الجمهورية الجزائرية الديمقراطية الشعبية
وزارة التعليم العالي والبحث العلمي

UNIVERSITE BADJI MOKHTAR - ANNABA
BADJI MOKHTAR – ANNABA UNIVERSITY

جامعة باجي مختار – عنابـــــــــة

Faculté : Sciences de l'ingéniorat

Département : Informatique

Domaine : Math et informatique

Filière : Informatique

Spécialité : Systèmes Embarqués et Mobilité

Mémoire

Présenté en vue de l'obtention du Diplôme de Master

Thème:

Étude de la sûreté des agents logiciels embarqués utilisant l'apprentissage par renforcement

Présenté par : *Zennadi Naama*

Encadrant : *Benazzouz Yazid* *Grade MCB* *Université Badji Mokhtar Annaba*

Jury de Soutenance :

Boudour Rachid	Pr	Université Badji Mokhtar Annaba	Président
Benazzouz Yazid	MCB	Université Badji Mokhtar Annaba	Encadrant
Benabbas Farouk	MCB	Université Badji Mokhtar Annaba	Examinateur

Année Universitaire : 2020/2021

ملخص

الأنظمة السيبرانية الفيزيائية (CPS) موجودة في كل مكان في مجتمعاتنا الحديثة. هذه هي وكلاء البرامج المضمنة مع الاتصال بالشبكات الرقمية وتتمثل وظيفتها الأساسية في التعاون من أجل التحكم في الكائنات المادية. توجد في العديد من التطبيقات مثل السيارة المستقلة والمنزل المتصل والتطبيقات في صناعة الطيران. الأسلوب المستخدم على نطاق واسع هو التعلم المعزز الذي يسمح للعامل بتعلم سلوك لم يتم تحديده مسبقًا. ثم يكتشف الوكيل البيئة والعواقب المختلفة لأفعاله من خلال التفاعل معها. يتعلم من تجربته الخاصة ، وليس لديه معرفة مسبقة بأهداف وتأثيرات أفعاله. ومع ذلك ، نظرًا لاستقلاليتهم ، فإن هؤلاء العملاء الأذكياء على متن الطائرة قادرون على التسبب في أضرار جسدية خطيرة نتيجة لخرق أمني أو عطل. في هذا العمل ، نستكشف مشكلة أمان وكلاء البرامج المضمنة في نظام إلكتروني فيزيائي ونبين من خلال دراسة حالة كيف يمكن جعل التعلم المعزز أكثر أمانًا.

Résumé

Les systèmes cyber-physiques (CPS, de l'anglais Cyber-Physical Systems) sont omniprésents dans nos sociétés modernes. Ceux sont des agents logiciels embarqués disposant d'une connectivité aux réseaux numériques et dont la fonctionnalité première est de collaborer pour le contrôle des objets physiques. On les retrouve dans plusieurs applications telles que la voiture autonome, la maison connectée et les applications de l'industrie aérospatiale. La technique, largement employée, est l'apprentissage par renforcement qui permet à un agent d'apprendre un comportement non préalablement défini. L'agent découvre alors l'environnement et les différentes conséquences de ses actions à travers des interactions avec celui-ci. Il apprend de sa propre expérience, n'ayant pas de connaissances préétablies des buts ni des effets de ses actions. Cependant, en raison de leur autonomie, ces agents embarqués intelligents sont capables de causer un préjudice physique grave à la suite d'une faille de sécurité ou un mauvais fonctionnement. Dans ce travail, nous explorons le problème de sûreté des agents logiciels embarqués dans un système cyber-physique et nous montrons à travers un cas d'étude comment il est possible de rendre l'apprentissage par renforcement plus sure.

Abstract

Cyber-Physical Systems (CPS) are everywhere in our modern societies. They are embedded software agents connected to digital networks and whose primary task is to collaborate to control physical objects. They can be found in several applications such as autonomous cars, the connected home and aerospace industry applications. The widely used technology is reinforcement learning, which allows an agent to learn a behavior not previously defined. The agent discovers the environment and the different consequences of its actions through interactions with that environment. It learns from its own experience, having no pre-established knowledge of the goals or of the effects of its actions. However, due to their autonomy, these intelligent embedded agents are capable of causing serious physical harm due to a security breach or malfunction. In this work, we explore the safety problem of embedded software agents in a cyber-physical system and show through a case study how reinforcement learning can be made more reliable.

Remerciements

Mon remerciement va en premier lieu à ALLAH le tout puissant de Nous avoir donné la foi et de nous avoir permis d'en arriver là.

Je tiens à exprimer mes plus vifs remerciements à mon directeur de mémoire monsieur Y.Benazzouz qui fut pour moi un encadreur attentif et disponible malgré ses nombreuses charges. Sa compétence,sa rigueur scientifique et sa clairvoyance m'ont beaucoup appris. Ils ont été et resteront des moteurs de mon travail de chercheur.

J'exprime tous mes remerciements à l'ensemble des membres de mon jury monsieur R.Boudour et monsieur F.Benabbas de m'avoir fait l'honneur d'accepter d'évaluer ce travail.

Le seul moyen de se délivrer d'une tentation, c'est d'y céder paraît-il ! Alors j'y cède en disant en grand Merci aux personnes qui ont cru en moi et qui m'ont permis d'arriver Au bout de cette thèse.

Je remercie toutes les personnes formidables qui existent dans ma famille et J'adresse toute ma gratitude à tous mes ami(e)s et à toutes les personnes qui m'ont aidé dans la réalisation de ce travail.

Table des matières

Table des figures

Liste des abréviations

CPS . . . . . Cyber Physical System
ES . . . . . Embedded System
IES . . . . . Intelligent Embedded Sytem
AI . . . . . Artificial intelligence
ML . . . . . Machine learning
RL . . . . . Reinforcment learning
MDP . . . . . Markov Decision Process

Chapitre 1

Introduction

Les systèmes embarqués ont énormément évolué au cours des dernières années, permettant l'interaction d'un nombre croissant d'agents logiciels embarqués entre eux ou avec nous-mêmes. Des agents avec différentes tailles, capacités, puissance de traitement et de calcul et prennent en charge différents types d'applications, participant ainsi à l'émergence de la technologie dans la vie quotidienne grâce à des applications dans divers domaines tels que : la santé, les villes intelligentes, les transports. En raison de la technologie avancée des systèmes embarqués, leur évolution est devenue un sujet crucial. L'association de l'ingénierie des systèmes embarquée avec l'intelligence artificielle qui est une technologie accessible utilisée dans de nombreux domaines. Elle utilise différentes techniques, parmi elles, l'apprentissage machine, plus précisément l'apprentissage par renforcement qui va permettre de développer des systèmes intelligents et autonomes. L'apprentissage par renforcement est un mode d'apprentissage statistique, inspiré de l'apprentissage humain et animal. Il consiste à apprendre à un agent comment interagir avec un environnement et associer des situations à des actions de façon à maximiser un signal de récompense numérique. L'agent doit découvrir les actions qui rapportent le plus en les essayant, Lorsqu'elles ont un résultat positif et induisent des récompenses, on conclut que ces expériences sont positives et qu'elles doivent être réitérées. Inversement, si le résultat de l'expérience n'est pas concluant, on le mémorise pour ne plus faire la même erreur. Ainsi, dans le cadre d'un apprentissage par renforcement, l'agent cherche à maximiser ses récompenses. Il n'y a pas si longtemps, les véhicules autonomes, les drones, les auto-robots alimentaient la science-fiction. Aujourd'hui, ils ont déjà été développés et sont continuellement améliorés, dans certains pays les auto-robots font déjà partie de la vie quotidienne. Leur autonomie les rend également vulnérables, car ils peuvent effectuer des actions qui peuvent mener à leur endommagement ou à des dégâts matériels et humains. Partant de ce constat, nous nous intéressons au problème de sûreté des agents embarqués, notre travail vise à protéger ces agents contre les risques sans dégrader leurs performances et par conséquent sécuriser leur travail.

1.1 Sûreté et sécurité des systèmes intelligents embarqués

Les systèmes embarqués d'aujourd'hui sont de plus en plus utilisés dans des contextes où la sûreté et la sécurité sont primordiales. Les systèmes doivent être extrêmement sûrs, ce qui signifie qu'ils ne doivent pas entraîner de risque inacceptable pour les humains, l'environnement ou l'équipement, ils doivent résister aux falsifications et aux tentatives malveillantes d'accès ou de contrôle du système. Une vulnérabilité dans la sécurité des systèmes embarqués donne aux pirates une chance d'accéder à des informations confidentielles, d'utiliser un système embarqué comme plateforme pour exécuter d'autres attaques, et même de causer des dommages physiques aux appareils qui peuvent potentiellement entraîner des dommages humains. Compte tenu du fait que les systèmes embarqués sont des composants de machines extrêmement coûteuses et précieuses, la possibilité de pirater ces systèmes attire de nombreux pirates. C'est pourquoi la sécurisation des systèmes embarqués est extrêmement importante. L'analyse des dangers des systèmes embarqués dépend généralement de la reconnaissance des composants logiciels et matériels vulnérables, mais les dangers peuvent aussi être dus à l'autonomie qui les caractérise et qui leur permet d'effectuer des actions qui les mettent en danger selon leur position dans l'environnement. Bien que la sûreté et la sécurité soient difficiles et coûteuses à réaliser en soi, il existe des méthodes pour les assurer telles que l'apprentissage automatique par renforcement qui permet au système d'apprendre à éviter les actions qui les mettent en danger. .[30]

1.2 Objectif de l'étude

Étant donné les travaux prometteurs ainsi que les problématiques liées aux agents embarqués, l'objectif principal de ce travail consiste à sécuriser et protéger les agents embarqués des diverses actions malveillantes qui peuvent nuire à leur travail en utilisant l'apprentissage par renforcement, sans affecter leur performance. Nous proposons donc un système qui assure la sûreté d'un agent en utilisant l'apprentissage par renforcement.

Chapitre 2

Système Cyber-Physique

De nos jours les innovations les plus intéressantes sont les systèmes autonomes appelés souvent systèmes cyber-physiques (CPS) par exemple : les voitures autonomes, surveillance médicale autonome, systèmes de contrôle industriel, systèmes robotiques et les systèmes de pilotage automatique, etc. Ces derniers créent une connexion entre le monde virtuel et le monde réel, ils sont déjà utilisés dans plusieurs pays dans le monde comme les trains sans conducteur en France et au Danemark. Nous aborderons dans ce chapitre la définition et les caractéristiques d'un système cyber-physique, ensuite nous allons décrire la relation entre les systèmes embarqués et l'intelligence artificielle ainsi que les défis de leur développement.

2.1 Les systèmes cyber-physiques

Les CPSs sont une extension des Systèmes Embarqués, fortement distribués et communicants, c'est la réalisation d'entités autonomes communicantes contrôlant des processus physiques avec un concept plus large comprenant la cybersécurité, le contrôle, l'évolution, l'optimisation, la validation et la vérification, etc.[38]

Tout systèmes cyber-physiques contient des systèmes embarqués intelligents qui reçoivent des informations du monde réel par le moyen de capteurs, ils traitent les informations en utilisant des algorithmes et des techniques d'intelligence artificielle, renvoient les résultats et effectuent des actions dans l'environnement via les actionneurs. Ils sont connectés les uns aux autres via des réseaux digitaux, et ils sont aptes à utiliser des services distants pour les assister.

Ces systèmes sont embarqués sur un élément physique pour lui permettre d'interagir avec l'environnement extérieur, prendre des décisions et effectuer des actions, donc l'entité devient autonome, intelligentes et ne nécessite pas l'intervention de l'être humain, par exemple un robot capable d'éviter des obstacles, une voiture autonome qui évite de faire un accident, etc.

La première définition des systèmes cyber-physiques a été donner en de 2006, lors de travaux avec la National Science Foundation (NSF) américaine qui invoquait que : « Les CPSs intègrent des processus physiques et compu-

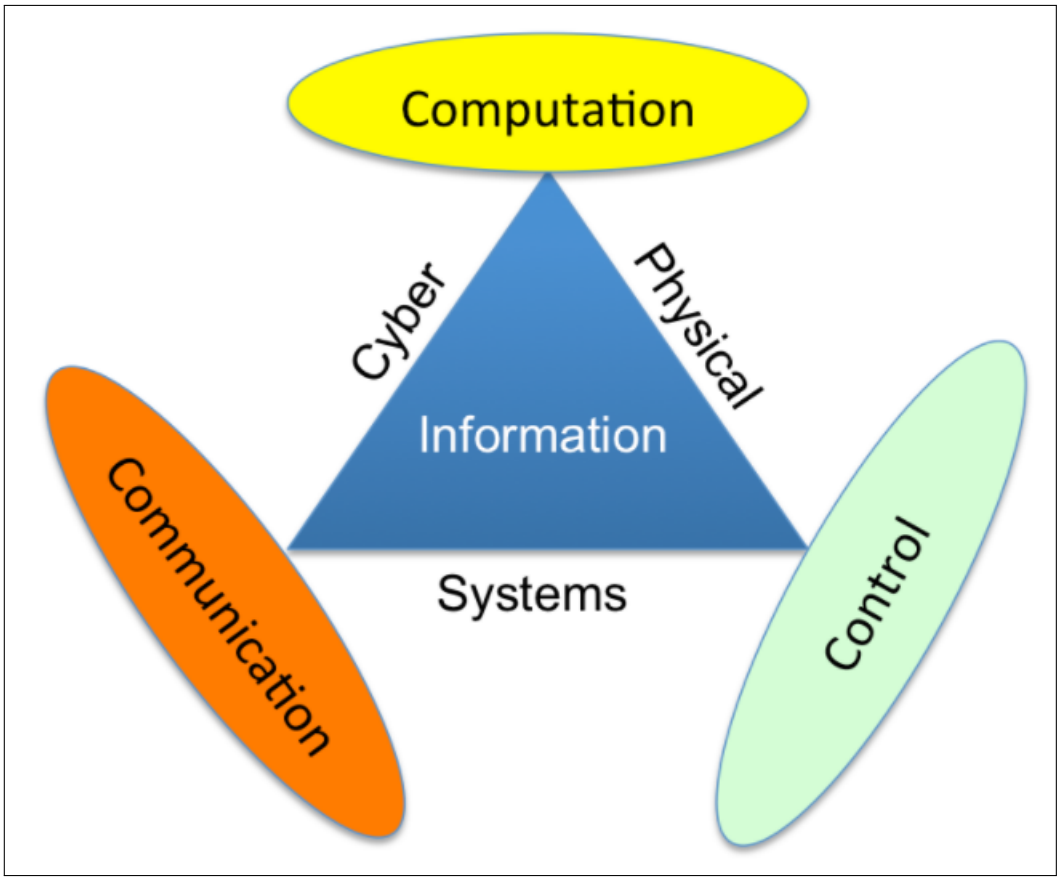

FIGURE 2.1 – Les composants de base d'un système cyber-physique

tationnels. Des ordinateurs et réseaux embarqués surveillent et contrôlent les processus physiques, généralement avec des boucles de rétroaction où les processus physiques affectent les calculs et vice-versa. En d'autres mots, les CPSs utilisent des computations et de la communication profondément intégrée et interagissant avec les processus physiques afin de produire de nouvelles capacités du système. Un CPS peut être considéré aussi bien à une petite échelle (ex. pacemaker) qu'à de grandes échelles (un réseau national de distribution d'énergie) » [24] Au fil des évolutions de ces systèmes leurs définitions, c'est affiner et se suffisait à : « Les CPSs sont des systèmes formés d'entités collaboratives, dotées de capacité de calcul, qui sont en connexion intensive avec le monde physique environnant et les phénomènes s'y déroulant, fournissant et utilisant à la fois les services de mise à disposition et de traitement de données disponibles sur le réseau » [28].

Les CPSs sont des systèmes assez complexes, ils se caractérisent par :

— Sécurité et confidentialité : les systèmes cyber-physiques sont basés sur la communication sans fil qui soulèvent souvent des problèmes de sécurité critiques. La sécurité est devenue un problème mondial.
— Fiabilité : les systèmes cyber-physiques doivent être certifiés dans certains cas, toute défaillance des composants peut entraîner la dégradation du système, ce qui peut causer des dommages matériels et humains.
— L'interopérabilité : Les systèmes cyber-physiques sont composés de plu-

sieurs sous-systèmes qui proviennent de divers fournisseurs, pour assurer la bonne communication entre les différents composants il faut savoir comment définir et utiliser des architectures communes, des interfaces normalisées et des normes de données.

