicolò Cesa-Bianchi al. **Boltzmann Exploration Done Right**. <https://papers.nips.cc/paper/2017/file/b299ad862b6f12cb57679f0538eca514-Paper.pdf>, Consulté le 23 avril 2021.
- [URL3] Olivier Chapelle al. **Semi-Supervised Learning**, 2017. www.acad.bg/ebook/ml/MITPress-%20SemiSupervised%20Learning.pdf, Consulté le 14 février 2021.
- [URL4] Rohan Rao al. **m-Stage Epsilon-Greedy Exploration for Reinforcement Learning**. http://aaai-rlg.mlancotot.info/papers/AAAI21-RLG_paper_48.pdf, Consulté le 01 mai 2021.
- [URL5] Steven F. Railsback al. **Agent-based Simulation Platforms :Review and Development Recommendations**. <http://cormas.cirad.fr/en/outil/classroom/example/StupidModel/railsback.pdf>, Consulté le 30 avril 2021.
- [URL6] H.B Barlow. **Unsupervised Learning**, 2017. <https://direct.mit.edu/neco/article/1/3/295/5492/Unsupervised-Learning>, Consulté le 15 février 2021.
- [URL7] Jacques Berber. **Les systèmes multi-agents**, 1997. https://www.lirmm.fr/~ferber/publications/LesSMA_Ferber.pdf, Consulté le 27 février 2021.
- [URL8] Maroun Bercachi. **Algorithme Évolutionnaire à États pour l'Optimisation Difficile**, 2010. <https://tel.archives-ouvertes.fr/tel-00818459>, Consulté le 04 mai 2021.
- [URL9] Université Libre de Bruxelles. **Labyrinthes**, 2004. https://www2.ulb.ac.be/soco/matsch/musee/expo/2000/laby_pan1.html, Consulté le 30 avril 2021.
- [URL10] geeksforgeeks. **Epsilon-Greedy Algorithm in Reinforcement Learning**, 2020. <https://www.geeksforgeeks.org/epsilon-greedy-algorithm-in-reinforcement-learning/>, Consulté le 01 mai 2021.
- [URL11] Nazia Habib. **Hands-On Q-Learning with Python**, 2019. <https://www.ebooks.com/en-us/book/209667999/hands-on-q-learning-with-python/nazia-habib/>, Consulté le 05 avril 2021.

- [URL12] Nazia Habib. **Q-Learning with Python**, 2019. www.subscription.packtpub.com/book/data/9781789345803/1/ch01lv11sec13/sarsa-versus-q-learning-on-policy-or-off, Consulté le 30 avril 2021.
- [URL13] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. **The Elements of Statistical Learning**, 2017. www.web.stanford.edu/~hastie/ElemStatLearn/printings/ESLII_print12.pdf, Consulté le 13 février 2021.
- [URL14] Yaru Li. **Hyper-parameter estimation method with particle swarm optimization**, 2020. <https://arxiv.org/pdf/2011.11944.pdf>, Consulté le 06 mai 2021.
- [URL15] mdbresources. **An Introduction to Deep Reinforcement Learning - Q-learning and Double Q-learning**, 2019. https://mdbresources.github.io/artificial_intelligence/q-learning-1, Consulté le 30 avril 2021.
- [URL16] Gabor Melis. **ON THE STATE OF THEART OF EVALUATION IN NEURAL LANGUAGE MODELS**, 2018. <https://arxiv.org/pdf/1707.05589.pdf>, Consulté le 02 mai 2021.
- [URL17] Thijs Nieuwdorp. **Dare to Discover : The Effect of the Exploration Strategy on an Agent's Performance**, 2017. https://theses.ubn.ru.nl/bitstream/handle/123456789/5216/Nieuwdorp%2CT._1.pdf?sequence=1, Consulté le 30 avril 2021.
- [URL18] Pierre. **Optimisation des hyper paramètres**, 2019. <https://www.anakeyn.com/2019/07/02/optimisation-des-hyper-parametres-xgboost-via-un-algorithme-genetique/>, Consulté le 02 mai 2021.
- [URL19] prabhu. **Understanding Hyperparameters and its Optimisation techniques**, 2018. <https://towardsdatascience.com/understanding-hyperparameters-and-its-optimisation-techniques-f0debb07568>, Consulté le 02 mai 2021.
- [URL20] Richard S. Sutton and Andrew G. Barto. **Reinforcement Learning : An Introduction**, 2014. <https://www.web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf>, Consulté le 21 février 2021.
- [URL21] Chuanqi Tan. **A Survey on Deep Transfer Learning**, 2018. [https://1808.01974.pdf>\(arxiv.org\)](https://1808.01974.pdf>(arxiv.org)), Consulté le 08 mai 2021.
- [URL22] John Tranier. **Vers une vision intégrale des systèmes multi-agents**, 2007. <https://www.tel.archives-ouvertes.fr/tel-00203489/document#figure.1.2>, Consulté le 18 mars 2021.
- [URL23] Robin van Emden. **Epsilon Greedy Exploration**, 2020. <https://paperswithcode.com/method/epsilon-greedy-exploration#>, Consulté le 30 avril 2021.

-
- [URL24] Lilian Weng. **A (Long) Peek into Reinforcement Learning**, 2018. <https://lilianweng.github.io/lil-log/2018/02/19/a-long-peek-into-reinforcement-learning.html>, Consulté le 02 mai 2021.
- [URL25] Marco A. Wiering. **Ensemble Algorithms in Reinforcement Learning**. <https://core.ac.uk/download/pdf/192286723.pdf>, Consulté le 07 mai 2021.
- [URL26] Jeremy Zhang. **Reinforcement Learning — Cliff Walking Implementation**. <https://core.ac.uk/download/pdf/192286723.pdf>, Consulté le 08 mai 2021.