— Robustesse : le système doit être capable de gérer les risques qui peuvent être difficiles à déterminer lors de la phase de conception. Afin de s'assurer que ces incertitudes sont prises en compte, elles doivent être suivies et traitées pendant les phases de mise en œuvre.

— Temps réel : Les systèmes cyber-physiques doivent s'assurer qu'ils disposent de la bande passante ou de la capacité du système nécessaire pour répondre aux fonctions critiques en termes de temps.

— Hautement connectés : parce qu'ils sont situés dans des environnements ouverts et il faut assurer la connectivité entre les différents composants. [21]

2.2 Les systèmes embarqués intelligents

Les systèmes embarqués intelligents (IES, de l'anglais intelligent embedded system)représentent une génération nouvelle et prometteuse de systèmes embarqués. Les IES ont la capacité de raisonner sur leurs environnements externes et d'adapter leur comportement en conséquence. De tels systèmes sont situés à l'intersection de deux branches différentes que sont l'informatique embarquée et l'informatique intelligente. D'un autre côté, les logiciels embarqués intelligents représentent une part importante du coût d'ingénierie des systèmes embarqués intelligents. [9] Ces systèmes peuvent inclure certains systèmes basés sur l'intelligence artificielle (IA) tels que des systèmes-experts, des réseaux de neurones et d'autres modèles sophistiqués d'intelligence artificielle (IA) pour garantir certaines caractéristiques importantes telles que l'auto-apprentissage, l'auto optimisation et l'auto-réparation.[23]

2.3 Défis de développement des systèmes embarqués intelligent

Dans l'évolution rapide des exigences de vie et de la technologie, les logiciels embarqués continuent de dominer les valeurs et les coûts de l'industrie des systèmes embarqués intelligents.[8] Si nous savons que lorsque nous nous référons à l'IA, nous nous référons automatiquement aux activités intellectuelles, humaines telles que la perception, l'apprentissage, le raisonnement et la mémorisation, l'auto-optimisation, l'auto-adaptation, etc. Le jugement de l'industrie est peut-être dû au fait que les activités intellectuelles prennent beaucoup de temps, ce qui peut être un goulot d'étranglement pour la performance, en particulier dans un contexte en temps réel, où les activités ou les tâches ont des délais ou une autre forme de contraintes de temps, ou peut-être en raison du fait que l'IA n'atteint pas un certain niveau de maturité surtout au stade pragma-

tique,afin qu'il puisse être appliqué efficacement dans des systèmes physiques réels.[27] Le jugement de l'industrie est peut-être dû au fait que les activités intellectuelles prennent beaucoup de temps, ce qui peut être un goulot d'étranglement pour la performance, en particulier dans un contexte en temps réel, où les activités ou les tâches ont des délais ou une autre forme de contraintes de temps, ou peut-être en raison du fait que l'IA n'atteint pas un certain niveau de maturité surtout au stade pragmatique,afin qu'il puisse être appliqué efficacement dans des systèmes physiques réels.

Dans un contexte, temps réel, le raisonnement est connu pour être un goulot d'étranglement en termes de performances donc pour résoudre ce dilemme, on peut par exemple paralléliser le raisonnement ou l'implémenter sous forme de composants matériels. Dans tous les cas, nous voyons que nous devons créer un pont entre les modèles d'IA et les méthodologies ES Codesign déjà bien pratiquées et les outils associés[**intelligent2017embedded**]

Chapitre 3

Apprentissage par renforcement

La connexion entre l'être humain et son environnement aide à acquérir de l'expérience sur les conséquences des actions et les décisions à prendre afin d'atteindre des objectifs. Durant toute la vie, ces interactions constituent la base de connaissance sur l'environnement et l'individu. Peu importe l'objectif à atteindre par exemple marcher, courir, lire, conduire une voiture... etc. La personne connaît la réaction de l'environnement et essaie d'influencer la réaction par son comportement.

Dans ce chapitre, nous nous attarderons sur l'apprentissage automatique par renforcement, comment décrire et résoudre un problème en apprentissage par renforcement en utilisant les MDP , les outils d'apprentissage, les différentes approches et leur classifications, ainsi que quelques algorithmes et leur application.

3.1 Vue d'ensemble sur l'apprentissage par renforcement

L'apprentissage par renforcement (RL, de l'anglais Reinforcement learning)consiste à apprendre comment associer des situations à des actions de façon à maximiser les récompenses. On ne dit pas à l'apprenant quelles actions il doit entreprendre, mais il doit plutôt découvrir les actions qui rapportent le plus en les essayant. Cette méthode d'apprentissage par renforcement est une branche de l'intelligence artificielle qui consiste à permettre à un agent d'interagir avec l'environnement afin d'apprendre à effectuer une tâche en réalisant une séquence d'actions.[25]L'agent reçoit une récompense s'il effectue une bonne action et une pénalité dans le cas contraire. Dans certains cas, les actions peuvent avoir un effet non seulement sur la récompense immédiate, mais aussi sur la situation suivante et, par conséquent, sur toutes les récompenses ultérieures. Les deux caractéristiques « recherche par essais » et « erreurs et récompense différée » sont les deux plus importants facteurs distinctifs de cet apprentissage. [36]

L'apprentissage par renforcement est différent de l'apprentissage supervisé, le type d'apprentissage étudié dans la plupart des recherches actuelles dans le

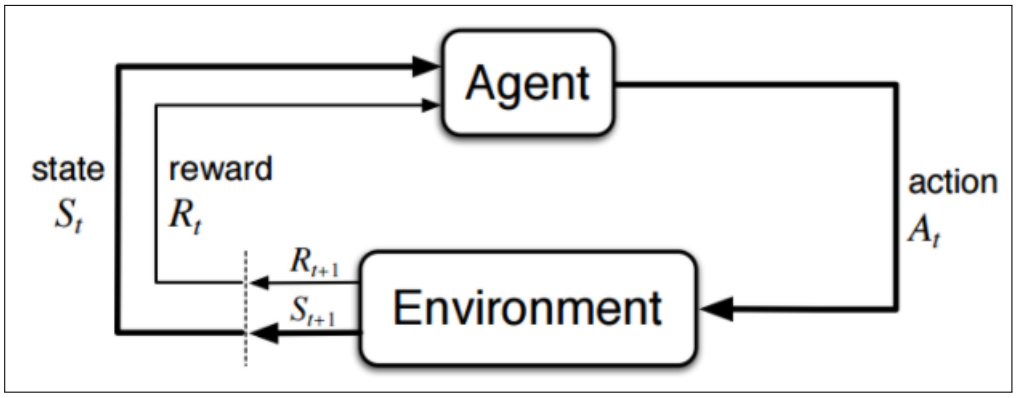

FIGURE 3.1 – Interaction agent-environnement

domaine de l'apprentissage automatique. Il consiste à apprendre à partir d'un ensemble d'exemples étiquetés fournis par un superviseur externe compétent. Chaque exemple est une description d'une situation ainsi qu'une spécification (l'étiquette) de l'action correcte que le système doit entreprendre dans cette situation.[34] Il s'agit d'un type d'apprentissage important, mais qui, à lui seul, ne permet pas d'apprendre à partir de l'interaction. Pas adéquat pour l'apprentissage à partir de l'interaction. Dans les problèmes interactifs, il est souvent peu pratique d'obtenir des exemples de comportement souhaité qui soient à la fois corrects et représentatifs de toutes les situations dans lesquelles l'agent doit agir. En territoire inconnu, où l'on s'attend à ce que l'apprentissage soit le plus bénéfique, un agent doit être capable d'apprendre de ses propres expériences.[22]

L'apprentissage par renforcement diffère également de ce que les chercheurs en apprentissage automatique appellent l'apprentissage non supervisé qui consiste généralement à trouver une structure cachée dans des collections de données non étiquetées. Les termes d'apprentissage supervisé et d'apprentissage non supervisé semblent classifier de façon exhaustive les paradigmes d'apprentissage, mais ce n'est pas le cas. Bien que l'on puisse être tenté de considérer l'apprentissage par renforcement comme un type d'apprentissage non supervisé, parce qu'il n'y a pas de différence entre les deux, parce qu'il ne repose pas sur des exemples de comportement correct, il tente de maximiser une récompense, au lieu d'essayer de découvrir une structure cachée.[34]

L'apprentissage par renforcement en termes formels est une méthode d'apprentissage automatique dans laquelle l'agent logiciel apprend à effectuer certaines actions dans un environnement qui le mènent à une récompense maximale. Il le fait par l'exploration et l'exploitation des connaissances qu'il apprend par des essais répétés de maximisation de la récompense.[10]

3.2 Formalisation

Nous formalisons le problème de l'apprentissage par renforcement en utilisant les processus de décision de Markov, l'idée de base est simplement de

saisir les aspects les plus importants du problème réel auquel est confronté un agent d'apprentissage qui interagit dans le temps avec son environnement pour atteindre un objectif. [39]

3.2.1 Interaction agent-environnement

Un processus de décision markovien (MDP, de l'anglais Markov Decision Process) est un framework très utilisé en apprentissage, car il décrit formellement les interactions entre un agent et un environnement, ainsi que le problème de prise de décision séquentielle. Ainsi, si quelqu'un souhaite résoudre un problème de prise de décision séquentielle, il peut transformer ce problème en MDP, et ainsi bénéficier d'outils, mais aussi de preuves que la solution optimale sera trouvée.[31]

- L'agent est abstrait et se caractérise seulement par son algorithme d'apprentissage et de prise de décision.

- L'environnement représente tout ce qui est en dehors de l'agent.

Ces deux entités interagissent de la façon suivante :

À chaque pas de temps t : 1- l'agent reçoit l'état de l'environnement S_t et la récompense R_t 2- l'agent effectue en conséquence une action A_t sur l'environnement. 3- à $t+1$, l'agent recevra le nouvel état S_{t+1} ainsi qu'une récompense R_{t+1}, qui dépendent de l'action A_t. Ainsi l'interaction agent-environnement génère une trajectoire :

$$S_0, A_0, R_1, S_1, A_1, R_2, ... \tag{3.1}$$

Cette interaction peut se terminer et elle est donc qualifiée d'épisodique ou elle peut aussi continuer à l'infini.

3.2.2 Processus de décision markovien (MDP)

Un MDP se définit à partir d'un tuple $M =< S, A, P, R >$:

— S est l'ensemble des états dans lequel peut se trouver l'environnement.
— A est l'ensemble des actions que l'agent peut choisir.
— P représente les dynamiques de l'environnement, c'est-à-dire les probabilités de transition entre chaque état.
— R décrit les récompenses obtenues en fonction d'un état et d'une action prise.

S et A sont pour l'instant considérés comme finis. Les dynamiques de l'environnement P ainsi que les récompenses R sont inconnues dans la partie model-free de l'apprentissage par renforcement. Dans le cas où les états et les actions appartiendraient à un ensemble discret, les MDP peuvent être représentés par un graphe où les nœuds sont des états et les actions des arcs pour naviguer entre les états comme sur la Figure suivante :

3.2.2.1 Dynamiques et propriété markovienne

Il s'agit de quelque chose qui imite le comportement de l'environnement, ou plus généralement, qui permet de faire des déductions sur le comportement

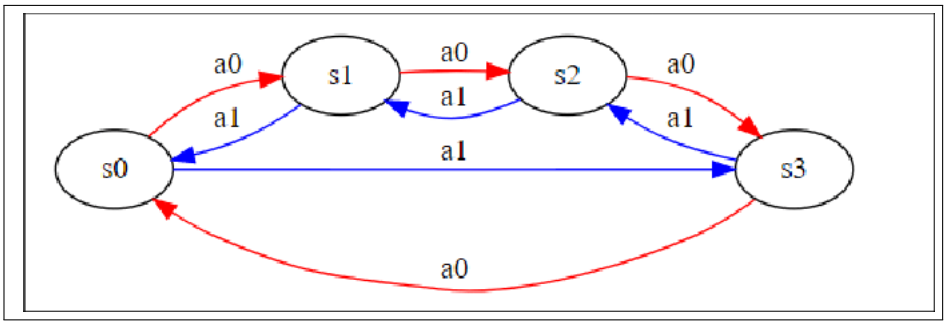

FIGURE 3.2 – Représentation d'un MDP par un graphe

de l'environnement. Par exemple, étant donné un état et une action, le modèle peut prédire le prochain état et la prochaine récompense qui en résulteraient. Les modèles sont utilisés pour la planification, c'est-à-dire tout moyen de décider d'un plan d'action en considérant les situations futures possibles avant qu'elles ne soient réellement vécues.[20] Les dynamiques de l'environnement se caractérisent par une distribution de probabilité de l'état successeur s' étant donné l'état actuel s et l'action prise a :

$$p(s'|s,a) = P[S_{t+1} = s'|S_t = s, A_t = a] \tag{3.2}$$

$$\sum_{s'} p(s'|s,a) = 1 \tag{3.3}$$

Il est important de constater que l'état s' ne dépend que de l'état s et de l'action a, et non pas des anciens états. Cette assomption s'appelle la propriété markovienne (Markov property) et est nécessaire pour qu'un MDP soit applicable à un problème :

$$P[S_{t+1}|S_t, A_t, S_{t-1}, A_{t-1}..., S_0, A_0] = P[S_{t+1}|S_t, A_t] \tag{3.4}$$

Beaucoup de problèmes respectent cette propriété. Et même si à première vue un problème ne semble pas le respecter, modifier l'espace des états permet de respecter cette propriété, même si cette modification entraînera son agrandissement. L'agent connaît donc $p(s'|s,a), \forall s', s, a$, aussi écrit $P^a_{s,s'}$.[39] P peut être vu comme un facteur de dimensions $(|A|*|S|*|S|)$, soit $|A|$ matrices de $(|S|*|S|)$ décrivant les probabilités de transitions entre chaque état pour chaque action.

3.2.2.2 La politique

elle désigne la manière dont l'agent d'apprentissage se comporte à un moment donné. En résumé, une politique est une correspondance entre les états perçus de l'environnement et les actions à entreprendre dans ces états. Dans certains cas, elle peut être une simple fonction ou une table de consultation, tandis que dans d'autres cas, elle peut impliquer un calcul approfondi. La politique est le cœur d'un agent d'apprentissage par renforcement dans le sens où elle seule suffit à déterminer le comportement. Les politiques peuvent être déterministes ou stochastiques [20]

1. Politique déterministe : Une politique déterministe π associe un état $s \in S$ a une action $a \in A$ $\pi : S \rightarrow A$
   $s \mapsto \pi(s)$
2. Politique Stochastique :Une politique stochastique ψ_θ, paramétrée par un ensemble de n paramètres $\theta \in 2R^n$ et un modèle ψ_θ, associe un état $s \in S$ S et les paramètres θ à une densité de probabilité sur l'espace des actions A :

$$\pi_\psi : SXAR^n \rightarrow R^+ avec \int_A \pi_\psi(a|s,\theta)da = 1$$
$$s, a, \theta \mapsto \pi_\psi(a|s,\theta)$$

3.2.2.3 Les fonctions de valeur

Elles indiquent ce qui est bon à long terme contrairement à la récompense qui indique ce qui est bon dans un sens immédiat. La valeur d'un état est le montant total de la récompense qu'un agent peut s'attendre à accumuler dans le futur, à partir de cet état. Elle indique la pertinence à long terme des états après avoir pris en compte les états qui sont susceptibles de suivre et les récompenses disponibles dans ces états. Par exemple, un état peut toujours donner une faible récompense immédiate, mais avoir une valeur élevée parce qu'il est régulièrement suivi d'autres états qui rapportent des récompenses élevées Ou l'inverse pourrait être vrai.[7] Les récompenses sont en quelque sorte primaires, tandis que les valeurs, en tant que prédictions des récompenses, sont secondaires. Sans récompenses, il n'y aurait pas de valeurs, et le seul but de l'estimation des valeurs est d'obtenir plus de récompenses.[22] Néanmoins, ce sont les valeurs qui nous préoccupent le plus lorsque nous prenons et évaluons des décisions. Les choix d'action sont faits sur la base de jugements de valeur. Nous recherchons des actions qui apportent des états de plus grande valeur, et non la récompense la plus élevée, car ces actions nous procurent la plus grande récompense à long terme. Malheureusement, il est beaucoup plus difficile de déterminer les valeurs que les récompenses. Les récompenses sont fondamentalement données directement par l'environnement, mais les valeurs doivent être estimées et ré-estimées à partir des séquences d'observations qu'un agent fait au cours de sa vie.[39] En fait, le composant le plus important de presque tous les algorithmes d'apprentissage par renforcement que nous considérons est une méthode d'estimation intelligente des valeurs. Le rôle central de l'estimation des valeurs est sans doute la chose la plus importante que nous ayons apprise sur l'apprentissage par renforcement au cours des dernières décennies.

3.2.2.4 Fonction de valeur d'état

Étant donnée une politique π, pour tout état $s \in S$, la fonction de valeur d'état V^π est définie comme suit :

$$
\begin{aligned}
V^{\pi}(s) &= E_{\pi}\{R_t | s_t = s\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right\} \\
&= E_{\pi}\left\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s\right\} \\
&= E_{\pi}\{r_{t+1} + \gamma V^{\pi}(s_{t+1}) | s_t = s\} \\
&= \sum_{a} \pi(s,a) \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^{\pi}(s')\right]
\end{aligned}
$$

3.2.2.5 Fonction action-valeur

Étant donnée une politique π, pour tout couple $(s,a) \in SXA$, la fonction de valeur état-action $Q^{\pi}(s,a)$ est définie comme suit :

$$
Q^{\pi}(s,a) = E_{\pi}\{R_t | s_t = s, a_t = a\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\right\}
$$

3.3 Résolution d'un Problème par l'apprentissage par renforcement

Les méthodes de résolution des problèmes d'apprentissage par renforcement qui utilisent des modèles et la planification sont appelées méthodes basées sur des modèles, par opposition aux méthodes plus simples sans modèle qui sont explicitement des méthodes d'essai-erreur, considérées comme presque l'opposé de la planification.[33]

Un agent d'apprentissage doit être capable de détecter l'état de son environnement dans une certaine mesure et doit être capable de prendre des mesures qui affectent cet état, Il doit également avoir un ou plusieurs objectifs liés à l'état de l'environnement.[10]

Les processus de décision de Markov sont destinés à inclure uniquement ces trois aspects| état, action et but| dans leurs formes les plus simples possibles sans en banaliser aucun. Toute méthode qui est bien adaptée à la résolution de tels problèmes, nous la considérons comme une méthode d'apprentissage par renforcement.[39]

L'un des défis qui se pose dans l'apprentissage par renforcement, et non dans d'autres types d'apprentissage, est le compromis entre l'exploration et l'exploitation. Pour obtenir beaucoup de récompenses, un agent d'apprentissage par renforcement doit préférer les actions qu'il a essayées dans le passé et qui se sont avérées efficaces pour produire une récompense.[22]

Mais pour découvrir ces actions, il doit essayer des actions qu'il n'a pas sélectionnées auparavant. L'agent doit exploiter ce qu'il a déjà expérimenté afin d'obtenir une récompense, mais il doit aussi explorer afin de faire de meilleures

sélections d'actions dans le futur. Le dilemme est que ni l'exploration ni l'exploitation ne peuvent être poursuivies de manière exclusive sans échouer à la tâche. L'agent doit essayer une variété d'actions et privilégier progressivement celles qui semblent les meilleures. Dans une tâche stochastique, chaque action doit être essayée de nombreuses fois pour obtenir une estimation fiable de sa récompense attendue.[34]

Le dilemme de l'exploration-exploitation a été étudié de manière intensive par les mathématiciens depuis plusieurs décennies, mais n'est toujours pas résolu. Pour l'instant, la question de l'équilibre entre exploration et exploitation ne se pose même pas dans l'apprentissage supervisé et non supervisé.[36]

L'apprentissage par renforcement adopte une approche commençant par un agent complet, interactif et à la recherche d'un but. Tous les agents d'apprentissage par renforcement ont des objectifs explicites, peuvent détecter des aspects de leur environnement et peuvent choisir des actions pour influencer leur environnement. De plus, on suppose généralement dès le départ que l'agent doit fonctionner malgré une incertitude importante sur l'environnement auquel il est confronté.[18]

En outre l'un des aspects les plus passionnants de l'apprentissage par renforcement moderne est son interaction profonde et utile avec d'autres disciplines scientifiques et d'ingénierie. Il fait partie d'une tendance qui dure depuis des décennies, interagit fortement avec la psychologie et les neurosciences, avec des avantages notables dans les deux sens. De toutes les formes de l'apprentissage automatique, l'apprentissage par renforcement est celui qui se rapproche le plus du type d'apprentissage que font les humains et d'autres animaux, et nombreux algorithmes de l'apprentissage par renforcement ont été inspirés à l'origine par des systèmes d'apprentissage biologiques[10]

La résolution d'un problème par l'apprentissage par renforcement consiste à résoudre un processus décisionnel de Markov

3.3.1 Résolution d'un MDP

La résolution d'un MDP revient à maximiser les récompenses, en optimisant la politique π, la fonction de valeur d'état V^{π} et la fonction action-valeur $Q^{\pi}(s, a)$, en utilisant les équations de Bellman.[27]

3.3.1.1 Notion de retour

Résoudre un MDP revient à avoir un comportement qui maximise les récompenses, aussi appelé retour G_t, qui est la somme de toutes les récompenses que l'agent reçoit à partir de l'instant t :

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T \tag{3.5}$$

Comme on peut le remarquer, cette définition de G_t part du principe que l'interaction agent-environnement se termine à l'instant T, ce qui n'est pas

toujours le cas. Cette définition pourrait être modifiée pour convenir à une interaction infinie, mais alors la variable aléatoire G_t serait infinie. C'est pourquoi la définition la plus courante de G_t est la suivante [14] :

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... \tag{3.6}$$

$$= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{3.7}$$

$\gamma \in [0; 1]$ et est un hyper paramètre appelé facteur de réduction (discount factor), qui détermine l'importance des récompenses les plus éloignées dans le temps. Ainsi, pour $\gamma = 0$, le retour G_t ne sera composé que de l'immédiate récompense R_{t+1}, ce qui fait que l'agent agira pour maximiser l'immédiate récompense seulement et oubliera les récompenses futures. L'agent est alors qualifié de myope. Au contraire, si $\gamma = 1$, l'agent sera hypermétrope, car les récompenses futures comptent autant que la récompense immédiate.[39]

En plus d'assurer que le retour G_t ne soit pas infini, l'introduction de la remise s'inspire du comportement humain et animal qui privilégie les récompenses immédiates.[27]

Il est également important de noter pour la suite que le retour G_t peut être décomposé en deux parties : la récompense immédiate R_{t+1} et le reste des récompenses futurs, G_{t+1},[22]réduit :

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = R_{t+1} + \gamma G_{t+1} \tag{3.8}$$

3.3.1.2 Les équations de Bellman

Les équations de Bellman sont des outils très utilisés dans l'apprentissage par renforcement, à la fois dans l'approche basée sur un modèle et l'approche sans modèle. Un des premiers résultats importants pour évaluer les fonctions de valeur provient de la programmation dynamique (Bellman, 1956). Les équations de Bellman vont servir à approximer les fonctions v_π et q_π.[14]

1 - Exprimer récursivement $v_\pi(s)$

Comme vu précédemment, $G_t = R_{t+1} + \gamma G_{t+1}$. Ainsi, on peut écrire :

$$v_\pi(s) = E_\pi[G_t|S_t = s] = E_\pi[R_{t+1} + \gamma G_t|S_t = s] = E_\pi[R_{t+1}|S_t = s] + \gamma E_\pi[G_{t+1}|S_t = s] \tag{3.9}$$

R_{t+1} correspond à l'état immédiat, et G_{t+1} à la value de l'état successeur de s.

Calculer $E_\pi[R_{t+1}|S_t = s]$ ainsi que $E_\pi[G_{t+1}|S_t = s]$ permettra donc de calculer $v_\pi(s)$. $E_\pi[R_{t+1}|S_t = s]$ représente la récompense immédiate moyenne que l'agent reçoit lorsqu'il sort de l'état s et suit π.

Comme montré dans le diagramme juste après , l'agent peut recevoir une récompense immédiate en sortant de l'état s et en effectuant l'action a. Chaque

récompense R_s^a a une probabilité $\pi(a|s)$ d'être reçu par l'agent. Ainsi, l'espérance (ou moyenne) de la récompense immédiate R_{t+1} de l'état s se calcule[23] :

$$E_\pi[R_{t+1}|S_t = s] = \sum_a \pi(a|s) R_s^a \tag{3.10}$$

$E_\pi[G_{t+1}|S_t = s]$ représente la récompense futur moyen que l'agent peut recevoir lorsqu'il est dans l'état s, et suit π

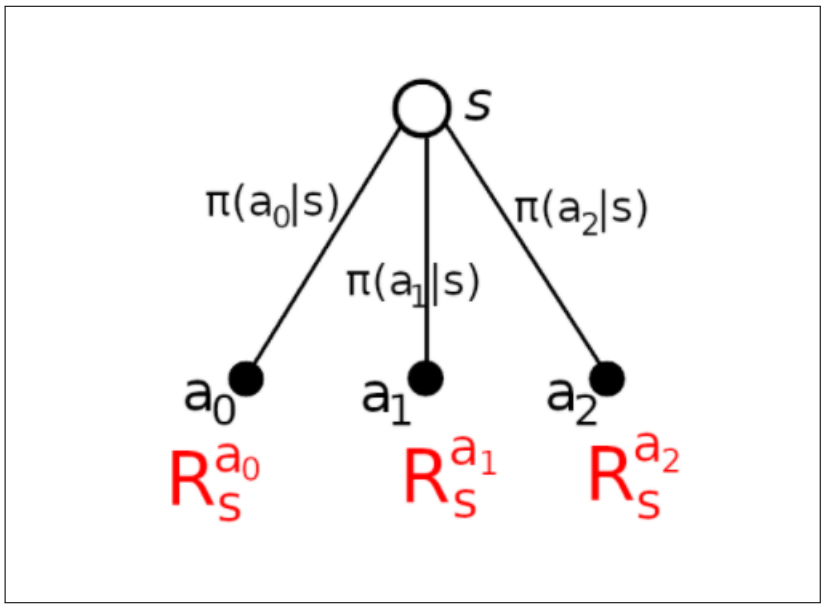

FIGURE 3.3 – diagramme de sauvegarde diagramme des récompenses immédiates

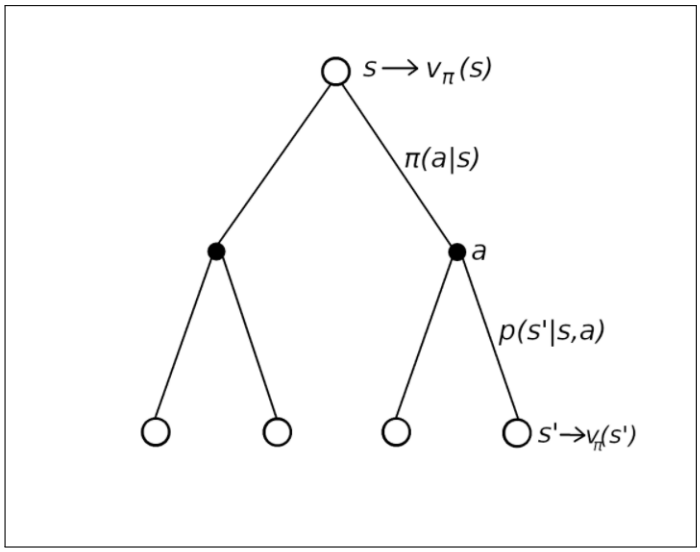

FIGURE 3.4 – diagramme de sauvegarde diagramme des récompenses futures

Le diagramme ci-dessus montre que, dans l'état s, l'agent peut avoir différent futur retour G_{t+1}, chacun associé à une probabilité d'arriver, tout comme

l'agent pouvait précédemment recevoir plusieurs récompenses immédiates R_{t+1}. La probabilité d'avoir un certain retour $v_\pi(s')$ est d'abord la probabilité $\pi(a|s)$ de prendre l'action a qui mène l'agent "à portée" de s' multipliée par la probabilité $p(s'|s,a)$ de transitionner, compte tenu de s et a, vers l'état s'[39]. Ainsi, la somme de ces probabilités avec leur valeur correspondante donne la définition suivante de $E_\pi[G_{t+1}]$:

$$E_\pi[G_{t+1}|S_t = s] = \sum_a \pi(a|s) \sum_{s'} p(s'|s,a) v_\pi(s') \tag{3.11}$$

On peut donc écrire :

$$\begin{aligned} v_\pi(s) &= E_\pi[R_{t+1}|S_t = s] + \gamma E_\pi[G_{t+1}|S_t = s] && (3.12)\\ &= \sum_a \pi(a|s) R_s^a + \gamma \sum_a \pi(a|s) \sum_{s'} p(s'|s,a) v_\pi(s') && (3.13)\\ &= \sum_a \pi(a|s)[R_s^a + \gamma \sum_{s'} p(s'|s,a) v_\pi(s')] && (3.14) \end{aligned}$$

2- Exprimer récursivement $q_\pi(s,a)$
Comme vu précédemment, $G_t = R_{t+1} + \gamma G_{t+1}$. ainsi, on peut écrire :

$$\begin{aligned} q_\pi(s,a) = E_\pi[G_t|S_t = s, A_t = a] &= E_\pi[R_{t+1} + \gamma G_t|S_t = s, A_t = a] && (3.15)\\ &= E[R_{t+1}|S_t = s, A_t = a] + \gamma E_\pi[G_{t+1}|S_t = s, A_t = a] && (3.16) \end{aligned}$$

Donc Calculer $E[R_{t+1}|S_t = s, A_t = a]$ ainsi que $E_\pi[G_{t+1}|S_t = s, A_t = a]$ permettra de calculer $q_\pi(s,a)$. $E[R_{t+1}|S_t = s, A_t = a]$ représente l'espérance de la récompense immédiate étant donné le courant état s et l'action prise a. Cette valeur est fournie à l'agent par l'environnement : c'est R_s^a. Aucun calcul n'est donc nécessaire, car l'agent sait qu'il va prendre l'action a avec une probabilité de 1 $E_\pi[G_{t+1}|S_t = s, A_t = a]$ représente la récompense futur moyen que l'agent peut recevoir lorsqu'il est dans l'état s, prend l'action a, et suit ensuite π[22]. Le raisonnement pour calculer $E_\pi[G_{t+1}|S_t = s, A_t = a]$ est similaire à celui utilisé avec $v_\pi(s)$, mais est même plus simple, car l'agent sait quelle action a il va prendre. Le diagramme montre donc que l'agent peut avoir différent futur retour G_{t+1} chacun associé à une probabilité d'arriver. La probabilité d'avoir un certain futur retour $v_\pi(s')$ est égale à la probabilité $p(s'|s,a)$ de transitionner, compte tenu de s et a, vers l'état s' [22]. Ainsi :

$$E_\pi[G_{t+1}|S_t = s, A_t = a] = 1 \sum_{s'} p(s'|s,a) v_\pi(s') \tag{3.17}$$

Ainsi :

$$\begin{aligned} q_\pi(s,a) = E[R_{t+1}|S_t = s|A_t = a] + \gamma E_\pi[G_{t+1}|S_t = s, A_t = a] && (3.18)\\ = R_s^a + \gamma \sum_{s'} p(s'|s,a) v_\pi(s') && (3.19) \end{aligned}$$

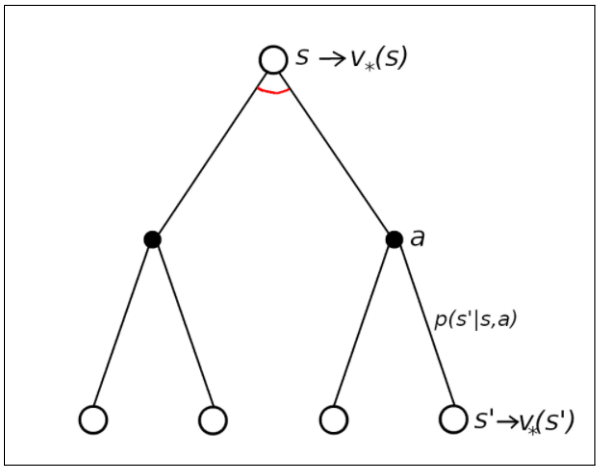

FIGURE 3.5 – diagramme de sauvegarde up diagramme exprimer récursivement $v_*(s)$

3.3.1.3 L'optimalité

π_* La politique optimale π_* est par définition la politique qui, pour tout état s et pour toute politique π :

$$v_{\pi_*}(s) \geq v_\pi(s), \quad \forall \pi, s \tag{3.20}$$

Il est très souvent possible qu'il existe une multitude de politiques optimale, voir même une infinité de politiques optimales stochastiques [39].

v_* : La fonction de valeur optimale de l'état v_* à la propriété suivante :

$$v_*(s) = v_{\pi_*}(s) = max_\pi v_\pi(s), \quad \forall s \tag{3.21}$$

Il n'existe qu'une seule valeur d'état optimale.

q_* : La fonction de valeur de l'état- action optimale q_* à la propriété suivante :

$$q_*(s,a) = q_{\pi_*}(s,a) = max_\pi q_\pi(s,a), \quad \forall s, a \tag{3.22}$$

Exprimer récursivement v_* et $q_*(s,a)$

Le trait rouge présent sur le diagramme signifie que $v_*(s)$ prendra la valeur la plus grande entre les deux branches. Cela se traduit sous forme d'équation ainsi :

$$v_*(s) = max_a[R_s^a + \gamma \sum_{s'} p(s'|s,a) v_*(s')] \tag{3.23}$$

Ainsi, si v_* est calculée, la valeur de tout état s sera la maximale, car $v_*(s)$ utilise l'action optimale de l'état s.

Le principe s'applique de la même manière avec $q_*(s,a)$:

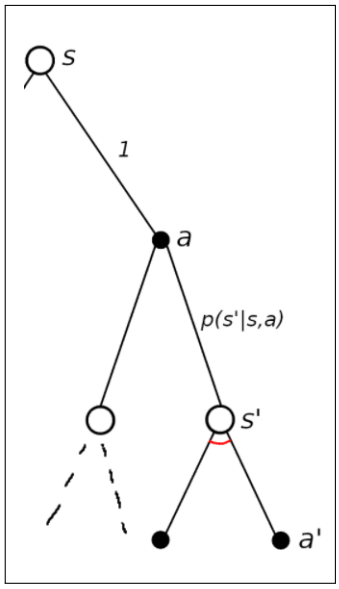

FIGURE 3.6 – diagramme de sauvegarde pour exprimer récursivement $q_*(s,a)$

$$q_*(s,a) = R_s^a + \gamma \sum_{s'} p(s'|s,a) v_*(s') \tag{3.24}$$

$$q_*(s,a) = R_s^a + \gamma \sum_{s'} p(s'|s,a) max_{a'} q_*(s',a') \tag{3.25}$$

Le max est ici pris lorsque l'agent prend l'action a' qui est l'action optimale dans l'état s'.

3.3.1.4 Tache de prédiction

La politique est fixe, l'objectif est de mesurer ses performances. Autrement dit, prédire la récompense totale attendue de tout état donné en assurant la fonction π (a | s)[18].

3.3.1.5 Tache de contrôle

La politique n'est pas fixe et l'objectif est de trouver la politique optimale. Autrement dit, trouver la politique π (a | s) qui maximise la récompense totale attendue de tout état donné.[18] Un algorithme de contrôle basé sur des fonctions d'évaluation fonctionne généralement en résolvant également le problème de prédiction, c'est-à-dire qu'il prédit les valeurs d'agir de différentes manières et ajuste la politique pour choisir les meilleures actions à chaque étape. En conséquence, la sortie des algorithmes basés sur la valeur est généralement une politique approximativement optimale et les récompenses futures attendues en suivant cette politique.[36] effectuer une tâche complète consiste à effectuer la tâche de prédiction et de contrôle

3.3.2 Exemple

La meilleure façon de comprendre l'apprentissage par renforcement est d'examiner certains des exemples et des applications possibles qui ont guidé son développement. - Un grand joueur d'échecs joue un coup. Le choix est informé à la fois par une planification anticipant réponses et contre-réponses possibles et par des jugements immédiats et intuitifs sur l'opportunité de positions et mouvements particuliers.

- Un robot mobile décide s'il doit entrer dans une nouvelle pièce à la recherche de plus de déchets à collecter ou commencer à essayer de retrouver son chemin vers sa station de recharge. Il prend sa décision en fonction du niveau de charge actuel de sa batterie et de la rapidité et de la facilité avec lesquelles il a pu trouver la station de recharge dans le passé.

- Voiture autonome, au départ, la voiture serpente, un peu comme un enfant qui fait ses premiers pas. À chaque fois qu'elle franchit les limites de sa voie, le chauffeur intervient pour corriger la trajectoire. L'algorithme comprend qu'il commet une erreur dès que le chauffeur le reprend et que la « récompense » consiste à éviter le plus longtemps possible cette intervention.

3.4 Classification des approches d'apprentissage par renforcement :

L'un des points de branchement les plus importants dans un algorithme RL est la question de savoir si l'agent a accès à (ou apprend) un modèle de l'environnement. Par modèle de l'environnement, nous entendons une fonction qui prédit les transitions d'état et les récompenses.

Le principal avantage d'avoir un modèle est qu'il permet à l'agent de planifier en anticipant, en voyant ce qui se passerait pour une gamme de choix possibles et en décidant explicitement entre ses options. Les agents peuvent ensuite condenser les résultats de la planification en une politique apprise. Un exemple particulièrement célèbre de cette approche est "AlphaZero". Lorsque cela fonctionne, cela peut entraîner une amélioration substantielle de l'efficacité de l'échantillon par rapport aux méthodes qui n'ont pas de modèle.

Le principal inconvénient est qu'un modèle de vérité terrain de l'environnement n'est généralement pas disponible pour l'agent. Si un agent souhaite utiliser un modèle dans ce cas, il doit apprendre le modèle uniquement par expérience, ce qui crée plusieurs défis. Le plus grand défi est que le biais dans le modèle peut être exploité par l'agent, résultant en un agent qui fonctionne bien par rapport au modèle appris, mais se comporte de manière sous-optimale (ou super terriblement) dans l'environnement réel. L'apprentissage des modèles est fondamentalement difficile, donc même un effort intense - être prêt à consacrer beaucoup de temps et à calculer - peut ne pas porter ses fruits.[7]

Les algorithmes qui utilisent un modèle sont appelés méthodes basées sur un modèle, et ceux qui ne le sont pas sont appelés sans modèle. Bien que les méthodes sans modèle renoncent aux gains potentiels d'efficacité de l'échan-

tillon grâce à l'utilisation d'un modèle, elles ont tendance à être plus faciles à mettre en œuvre et à régler. Au moment de la rédaction de cette introduction (septembre 2018), les méthodes sans modèle sont plus populaires et ont été plus largement développées et testées que les méthodes basées sur des modèles.[11]

3.4.1 Apprentissage par renforcement basé sur un modèle :

Dans un environnement RL basé sur un modèle, la politique est basée sur l'utilisation d'un modèle d'apprentissage automatique. . Sachant pertinemment que la politique est un algorithme qui décide de l'action d'un agent. Dans ce cas, lorsqu'un environnement ou un système de RL utilise des modèles d'apprentissage automatique tels que la forêt, le gradient boost, les réseaux neuronaux et autres, un tel système de RL est basé sur un modèle.[19] cette Approche Basé sur l'utilisation de l'expérience pour créer un modèle interne des transitions et des résultats dans l'environnement, le choix de la bonne action est fait par la rechercher ou la planification dans le modèle interne construit, Il tente à modéliser l'environnement en choisissant la politique optimale en fonction du modèle appris.[13] - Il existe deux approches principales pour représenter et former des agents : a- Optimisation des politiques : c'est une approche centrée sur la politique qui est la fonction qui fait correspondre l'état de l'agent à sa prochaine action. Qui considèrent l'apprentissage par renforcement comme un problème d'optimisation numérique et optimise la récompense attendue en fonction des paramètres de la politique.[39] Il existe plusieurs algorithmes dérivés de cette approche tels que : politique Gradient, A2C /A3C, PPO, etc. b- Q-Learning : une approche de l'apprentissage par renforcement, qui cherche à trouver la meilleure action à entreprendre compte tenu de l'état actuel. Son but est de mettre a jour la fonction valeur basée sur une équation (en particulier l'équation de Bellman) Cela signifie qu'elle apprend la valeur de la politique optimale indépendamment des actions de l'agent. Il existe plusieurs algorithmes dérivés de cette approche tels que : DQN, C51, QR-DQN, etc.

3.4.2 Apprentissage par renforcement sans modèle

C'est un système qui n'a pas de politique basée sur l'utilisation de modèles d'apprentissage automatique ; sa politique est guidée par l'utilisation d'algorithmes non-ML. [19] cette approche qui estime la politique optimale sans utiliser ni estimer la dynamique (fonctions de transition et de récompense) de l'environnement, basé sur l'expérience, c'est-à-dire à partir de l'interaction entre l'agent et l'environnement, elle estime directement une fonction de valeur. À partir de cette fonction de valeur, une politique peut alors être dérivée.[13] Il existe deux approches principales pour représenter et former des agents :

1. Apprendre le modèle : une approche de l'apprentissage automatique qui consiste à exécuter une politique de base, comme une politique aléatoire ou toute politique éduquée, et observe la trajectoire. Ensuite, ajuster un modèle en utilisant ces données échantillonnées. Il existe plusieurs

algorithmes dérivés de cette approche tels que : World Models, I2A, MBMF, MBVE.

2. Connaître le modèle : une approche qui permet à l'agent de planifier en anticipant, en voyant ce qui se passerait pour une gamme de choix possibles et en décidant explicitement entre ses options. Les agents peuvent ensuite condenser les résultats de la planification en une politique apprise.[10] Un algorithme particulièrement célèbre de cette approche est AlphaZéro

3.4.3 Comparaison entre les deux approches

La comparaison entre les deux approches vues juste avant est montré dans le tableau suivant :

	Avantages	Inconvénients
Basé sur un modèle	○ Petit nombre d'interactions entre l'agent et l'environnement. ○ Convergence plus rapide vers la solution optimale	○ Dépend du model de transition ○ La précision du modèle a un impact important sur les tâches d'apprentissage
Sans model	○ Pas besoin de connaissances préalables sur les transitions ○ Facile à mettre en œuvre	○ Convergence lente vers la solution optimale ○ Risque élevé d'endommagement

FIGURE 3.7 – Tableau de comparaison entre les deux approches

3.4.4 Exemple d'algorithme d'apprentissage par renforcement

3.4.4.1 Algorithme "Policy Evaluation"

L'algorithme de policy evaluation consiste à calculer la fonction de valeur de l'état v_π d'une politique donnée π pour chaque état s. π étant la politique qui décrit le comportement de l'agent dans son environnement , et v_π la fonction qui associe a chaque état le retour de récompense espéré, en suivant la politique π en étant dans l'état sPlus formellement, il s'agit d'une application itérative de l'équation d'espérance de Bellman :[18]

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s'} p(s'|s,a)[R^a_{ss'} + y v_\pi(s')] \tag{3.26}$$

La mise à jour de la fonction de valeur de l'état $V(s)$ se fait de façon asynchrone de la manière suivante :

$$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} p(s'|s,a)[R^a_{ss'} + y V(s')] \tag{3.27}$$

Voici le pseudo code de l'algorithme :

Input: the policy π to be evaluated
Initialize $V(s)$ arbitrarily (e.g., $V(s) = 0, \forall s \in \mathcal{S}^+$)
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    $A \leftarrow$ action given by π for S
    Take action A; observe reward, R, and next state, S'
    $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
    $S \leftarrow S'$
  until S is terminal

FIGURE 3.8 – Pseudo Code Algorithme' Policy evaluation'

3.4.4.2 Algorithme Policy Improvement

L'algorithme de policy improvement consiste à améliorer une politique π par rapport à sa fonction de valeur état-action Q, pour chaque état s. Il est souvent dit de cette amélioration que la politique agit "avec gourmandise" par rapport à la fonction de valeur état-action Q.[36]

L'amélioration assigne à π la meilleure action a pour chaque état s. Chaque action est évaluée grâce à la fonction de valeur de l'état V, connue. (calculée par l'algorithme de policy évaluation)

Il peut ainsi être décrit comme suit : pour une politique π déterministe :

$$\pi(s) \leftarrow argmax_a \, Q(s,a) \quad \forall s \tag{3.28}$$

$$\pi(s) \leftarrow argmax_a \, [\sum_{s'} p(s'|s,a)[R^a_{s,s'} + \gamma V(s')]] \quad \forall s \tag{3.29}$$

Ainsi, pour chaque état s, l'algorithme devra calculer la valeur $\sum_{s'} p(s'|s,a)[R^a_{s,s'} + \gamma v_\pi(s')]$ de chaque action a, et assignera à la politique π l'action qui a la plus grande valeur.

Algorithme "Policy iteration"

L'algorithme de Policy Iteration reprend une structure fondamentale en apprentissage par renforcement , l'enchaînement de la tâche de prédiction et celle de contrôle. La tâche de prédiction est accomplie par l'algorithme de "Policy Évaluation", il s'agit de calculer v_π pour π donnée. La tâche de contrôle est effectuée par l'algorithme de "Policy Improvement", et consiste à déduire de v_π une politique π_* meilleure que π.[39] voici le pseudocode de l'algorithme :

3.4.4.3 Algorithme Q-learning

L'algorithme Q-learning consiste à mettre à jour de manière itérative les valeurs Q pour chaque paire état-action en utilisant l'équation d'optimalité de Bellman jusqu'à ce que la fonction Q fonction de valeur action-valeur converge

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
      $temp \leftarrow \pi(s)$
      $\pi(s) \leftarrow \arg\max_a \sum_{s'} p(s'|s,a)\Big[r(s,a,s') + \gamma v(s')\Big]$
      If $temp \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return v and π; else go to 2

FIGURE 3.9 – Pseudo Code Algorithme' Policy improvement

1. Initialization
   $v(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
      $\Delta \leftarrow 0$
      For each $s \in \mathcal{S}$:
         $temp \leftarrow v(s)$
         $v(s) \leftarrow \sum_{s'} p(s'|s,\pi(s))\Big[r(s,\pi(s),s') + \gamma v(s')\Big]$
         $\Delta \leftarrow \max(\Delta, |temp - v(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
      $temp \leftarrow \pi(s)$
      $\pi(s) \leftarrow \arg\max_a \sum_{s'} p(s'|s,a)\Big[r(s,a,s') + \gamma v(s')\Big]$
      If $temp \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return v and π; else go to 2

FIGURE 3.10 – Pseudo Code Algorithme' Policy iteration

vers la fonction Q optimale, q, ce processus est appelé itération des valeurs. Pour que la valeur de Q converge éventuellement vers une valeur de Q optimale Q^*, il faut que, pour la paire état-action donnée, la valeur de la Q soit aussi proche que possible du côté droit de l'équation d'optimalité de Bellman. Pour cela, la perte entre la valeur Q et la valeur Q^* pour la paire état-action donnée sera comparée itérativement, puis à chaque fois que nous rencontrerons la même paire état-action, nous mettrons à jour la valeur Q, encore et encore, pour réduire la perte[14]. La perte peut être donnée comme suit :

$$Q(s_t, a_t) < -Q(s_t, a_t) + \alpha[r_t + 1 + \gamma max_a Q(s_t + 1, a) - Q(s_t, a_t)]$$

Voici le pseudocode de l'algorithme Q_learning

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Choose a from s using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g., ε-greedy)
    $Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma Q(s',a') - Q(s,a)\big]$
    $s \leftarrow s'$; $a \leftarrow a'$;
  until s is terminal

FIGURE 3.11 – Pseudo Code l'algorithme Q learning

3.5 Les outils d'apprentissage par renforcement

3.5.1 Spinning up

Spinning up est une ressource éducative qui permet de devenir un praticien qualifié dans l'apprentissage par renforcement, il se compose d'exemples clairs de code RL, d'exercices pédagogiques, de documentation et de didacticiels [5]. Exemple : L'un des meilleurs moyens de se faire une idée de la RL est d'exécuter les algorithmes et de voir comment ils fonctionnent sur différentes tâches. La bibliothèque de code Spinning Up facilite les expériences à petite échelle (locales), il existe deux façons de les exécuter : soit à partir de la ligne de commande, soit via des appels de fonction dans des scripts.

Lancement depuis la ligne de commande Spinning Up est livré avec spinup/run.py, un outil pratique qui permet de lancer facilement n'importe quel algorithme avec n'importe quel choix d'hyper paramètres à partir de la ligne de commande. Il sert également de wrapper mince sur les utilitaires pour regarder les politiques entraînées et le traçage. [1] La manière standard d'exécuter un algorithme Spinning Up à partir de la ligne de commande est : - python - m spinup . exécuter [algo nom] [expérience drapeaux]-

Exemple :

— Graines aléatoires. Toutes les expériences ont été effectuées pour 10 graines aléatoires chacune. Les graphiques montrent la moyenne (ligne continue) et le dev std (ombré) des performances par rapport aux graines aléatoires au cours de la formation.

— Métrique de performance. Les performances des algorithmes sur la politique sont mesurées comme le retour de trajectoire moyen à travers le lot collecté à chaque époque. Les performances des algorithmes hors politique sont mesurées une fois toutes les 10 000 étapes en exécutant la politique déterministe (ou, dans le cas de SAC, la politique moyenne) sans bruit d'action pour dix trajectoires, et en signalant le retour moyen

sur ces trajectoires de test.

— Architectures de réseau. Les algorithmes sur la politique utilisent des réseaux de taille (64, 32) avec des unités tanh pour la politique et la fonction de valeur. Les algorithmes hors politique utilisent des réseaux de taille (256, 256) avec des unités relues.

— Taille du lot. Les algorithmes sur la politique ont collecté 4000 étapes d'interaction agent-environnement par mise à jour par lot. Les algorithmes hors politique utilisaient des mini-lots de taille 100 à chaque étape de descente de gradient.

Voici la visualisation dans les figures suivantes :

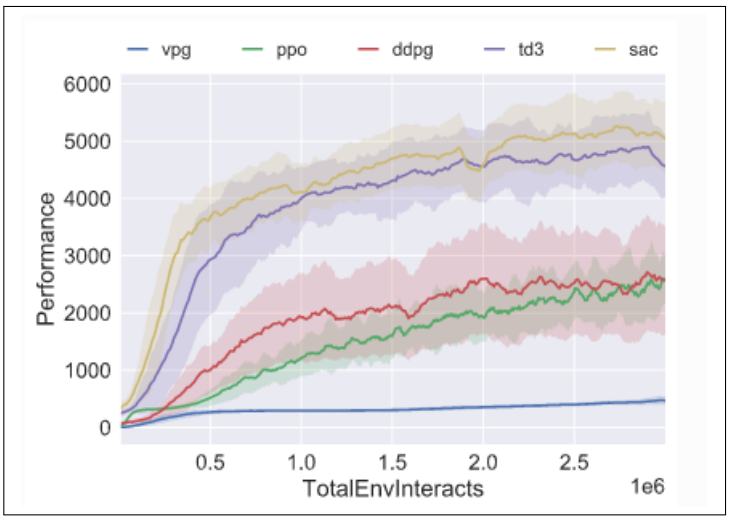

FIGURE 3.12 – Benchmark temporel 3M pour Walker2d-v3 à l'aide des implémentations PyTorch .

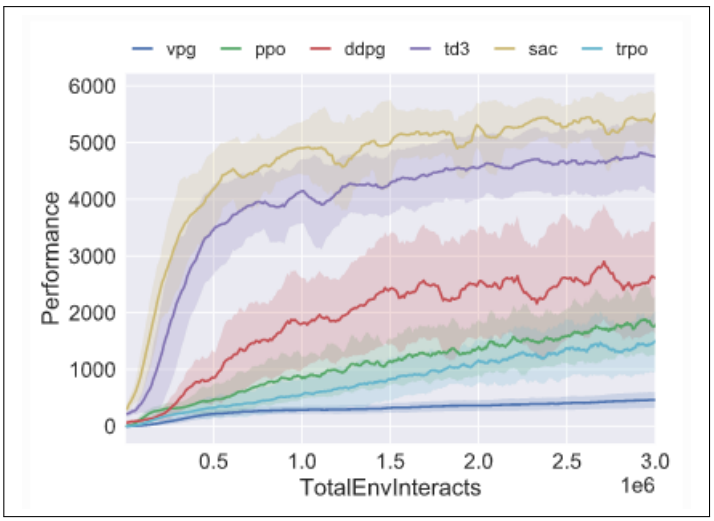

FIGURE 3.13 – Benchmark temporel 3M pour Walker2d-v3 à l'aide des implémentations Tensorflow .

3.5.2 Gym

Gym est une boîte à outils pour le développement et la comparaison d'algorithmes d'apprentissage par renforcement, La bibliothèque gym est une collection d'environnements de test que vous pouvez utiliser pour mettre au point vos algorithmes d'apprentissage par renforcement. Ces environnements ont une interface partagée, vous permettant d'écrire des algorithmes généraux.[2] Exemple :

— CartPole-v1 Un poteau est attaché par un joint non actionné à un chariot, qui se déplace le long d'une piste sans friction. Le système est contrôlé en appliquant une force de +1 ou -1 au chariot. Le pendule démarre debout et le but est de l'empêcher de basculer. Une récompense de +1 est fournie pour chaque pas de temps pendant lequel le poteau reste debout. L'épisode se termine lorsque le poteau est à plus de 15 degrés de la verticale ou que le chariot se déplace à plus de 2,4 unités du centre. L'environnent se présente comme suit :

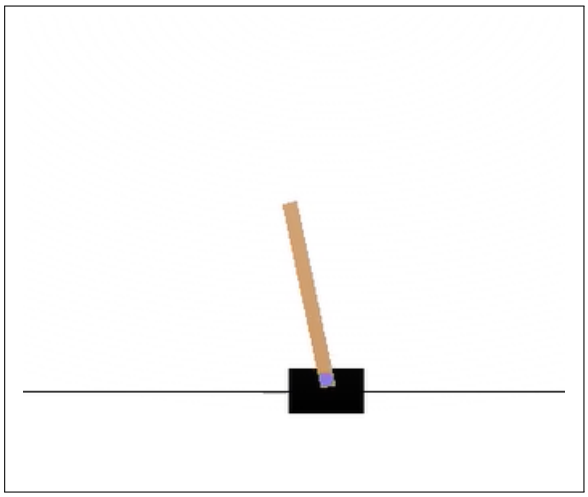

FIGURE 3.14 – Visuel de l'environnement CartPole-v1

— CrazyClimber-v0 jeu Atari 2600 CrazyClimber. Dans cet environnement, l'observation est une image RVB de l'écran, qui est un tableau de forme (210, 160, 3) Chaque action est répétée pendant une durée de k cadres, où k est uniformément échantillonné à partir de {2, 3, 4}.

3.5.3 RLlib

Rllib est une bibliothèque open-source pour l'apprentissage par renforcement qui offre à la fois une grande évolutivité et une API unifiée pour une variété d'applications. Rllib supporte nativement TensorFlow, TensorFlow Eager et PyTorch [3] Exemple :

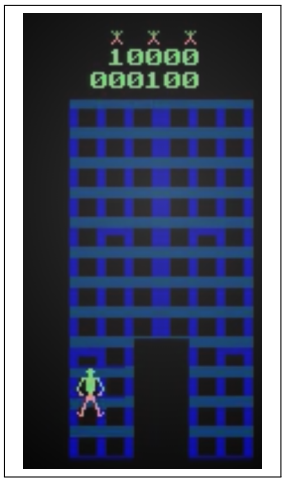

FIGURE 3.15 – Visuel de l'environnement CrazyClimber-v0

```
from ray import tune
from ray.rllib.agents.ppo
import PPOTrainer
tune.run(PPOTrainer, config={"env": "CartPole-v0"})  # "log_level": "INFO"
          # "framework": "tfe" for tf-eager,
          # "framework": "torch" for PyTorch
```

FIGURE 3.16 – Code d'entraînement en utilisant RLlib

3.5.4 Isaac Gym

Isaac Gym est un Environnement de simulation physique et de calcul de récompense de NVIDIA pour la recherche d'apprentissage par renforcement, il fournit une API de base pour créer et peupler une scène avec des robots et des objets, prenant en charge le chargement de données à partir des formats de fichiers URDF et MJCF. Chaque environnement est dupliqué autant de fois que nécessaire et peut être simulé simultanément sans interaction avec d'autres environnements.[4] Exemple :

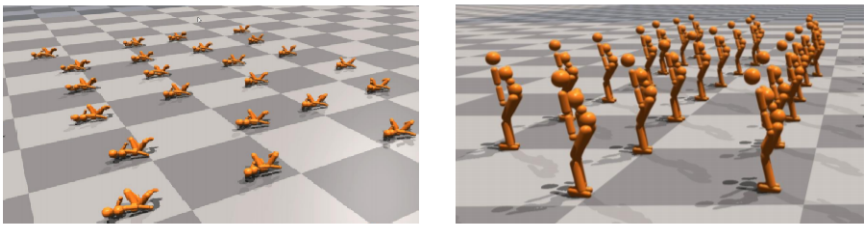

FIGURE 3.17 – exemple de mouvement avec "Isaac Gym

Chapitre 4

Implémentation

4.1 Définition du problème

L'intégration des systèmes cyber physiques dans la vie quotidienne a de nombreux avantages pour l'être humain, mais il y a aussi le souci des risques que ce système peut engendrer, et qui peuvent nuire au système ou à l'être humain. Toutes les variantes de RL impliquent un apprentissage par essai et erreurs, l'agent prend des actions aléatoires jusqu'à ce qu'il apprenne quels comportements rapportent le plus de récompenses. Ceci est généralement problématique si l'apprentissage doit avoir lieu dans le monde réel, où les erreurs peuvent être coûteuses ou mortelles. Étant donné qu'un agent embarqué doit tester ou explorer une grande variété d'actions avant d'apprendre quels comportements adopter et lesquels éviter, ce problème est appelé exploration non sécurisée. À titre d'exemple, il serait plutôt indésirable que l'agent expérimente en effectuant des actions dangereuses pour son environnement ou pour lui-même pour apprendre que de telles actions reçoivent de lourdes pénalités. [6]. Dans cet article, nous considérons le problème de sécurité des agents embarqués en utilisant l'approche de l'exploration sûre dans l'apprentissage par renforcement. Alors que l'apprentissage par renforcement est bien adapté aux domaines avec des dynamiques de transition complexes et des espaces état-action de grande dimension, un défi supplémentaire est posé par le besoin d'une exploration sûre et efficace. Les techniques d'exploration traditionnelles ne sont pas particulièrement utiles pour résoudre des tâches dangereuses, où le processus d'essais et d'erreurs peut conduire à la sélection d'actions dont l'exécution dans certains états peut endommager le système d'apprentissage (ou tout autre système).[15] Par conséquent, lorsqu'un agent embarqués commence une interaction avec un espace état-action dangereux et de grande dimension, une question importante se pose ; à savoir, celui de comment éviter (ou du moins minimiser) les dommages causés par l'exploration de l'espace état-action. Nous introduisons la méthode d'exploration sûre qui améliore en toute sécurité les comportements sous-optimaux bien que robustes pour les tâches de contrôle d'état et d'action continues et qui apprennent efficacement de l'expérience acquise de l'environnement. Nous évaluons la méthode proposée avec deux solutions différentes dans un exemple d'agent qui monte un escalier à N marches.

4.2 Formulation

L'apprentissage par renforcement sûr peut être défini comme le processus d'apprentissage de politiques qui maximisent l'espérance de rendement dans des problèmes où il est important d'assurer une performance raisonnable du système et de respecter les contraintes de sécurité pendant les processus d'apprentissage. Pour résoudre le problème de sûreté et de sécurité de l'agent, nous allons utiliser un exemple qui consiste à contrôler le déplacement d'un agent dans un escalier. L'agent peut monter les escaliers en marchant ou en sautant, mais il y a un risque d'endommagement de l'agent s'il saute un grand nombre de marches. La modélisation de ce problème a été effectuée en utilisant l'apprentissage par renforcement basé sur un modèle.[16] - la solution consiste à éviter que l'agent effectue des actions qui mène à son endommagement ou a un problème dans l'environnement, Afin de pouvoir démontrer nos deux solutions nous avons imaginé qu'un agent qui a comme objectif de monter toutes les marches d'un escalier en utilisant l'action marcher pour se déplacer d'une marche à une autre ou sauter pour sauter plusieurs marches, mais il y a un risque d'endommagement de l'agent s'il saute un grand nombre de marches. Le but de cet exemple est d'éviter l'endommagement de l'agent tout en lui permettant d'apprendre de son environnement.

1. **Environnement** :
   L'environnement dans lequel l'agent vas être entraîné est un escalier à N marches, dans notre cas les dimensions des marches ne nous intéressent pas, car on traite le problème en général pour simuler la méthode utilisée.

2. **Agent** :
   l'agent est l'entité qui va interagir avec l'environnement en utilisant les deux approches d'exploration et d'exploitation, mais tout en assurant sa sécurité.

3. **Action** :
   L'agent pourra effectuer deux actions : 1- marcher :- elle consiste à déplacer l'agent de la marche m a la marche $m + 1$ 2- Sauter : - elle consiste à déplacer l'agent de la marche m a la marche (m+le nombre de marches sauté)

4. **Récompense :**
   Chaque marche m'a une récompense de m donc la récompense totale est la somme des récompenses de chaque étage .

5. **Dynamique de transition $T(s_t{+}_1)|(s_t, a_t)$:**
   Elle permet de générer les transitions d'un agent de l'état s_ta l'état $s_t{+}_1$ en effectuant l'action a_t

4.3 développement des solutions

4.3.1 Description des outils utilisés

Les Solutions seront développées en utilisant les outils suivants :

1. Le langage python : Python est un langage de programmation puissant et facile à apprendre. Il possède des structures de données de haut niveau efficaces et une approche simple, mais efficace de la programmation orientée objet. La syntaxe élégante et le typage dynamique de Python, ainsi que sa nature interprétée, en font un langage idéal pour les scripts et le développement rapide d'applications dans de nombreux domaines sur la plupart des plates-formes.[26]
2. Les Bibliothèques utilisé :
   — Numpy : Fournit une interface pour stocker et effectuer des opérations sur les données. D'une certaine manière, les tableaux Numpy sont comme les listes en Python, mais Numpy permet de rendre les opérations beaucoup plus efficaces, surtout sur les tableaux de large taille. Les tableaux Numpy sont au cœur de presque tout l'écosystème de data science en Python.[29]
   — Matplotlib : est une bibliothèque complète permettant de créer des visualisations statiques, animées et interactives en Python.[35]
   — Pandas : fournit des structures de données puissantes et simples à utiliser, ainsi que les moyens d'opérer rapidement des opérations sur ces structures.[32]
   — Seaborn c'est une bibliothèque de visualisation de données Python basée sur matplotlib. Il fournit une interface de haut niveau pour dessiner des graphiques statistiques attrayants et informatifs.[37]
   — NetworkX est un package Python pour la création, la manipulation et l'étude de la structure, de la dynamique et des fonctions de réseaux complexes.il fournit des outils pour l'étude de la structure et de la dynamique des réseaux sociaux, biologiques et d'infrastructure, une interface de programmation standard et une implémentation graphique adaptée à de nombreuses applications, un environnement de développement rapide pour des projets collaboratifs et multidisciplinaires.[17]

4.3.2 Première approche

- la solution consiste à éviter que l'agent effectue des actions qui mène à son endommagement ou a un problème dans l'environnement, Afin de pouvoir démontrer nos deux solutions nous avons imaginé qu'un agent qui a comme objectif de monter toutes les marches en utilisant l'action marcher pour se déplacer d'une marche à une autre ou sauter pour sauter plusieurs marches mais afin d'éviter son endommagement l'agent ne peut sauter que trois marche au maximum. Ceci est un choix inspiré de la capacité d'un être humain et peut varier d'un agent à un autre et d'un environnement à un autre.

4.3.2.1 Implémentation de l'environnement

Un agent interagit avec son environnement via un processus de décision de Markov, ou MDP, (S, A, T, R, γ). À chaque étape t, l'agent observe un état $s \in S$ puis choisit une action $a \in A$ en fonction de sa politique π. L'environnement passe ensuite à l'état $s_{t+1} \in S$ selon la dynamique de transition $T(s_t + 1|s_t, a_t)$ et génère une récompense r_t avec une espérance $R_(s, a)$. Ce cycle se poursuit jusqu'à ce que chaque épisode se termine. L'environnement a été implémenté dans une classe appelée "EscalGridWorld", en ayant comme attribut la taille de l'escalier, le nombre de marches S, le nombre d'actions A que le l'agent peut effectuer et la position de la Marche finale. L'environnement est considéré comme un tableau dont la taille est égale au nombre de marches de l'escalier et chaque case du tableau représente une marche. - l'environnement propose deux actions possibles :marcher et sauter. Dans cette classe, nous implémentons la table des transitions qui décrit la dynamique de transition $T(s_t + 1|s_t, a_t)$, est le problème de saut de plus de trois marches, sera résolu en permettant à l'agent de choisir aléatoirement le nombre de marches à sauter, mais ce choix seras limité dans un intervalle de 0 à 4. Donc l'agent ne pourra pas se mettre en danger et là, le problème d'endommagement sera résolu. La figure suivante montre le code qui a permis de développer l'environnement.

```python
import numpy as np
import random
class EscalGridWorld():
    def __init__(self, size=20):
        self.size = size
        self.nS = size
        self.nA = 2
        self.MAX_X = size-1

        P = {}

        for s in range(self.nS):
            dynamics_s = {}
            reward=s
            for a in range(self.nA):
                s_prime_list = []
                p=0.5

                if(a == 0):
                    s_prime = min(self.MAX_X, s+1)


                else:
                    s_prime = min(self.MAX_X, s+random.randint(2,4))


                if(s_prime == self.MAX_X):

                    done = True
                else:
                    done = False

                s_prime_list.append((p, s_prime, reward, done))
                dynamics_s.update({a:s_prime_list})

            P.update({s: dynamics_s})

        self.P = P
```

FIGURE 4.1 – Code de développement de la classe "EscalGridWorld"

4.3.2.2 Implémentation de l'apprentissage de l'agent

L'apprentissage de l'agent sera implémenté dans une classe qui développe un ensemble de méthodes qui permettent de calculer la politique optimale afin que l'agent puisse interagir avec son environnement en toute sécurité et maximise ces récompenses . Algorithme d'apprentissage Value iteration Nous Avons utilisé l'algorithme Value Iteration (Puterman, 1994), issu de la programmation dynamique, est l'un des algorithmes standards d'apprentissage par renforcement, il consiste à calculer la fonction de valeur d'état optimale en améliorant de manière itérative l'estimation de $V(s)$. Ici, nous allons implémenter une itération de valeur qui calcule une politique optimale pour l'agent en utilisant l'équation de Bellman qui peut être définie comme une équation récursive qui représente la valeur d'un état en termes de valeurs d'états successeurs, plus toutes les récompenses gagnées entre les deux.[12] On commence par attribuer une valeur initiale de 0 à chaque état, puis utiliser l'équation de Bellman pour trouver les valeurs optimales pour chaque état et l'action qui le produit. La valeur optimale est atteinte en un nombre d'itérations fini.

Hyperparamètres

1. Gamma :
   Le facteur de remise - l'hyper paramètre qui peut être réglé pour mettre l'accent sur la récompense à long terme ou faire en sorte que le modèle choisisse la meilleure solution à court terme. 0,95 était une estimation prudente pour cela après avoir considéré le compromis entre la performance et le succès.
2. Thêta :
   Le seuil de convergence de $v(_s)$ l'augmentation de thêta améliorerait le temps d'exécution et nuirait au succès de l'algorithme. Cela a été défini après avoir fait le compromis entre la performance et le succès.

Le pseudo-code suivant exprime l'algorithme value iteration utilisée :

Initialize V(s) to arbitrary values
Repeat until V(s) converge
  For all states
    For all actions
      $Q(s,a) \leftarrow \sum_s P^a_{ss'}(r(s,a) + \gamma V(s'))$
    $V(s) \leftarrow \max_a Q(s,a)$

FIGURE 4.2 – pseudo-code de l'algorithme value iteration

À fin de pouvoir calculer la fonction valeur de l'état, on calcule la fonction de valeur état, action $Q_{s,a}$ en utilisant l'équation d'optimalité de Bellman équivalente a : $Q(_{s,a}) = prob'_s * (rcompense * \gamma + v[s'])$

la figure suivante exprime le code de la fonction $q(_{s,a})$ qui calcule la fonction de valeur état-action :

```
def compute_q_value_for_s_a(env,V ,s ,a , gamma):
    q=0
    for (p_sPrime, sPrime , r_ss_a , done) in env.P[s][a]:
        q+= p_sPrime*(r_ss_a+gamma * V[sPrime])

    return q
```

FIGURE 4.3 – Code de la fonction de valeur état-action

La fonction décrite juste avant sera utilisée pour trouver la valeur de récompense maximale que l'agent peut gagner, la valeur optimale est atteinte en un nombre d'itérations fini. On stoppe donc l'algorithme lorsque la différence entre deux valeurs successives devient inférieure à un certain seuil θ, le code suivant permet de calculer ces valeurs optimales :

```
while True:
    iteration+=1
    delta=0
    for s in range (env.nS):
        q_s=np.zeros([env.nA,1])
        for a in range(env.nA):

            q_s[a]=compute_q_value_for_s_a(env,V, s ,a ,gamma)
        newV=np.max(q_s)
        delta=max(delta, np.abs(newV - V[s]))
        V[s]=newV
    plot_visualisation_V(env,V)

    if(delta< theta):

        print("terminer apres " + str(iteration) + " iteration")

        break
df=pd.DataFrame.from_dict(env.P,orient='index')
print(df)
pi=improved_policy(env,pi,V ,gamma)
print('V optimal :   \n',V)
print('policy optimal :  \n',pi)
```

FIGURE 4.4 – Code de calcul de la valeur d'état maximale

Après avoir calculé la valeur d'état optimale $V*$ nous allons utiliser l'algorithme "policy improvement" qui va déduire de $V*$ la politique optimale, ainsi il attribue à chaque état l'action qui maximise $q(_{s,a})$ en utilisant la fonction suivante : $\pi*(_s) = argmax(Q(s,a)\forall s$ l'algorithme "policy improvement" a été implémenter dans une méthode intitulée 'improved policy' , le code est dans la figure suivante :

3. Policy Improvement
policy-stable $\leftarrow$ *true*
For each $s \in \mathcal{S}$:
$temp \leftarrow \pi(s)$
$\pi(s) \leftarrow \arg\max_a \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma v(s')\right]$
If $temp \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
If *policy-stable*, then stop and return v and π; else go to 2

FIGURE 4.5 – Pseudo-code 'policy improvement'

La figure suivante décrit le développement de la fonction

```
def improved_policy(env , pi , V , gamma):
    for s in range(env.nS):
        q_s = np.zeros([env.nA , 1])
        for a in range (env.nA):
            q_s[a]= compute_q_value_for_s_a(env, V , s ,a , gamma)
            best_a = np.argmax(q_s)
            pi[s]= [best_a]
    plot_visualisation_P(env,pi)
    return pi
```

FIGURE 4.6 – Code fonction 'policy improvement'

Mais il faut noter que le fait d'effectuer plusieurs itérations sur tous les états. Ce n'est pas un problème pour notre environnement, car le but de notre système est d'assurer la sécurité de l'agent, mais dans un problème général d'apprentissage par renforcement, ça pourrait causer des problèmes.

4.3.2.3 Simulation

À chaque exécution, l'environnement génère une table de transition qui permet de retourner toutes les transitions possibles avec tous les éléments nécessaires pour calculer la fonction de valeur , la figure suivante montre un exemple de table de transitions.

Les marches	Les actions							
	Marcher				Sauter =1			
	Probabilité	s'	récompense	Terminer	Probabilité	s'	récompense	Terminer
0	0,5	1	0	False	0,5	2	0	False
1	0,5	2	1	False	0,5	3	1	False
2	0,5	3	2	False	0,5	6	2	False
3	0,5	4	3	False	0,5	6	3	False
4	0,5	5	4	False	0,5	6	4	False
5	0,5	6	5	False	0,5	8	5	False
6	0,5	7	6	False	0,5	9	6	False
7	0,5	8	7	False	0,5	10	7	False
8	0,5	9	8	False	0,5	12	8	False
9	0,5	10	9	False	0,5	11	9	False
10	0,5	11	10	False	0,5	12	10	False
11	0,5	12	11	False	0,5	13	11	False
12	0,5	13	12	False	0,5	15	12	False
13	0,5	14	13	False	0,5	17	13	False
14	0,5	15	14	False	0,5	18	14	False
15	0,5	16	15	False	0,5	17	15	False
16	0,5	17	16	False	0,5	18	16	False
17	0,5	18	17	False	0,5	19	17	True
18	0,5	19	18	True	0,5	19	18	True
19	0,5	20	19	True	0,5	19	19	True

FIGURE 4.7 – Table de transition

Après Avoir créé la table de transition, nous allons calculer la fonction de valeur $V(s)$ de chaque état qui est initialisé a zéro et ce calcule sera répété jusqu'à ce que V converge vers une valeur optimale et pour cela, nous utilisons un facteur de convergence θ si la différence entre $V(s_t)$ et $V(s_t+1)$ est inférieure ou égale à thêta , donc V à converger . Les figures suivantes montrent les fonctions de valeur à l'itération 1 et la dernière itération :

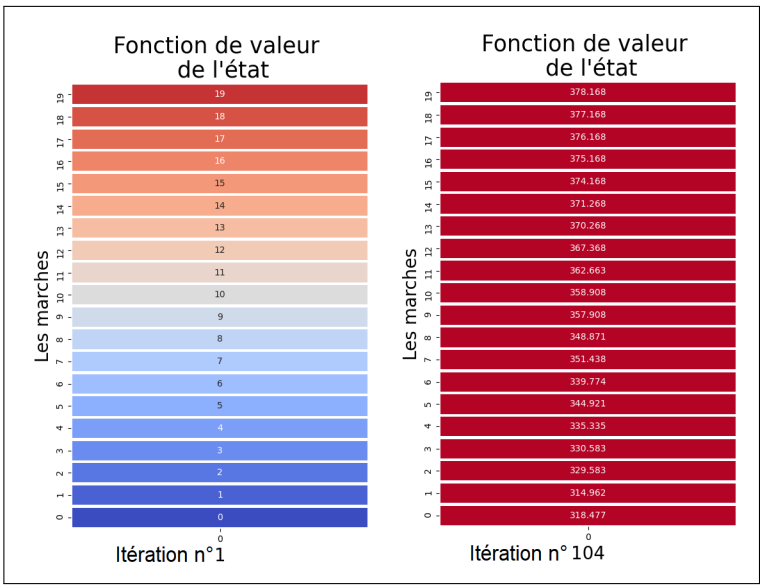

FIGURE 4.8 – fonction de valeur de chaque état a l'itération n°1 et la dernière itération n°104

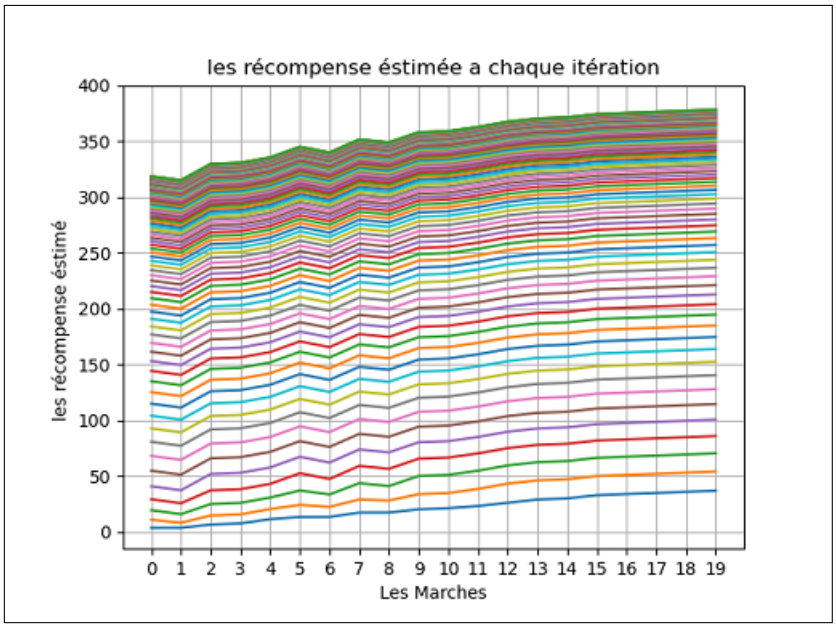

FIGURE 4.9 – Graphe de convergence de la fonction de valeur d'états

Après avoir calculé la fonction de valeur optimale pour chaque état, nous allons utiliser ces résultats pour choisir à chaque fois l'action qui maximise la fonction de valeur de chaque état. La figure suivante montre la politique optimale obtenue :

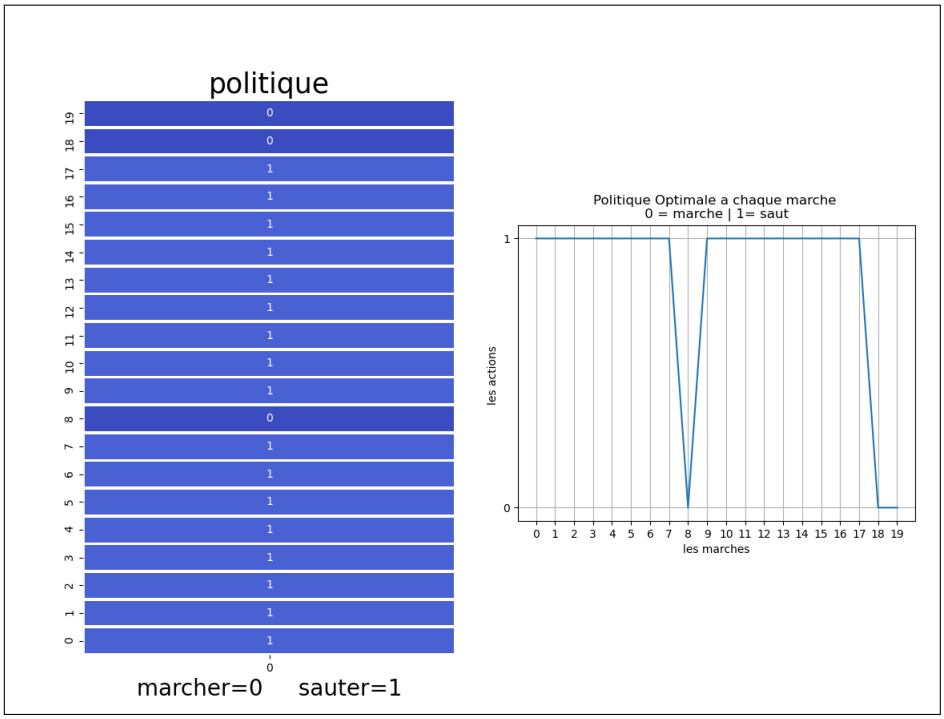

FIGURE 4.10 – politique optimale

4.3.2.4 Résultats

Les résultats de la première approche ne sont pas satisfaisants, car l'agent ne prend pas en considération le risque du saut, le nombre de marches qu'il saute est choisi aléatoirement et donc, l'agent n'est pas en train d'apprendre de son environnement, les figures suivantes montrent la différence des tables de transitions pendant deux exécutions différentes :

Les marches	Les actions							
	Marcher				Sauter =1			
	Probabilité	s'	récompense	Terminer	Probabilité	s'	récompense	Terminer
0	0,5	1	0	False	0,5	4	0	False
1	0,5	2	1	False	0,5	5	1	False
2	0,5	3	2	False	0,5	5	2	False
3	0,5	4	3	False	0,5	7	3	False
4	0,5	5	4	False	0,5	6	4	False
5	0,5	6	5	False	0,5	9	5	False
6	0,5	7	6	False	0,5	10	6	False
7	0,5	8	7	False	0,5	10	7	False
8	0,5	9	8	False	0,5	11	8	False
9	0,5	10	9	False	0,5	11	9	False
10	0,5	11	10	False	0,5	14	10	False
11	0,5	12	11	False	0,5	14	11	False
12	0,5	13	12	False	0,5	16	12	False
13	0,5	14	13	False	0,5	17	13	False
14	0,5	15	14	False	0,5	17	14	False
15	0,5	16	15	False	0,5	19	15	True
16	0,5	17	16	False	0,5	19	16	True
17	0,5	18	17	False	0,5	19	17	True
18	0,5	19	18	True	0,5	19	18	True
19	0,5	20	19	True	0,5	19	19	True

Execution n°1

Les marches	Les actions							
	Marcher				Sauter =1			
	Probabilité	s'	récompense	Terminer	Probabilité	s'	récompense	Terminer
0	0,5	1	0	False	0,5	2	0	False
1	0,5	2	1	False	0,5	3	1	False
2	0,5	3	2	False	0,5	6	2	False
3	0,5	4	3	False	0,5	6	3	False
4	0,5	5	4	False	0,5	6	4	False
5	0,5	6	5	False	0,5	8	5	False
6	0,5	7	6	False	0,5	9	6	False
7	0,5	8	7	False	0,5	10	7	False
8	0,5	9	8	False	0,5	12	8	False
9	0,5	10	9	False	0,5	11	9	False
10	0,5	11	10	False	0,5	12	10	False
11	0,5	12	11	False	0,5	13	11	False
12	0,5	13	12	False	0,5	15	12	False
13	0,5	14	13	False	0,5	17	13	False
14	0,5	15	14	False	0,5	18	14	False
15	0,5	16	15	False	0,5	17	15	False
16	0,5	17	16	False	0,5	18	16	False
17	0,5	18	17	False	0,5	19	17	True
18	0,5	19	18	True	0,5	19	18	True
19	0,5	20	19	True	0,5	19	19	True

Execution n°2

FIGURE 4.11 – Exemple de tables de transitions lors de deux exécutions

On remarque aussi qu'avec le changement de table de transition les valeurs d'état-action $Q(s, a)$ sont alors différentes a chaque exécution, donc la valeur d'état $V(s)$ change aussi, voici un exemple dans la figure suivante :

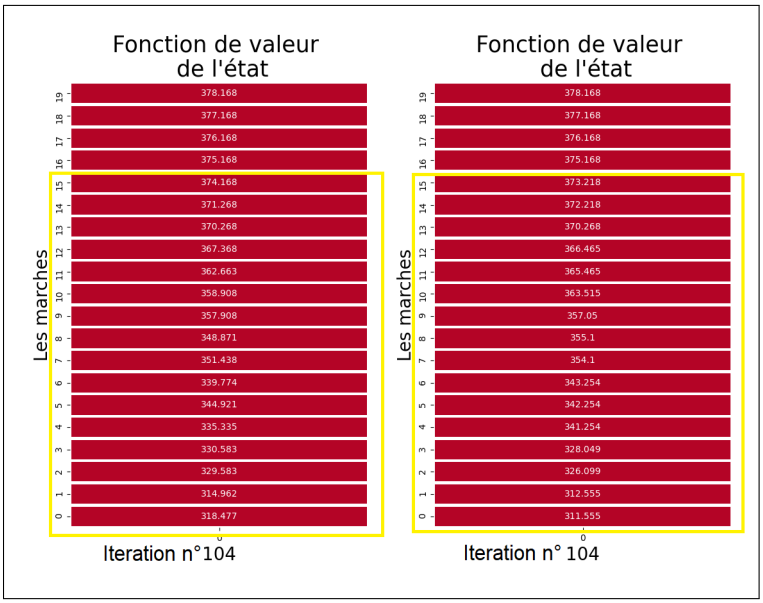

FIGURE 4.12 – Valeurs d'état optimal a l'itération n°104 de deux exécutions différentes.

Les deux figurent montrent que les valeurs ne sont pas optimales, car elles changent à chaque exécution , donc la politique générer à partir des valeurs d'état n'est pas optimale aussi , voici un exemple dans les figures suivant qui montre cela :

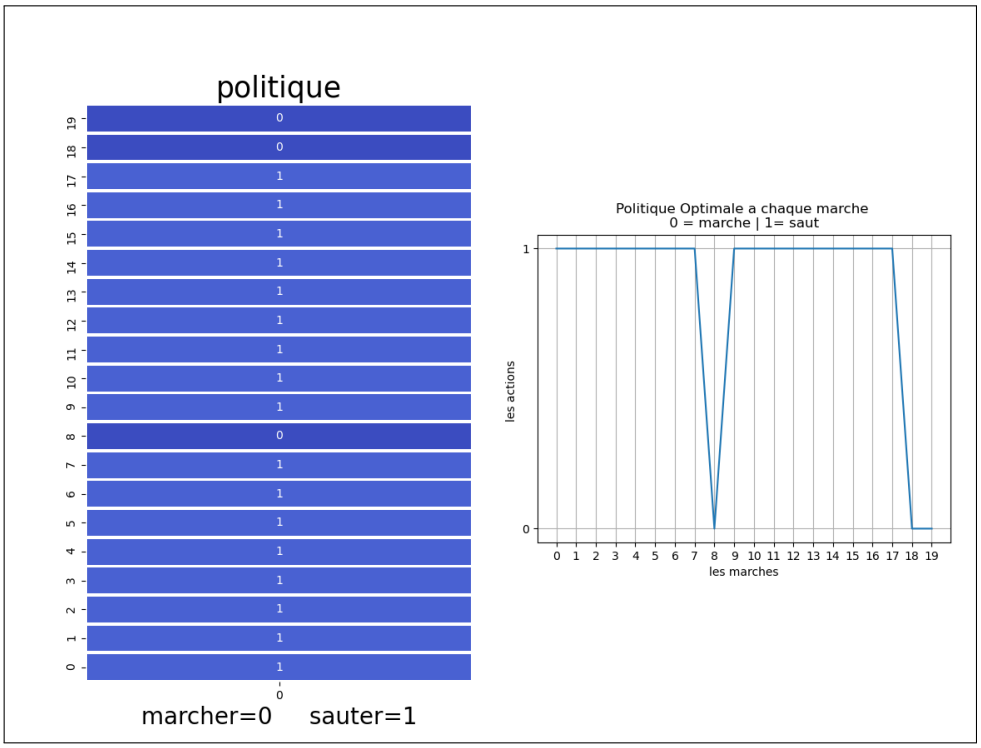

FIGURE 4.13 – Politique optimale à l'itération n°104 à l'exécution n°1

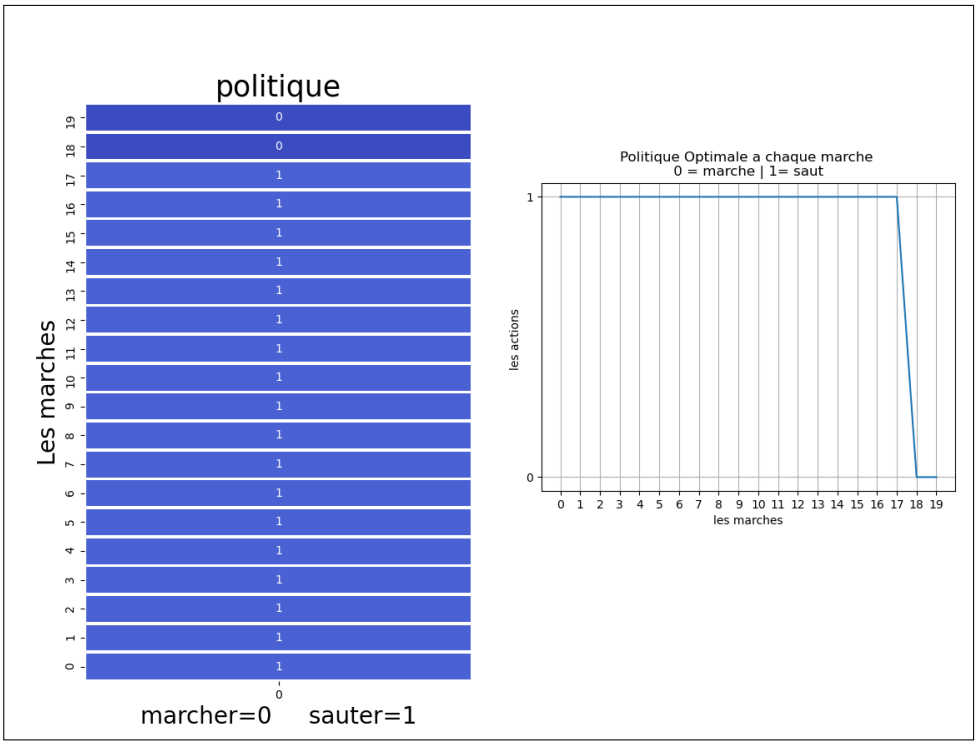

FIGURE 4.14 – Politique optimale à l'itération n°104 à l'exécution n°2

D'après, c'est résultats on constate que la première approche ne répond pas exactement a la problématique, car l'agent n'est pas en train d'apprendre de son environnement, il ne choisit pas le nombre de marches à sauter en fonction de son état par contre il est donné aléatoirement a chaque exécution, de plus le risque n'est pas du tout prit en compte, car le taux de risque augmente en augmentant le nombre de marches à sauter, c'est pourquoi à partir de ces résultats, nous avons développé une autre solution qui traite l'incompétence de la solution 1.

4.3.3 Seconde approche

Les résultats non-satisfaisants de la première approche décrite juste avant ont mené au développement de cette solution qui améliore l'apprentissage de l'agent tout en assurant sa sécurité.

4.3.3.1 Implémentation de l'environnement

Afin de résoudre le souci de l'apprentissage de l'agent, aux lieux de choisir aléatoirement le nombre de marches à sauter lors de l'action sauter, l'agent sera toujours limité à un saut maximal de 3 marche, mais chaque saut sera considéré comme action et chaque action auras un taux de risque qui sera pris en compte, donc le nombre d'actions sera égale à 4 comme suit :

- marcher :- elle consiste déplacer l'agent de la marche m a la marche $(m+1)$ avec un risque de 0

- saut 1 = elle consiste à déplacer l'agent de la marche m a la marche $(m+2)$ avec un risque de 25

- saut 2 = elle consiste à déplacer l'agent de la marche m a la marche $(m+3)$ avec un risque de 50

- saut 3 = elle consiste à déplacer l'agent de la marche m a la marche $(m+4)$ avec un risque de 75

Les figures suivantes montrent le code de la classe "EscalGridWorld" de l'environnement qui assez similaire à celui de la solution 1, mais en modifiant les actions et en ajoutant le taux de risques :

```python
import numpy as np
import random
class EscalGridWorld():
    def __init__(self, size=20):
        self.size = size
        self.nS = size
        self.nA = 4
        self.MAX_X = size-1

        P = {}

        for s in range(self.nS):
            dynamics_s = {}
            reward= s

            for a in range(self.nA):
                p = 1
                s_prime_list = []

                if(a == 0):
                    s_prime = min(self.MAX_X, s+1)
                    risk=0

                elif(a == 1 ):
                    s_prime = min(self.MAX_X, s+2)
                    risk = 0.25
                elif( a == 2 ):
                    s_prime = min(self.MAX_X,s+3)
                    risk=0.50
                elif(a==3):
                    s_prime = min(self.MAX_X,s+4)
                    risk=0.75
                if(s_prime == self.MAX_X):
                    done = True

                else:
                    done = False

                s_prime_list.append((p,s_prime, reward, risk, done))
                dynamics_s.update({a:s_prime_list})

            P.update({s: dynamics_s})

        self.P = P
```

FIGURE 4.15 – Code de la classe "EscalGridWorld"

Cette implémentation permet de visualiser les états et vérifier si l'agent se met en danger ou non, et la figure montrer juste avant montre que l'agent ne fait pas de choix aléatoire et qu'en même temps, ils assurent sa sécurité, car il ne peut pas sauter plus de trois marches.

4.3.3.2 Implémentation de l'apprentissage de l'agent

L'apprentissage de l'agent sera implémenté dans une classe qui développe un ensemble de méthodes qui permettent de calculer la politique optimale afin que l'agent interagit avec son environnement en toute sécurité et maximise ces récompenses.
L'algorithme utilisé est le même que celui de la solution 1, mais pour pouvoir calculer la fonction valeur de l'état , on calcule la fonction e valeur état $Q_{s,a}$ en utilisant une autre équation qui nous permet de prendre en compte le taux de risque de chaque action, donc l'équation utilisée est comme suit : $Q(_s, a) = p_(s') * (r - (r * t_r)\gamma + v[s'])$ $p(s')$= probabilité de s' r= récompense t_r= taux de risque de l'action γ = taux de réduction des récompenses $v[s']$= valeur de l'état s' Cette équation a été implémentée dans une fonction décrite dans la figure suivante :

```python
def compute_q_value_for_s_a(env,V ,s ,a , gamma):
    q=0
    for (p,s_prime, reward, risk, done) in env.P[s][a]:
        q= p*(reward-(reward*risk)+(gamma*V[s_prime]))

    return q
```

FIGURE 4.16 – Code de la fonction de calcule $Q(_{s,a})$

L'algorithme utilisé pour calculer la politique optimale est le même que celui de la première approche intitulé 'policy improvement' le code reste inchangé, car il s'appuie sur les valeurs d'états action donc on n'a pas besoin d'effectuer des modifications, la figure suivante montre le code de la fonction utiliser :

```python
def improved_policy(env , pi , V , gamma):
    for s in range(env.nS):
        q_s = np.zeros([env.nA , 1])
        for a in range (env.nA):
            q_s[a]= compute_q_value_for_s_a(env, V , s ,a , gamma)
            print("s",s,"valeur q(s,a)",q_s[a],"action",a)
            best_a = np.argmax(q_s)
            pi[s]= [best_a]
    plt.plot(pi,'bo')
    plt.xticks(range(0,20, 1))
    plt.xlabel("les marches")
    plt.yticks(range(0,4,1))
    plt.ylabel("les actions ")
    plt.grid(True)
    plt.title("Politique Optimale a chaque marche \n 0 = marche | 1= saut 1 | 2 = saut 2 | 3 = saut 3")
    plot_visualisation_P(env,pi)
    graph_transition1()

    return pi
```

FIGURE 4.17 – Code fonction 'policy improvement'

4.3.3.3 Simulation

À chaque exécution, l'environnement génère une table de transition qui permet de retourner toutes les transitions possibles avec tous les éléments nécessaires pour calculer la fonction fonction de valeur, la figure suivante montre un exemple de table de transitions.

Les marches	Les actions																			
	Marcher					Sauter =1					Sauter =2					Sauter =3				
	Probabilité	s'	récompense	Risque	Terminer	Probabilité	s'	récompense	risque	Terminer	Probabilité	s'	risque	récompense	Terminer	Probabilité	s'	risque	récompense	Terminer
0	1	1	0	0	False	1	2	0	0,25	False	1	3	0,5	0	False	1	4	0,75	0	False
1	1	2	1	0	False	1	3	1	0,25	False	1	4	0,5	1	False	1	5	0,75	1	False
2	1	3	2	0	False	1	4	2	0,25	False	1	5	0,5	2	False	1	6	0,75	2	False
3	1	4	3	0	False	1	5	3	0,25	False	1	6	0,5	3	False	1	7	0,75	3	False
4	1	5	4	0	False	1	6	4	0,25	False	1	7	0,5	4	False	1	8	0,75	4	False
5	1	6	5	0	False	1	7	5	0,25	False	1	8	0,5	5	False	1	9	0,75	5	False
6	1	7	6	0	False	1	8	6	0,25	False	1	9	0,5	6	False	1	10	0,75	6	False
7	1	8	7	0	False	1	9	7	0,25	False	1	10	0,5	7	False	1	11	0,75	7	False
8	1	9	8	0	False	1	10	8	0,25	False	1	11	0,5	8	False	1	12	0,75	8	False
9	1	10	9	0	False	1	11	9	0,25	False	1	12	0,5	9	False	1	13	0,75	9	False
10	1	11	10	0	False	1	12	10	0,25	False	1	13	0,5	10	False	1	14	0,75	10	False
11	1	12	11	0	False	1	13	11	0,25	False	1	14	0,5	11	False	1	15	0,75	11	False
12	1	13	12	0	False	1	14	12	0,25	False	1	15	0,5	12	False	1	16	0,75	12	False
13	1	14	13	0	False	1	15	13	0,25	False	1	16	0,5	13	False	1	17	0,75	13	False
14	1	15	14	0	False	1	16	14	0,25	False	1	17	0,5	14	False	1	18	0,75	14	False
15	1	16	15	0	False	1	17	15	0,25	False	1	18	0,5	15	False	1	19	0,75	15	True
16	1	17	16	0	False	1	18	16	0,25	False	1	19	0,5	16	True	1	19	0,75	16	True
17	1	18	17	0	False	1	19	17	0,25	True	1	19	0,5	17	True	1	19	0,75	17	True
18	1	19	18	0	True	1	19	18	0,25	True	1	19	0,5	18	True	1	19	0,75	18	True
19	1	19	19	0	True	1	19	19	0,25	True	1	19	0,5	19	True	1	19	0,75	19	True

FIGURE 4.18 – Table de transitions

Après Avoir créé la table de transition, nous allons calculer la fonction de valeur $V(s)$ de chaque état qui est initialisé a zéro et ce calcule sera répété jusqu'à ce que V converge vers une valeur optimale et pour cela, nous utilisons un facteur de convergence θ si la différence entre $V(s_t)$ et $V(s_t + 1)$ est inférieure ou égale à thêta, donc V à converger . Les figures suivantes montrent les fonctions de valeur à l'itération 1 et la dernière itération :

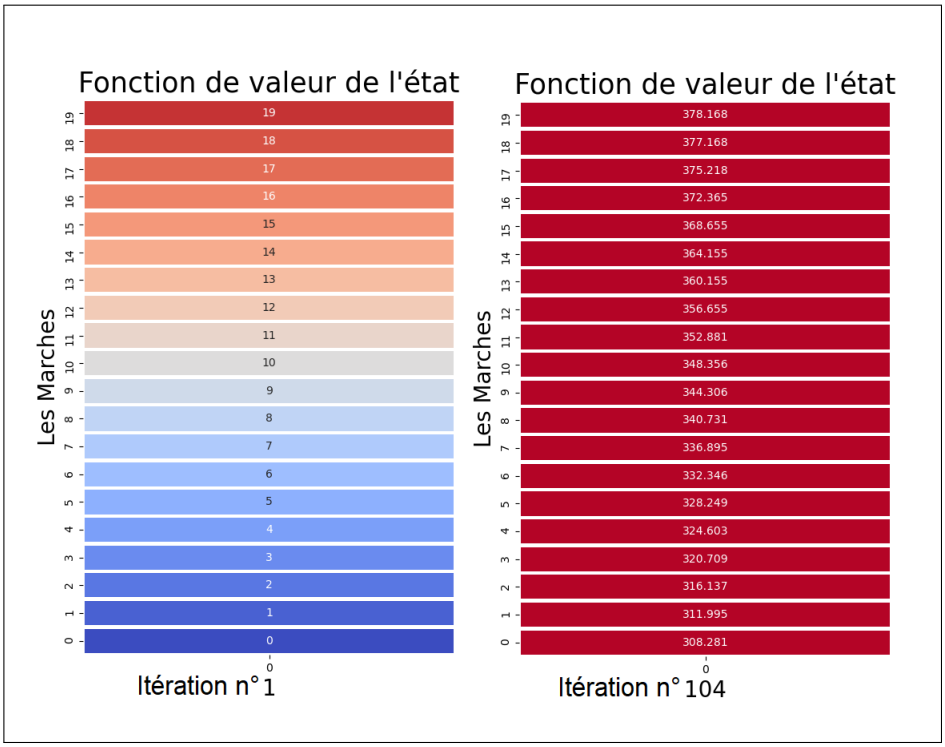

FIGURE 4.19 – Visualisation des valeurs d'états par rapport aux itérations

La figure juste avant montre que les valeurs se rapprochent ce qui prouve que $v(s)$ converge vers $v^*(s)$ et afin de mieux visualiser cela, voici dans la figure suivante une représentation graphique qui indique la valeur d'états a chaque itération :

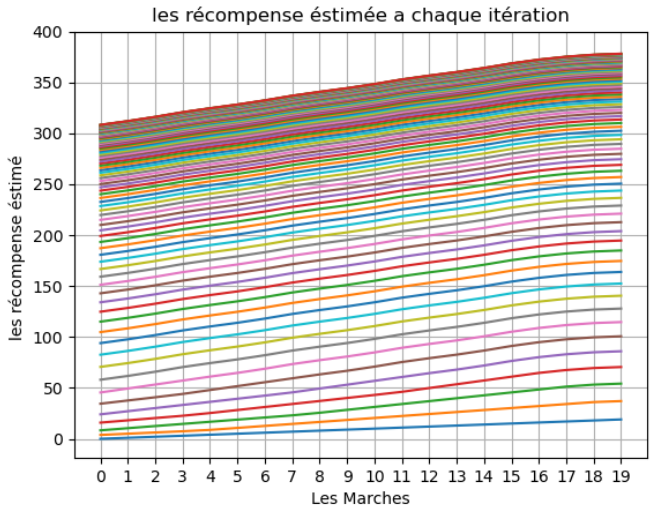

FIGURE 4.20 – Graphe de convergences des valeurs d'états par rapport aux itérations

Après avoir calculé la fonction de valeur optimale pour chaque état, nous allons utiliser ces résultats pour choisir à chaque fois l'action qui maximise la fonction de valeur de chaque état. La figure suivante montre la politique optimale obtenue :

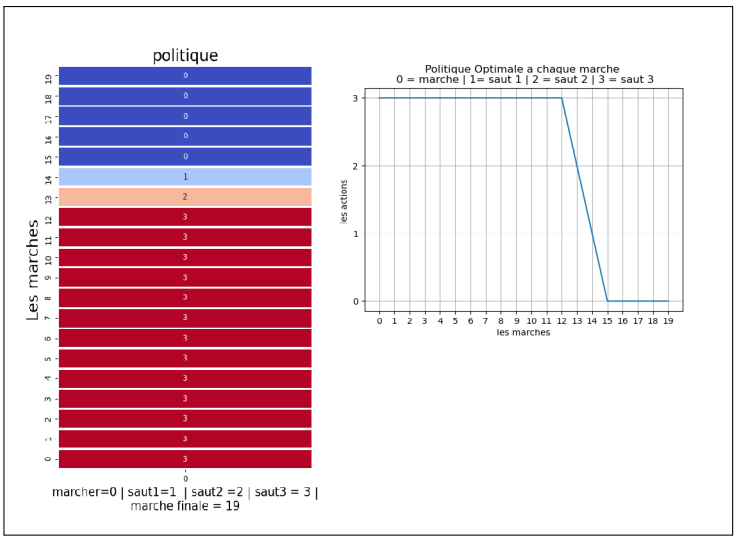

FIGURE 4.21 – Graphe de convergences des valeurs d'états par rapport aux itérations

4.3.3.4 Résultat :

Les résultats de la seconde approche montrent que l'agent ne peut pas effectuer d'actions dangereuses, comme montré dans la "Figure4.18-Table de transition " qui ne change pas a chaque exécution, donc les fonctions de valeur convergent vers une valeur optimale en prenant compte du risque de chaque action et c'est ce qui permet alors de calculer la politique optimale. Dans les figures suivantes, voici des exemples de l'exécution de la politique optimale par l'agent dans un escalier de 20 marche :

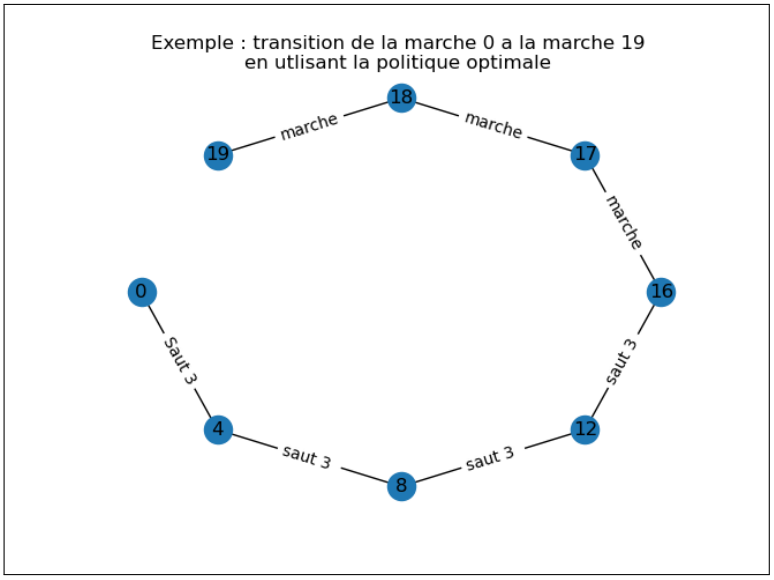

FIGURE 4.22 – Graphe exemple chemin de la marche 0 a la marche 19

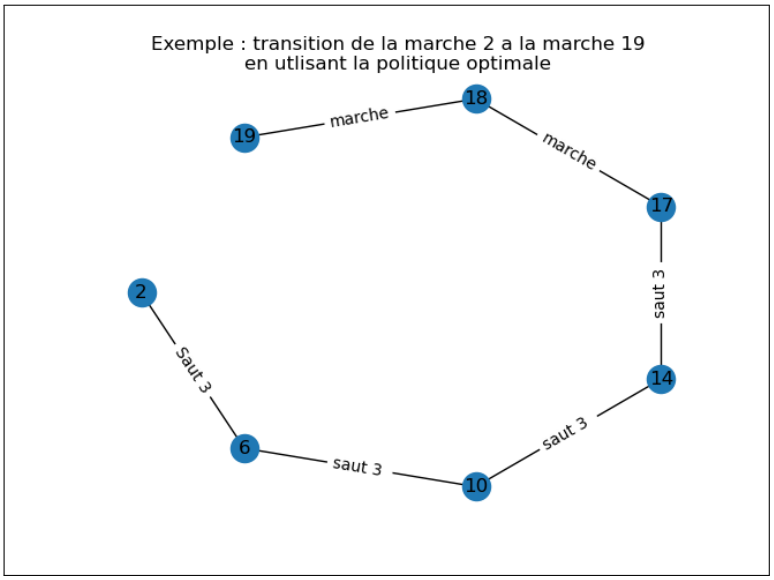

FIGURE 4.23 – Graphe exemple chemin de la marche 2 a la marche 19

4.3.4 Comparaison entre les deux solutions

Dans l'ensemble, ces résultats montrent que la seconde approche est très prometteuse. Elle garantit la sécurité en prenant en compte le taux de risque de chaque nombre de marches sauté sans sacrifier beaucoup de performances. Il existe inévitablement un compromis entre la sécurité et les bonnes performances. Contrairement à la première approche qui assure la sécurité de l'agent, mais sans lui permettre d'interagir et d'apprendre de son environnement, ce qui ne lui permet pas de maximiser ces récompenses, car le choix des actions est aléatoire, et il ne prend pas en considération le taux de risque de chaque nombre de marches sauté, c'est pourquoi la seconde approche a été développer et répond exactement à la problématique poser.

4.3.5 Discussion

Dans ce chapitre, nous avons présenté l'implémentation de nos deux solutions, tout d'abord, on a présenté l'environnement de développement ainsi que les algorithmes utilisés, puis nous avons donné une description détaillée du système à travers des fenêtres de capture qui représentent les codes et les résultats de ce dernier. Les résultats obtenus sur notre exemple sont encourageants, mais ils nécessitent encore des analyses de performance des agents plus détaillées afin qu'elles puissent être concrétisées. Dans tous les cas, les performances d'apprentissage des agents se sont améliorées par rapport au cas où l'agent n'était pas protégé. Le principal inconvénient de notre approche est que pour pouvoir empêcher l'agent embarqué de faire des actions dangereuses, certaines aptitudes sont nécessaires. Il faut disposer d'un rapport détaillé de l'agent afin de pouvoir préciser le moment où une action est dangereuse. Nous soutenons que cela est inévitable si les actions autorisées dépendent de l'état de l'environnement, car sinon, il n'y a aucun moyen de savoir quelles actions sont dangereuses.

Chapitre 5

Conclusion

Les systèmes embarqués intelligents sont de plus en plus intégrés dans des environnements complexes, du monde réel, la conception de la sécurité devient une considération essentielle. Nous nous concentrons spécifiquement sur la recherche de scénarios où les agents peuvent provoquer des réactions indésirables lors de l'exécution.

Notre objectif principal de recherche était de proposer et implémenter une approche qui permet d'assurer la sécurité des systèmes embarqués afin d'éviter les dégâts qui peuvent être causés sans perturber leur fonctionnement. La solution que nous avons proposée remplit parfaitement ces critères.

Notre travail débute par une introduction dans les systèmes cyber physiques, cette partie a pour objectif de comprendre ce qu'est un système cyber physique et définir la relation entre ces derniers et les systèmes embarqués intelligents cela nous a conduit vers une prise de connaissance sur la vulnérabilité de ces systèmes et l'importance d'assurer leur sécurité.

Dans l'optique d'assurer la sécurité des systèmes embarqués intelligents, nous nous sommes, tout d'abord inspirés des expériences réalisées dans des études existantes. Ces études nous ont permis de développer notre propre approche basée sur l'apprentissage par renforcement.

Nos investigations dans le domaine de l'apprentissage par renforcement, nous ont permise d'apporter des contributions substantielles que nous partageons en deux solutions, la première a été basée sur l'évitement total du danger en minimisant l'effet de l'action, mais en choisissant aléatoirement l'impacte de celle-ci, sans permettre à l'agent d'apprendre de son environnement, c'est pourquoi à partir de ces résultats, nous avons développé une deuxième solution qui évite aussi la mise en danger de l'agent, mais qui lui permet d'apprendre de son environnement en toute sécurité, Les simulations ont montré l'efficacité de la solution développer. Nos expériences dans les applications où l'apprentissage sûr est nécessaire, montre une grande promesse d'amélioration des performances d'apprentissage.

Bibliographie

[1] URL : https://spinningup.openai.com/en/latest/.

[2] URL : https://gym.openai.com/.

[3] URL : https://docs.ray.io/en/master/rllib-examples.html.

[4] URL : https://developer.nvidia.com/isaac-gym.

[5] J. Achiam. « Spinning up in deep reinforcement learning ». In : *Dosegljivo : spinningup. openai. com.[Dostopano 12. 7. 2020]* (2018).

[6] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman et D. Mané. « Concrete problems in AI safety ». In : *arXiv preprint arXiv :1606.06565* (2016).

[7] C. G. Atkeson et J. C. Santamaria. « A comparison of direct and model-based reinforcement learning ». In : *Proceedings of international conference on robotics and automation*. T. 4. IEEE. 1997, p. 3557-3564.

[8] S. Behere, F. Asplund, A. Söderberg et M. Törngren. « Architecture challenges for intelligent autonomous machines ». In : *Intelligent Autonomous Systems 13*. Springer, 2016, p. 1669-1681.

[9] K. W. Choy. *Blackboard architecture for intelligent embedded systems*. Nottingham Trent University (United Kingdom), 2005.

[10] E. S. P. Course. « Reinforcement Learning in Energy Systems ». In : ().

[11] A. massoud Farahmand, A. Shademan, M. Jagersand et C. Szepesvári. « Model-based and model-free reinforcement learning for visual servoing ». In : *2009 IEEE International Conference on Robotics and Automation*. IEEE. 2009, p. 2917-2924.

[12] A. M. Farahmand, R. Munos et C. Szepesvári. « Error propagation for approximate policy and value iteration ». In : *Advances in Neural Information Processing Systems*. 2010.

[13] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare et J. Pineau. « An introduction to deep reinforcement learning ». In : *arXiv preprint arXiv :1811.12560* (2018).

[14] J. Fu, S. Shen, C. Cao et C. Li. « Fast Robot Motor Skill Acquisition Based on Bayesian Inspired Policy Improvement ». In : *International Conference on Intelligent Robotics and Applications*. Springer. 2019, p. 356-367.

[15] N. Fulton et A. Platzer. « Safe reinforcement learning via formal methods : Toward safe control through proof and learning ». In : *Proceedings of the AAAI Conference on Artificial Intelligence*. T. 32. 1. 2018.

[16] J. Garcia et F. Fernández. « A comprehensive survey on safe reinforcement learning ». In : *Journal of Machine Learning Research* 16.1 (2015), p. 1437-1480.

[17] A. Hagberg, P. Swart et D. S Chult. *Exploring network structure, dynamics, and function using NetworkX*. Rapp. tech. Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[18] L. P. Kaelbling, M. L. Littman et A. W. Moore. « Reinforcement learning : A survey ». In : *Journal of artificial intelligence research* 4 (1996), p. 237-285.

[19] L. Kaiser, M. Babaeizadeh, P. Milos, B. Osinski, R. H. Campbell, K. Czechowski et al. « Model-based reinforcement learning for atari ». In : *arXiv preprint arXiv :1903.00374* (2019).

[20] M. Kearns et D. Koller. « Efficient reinforcement learning in factored MDPs ». In : *IJCAI*. T. 16. 1999, p. 740-747.

[21] Z. J. Kuang, L. Hu et C. Zhang. « Characteristic Analyzation of Cyber Physical Systems ». In : *Green Power, Materials and Manufacturing Technology and Applications III*. T. 484. Applied Mechanics and Materials. Trans Tech Publications Ltd, mar. 2014, p. 427-430. doi : 10.4028/www.scientific.net/AMM.484-485.427.

[22] R. Learning. *An Introduction, Richard S. Sutton and Andrew G. Barto*. 1998.

[23] D. D. Lee et H. S. Seung. « Learning in Intelligent Embedded Systems. » In : *Usenix Workshop on Embedded Systems*. 1999.

[24] E. A. Lee. « Cyber-physical systems-are computing foundations adequate ». In : *Position paper for NSF workshop on cyber-physical systems : research motivation, techniques and roadmap*. T. 2. Citeseer. 2006, p. 1-9.

[25] Y. Li. « Deep reinforcement learning : An overview ». In : *arXiv preprint arXiv :1701.07274* (2017).

[26] M. Lutz et C. Toporek. *Python*. O'Reilly Media, Inc., 2002.

[27] A. K. Mackworth. « Constraint-based design of embedded intelligent systems ». In : *Constraints* 2.1 (1997), p. 83-86.

[28] L. Monostori, B. Kádár, T. Bauernhansl, S. Kondoh, S. Kumara, G. Reinhart et al. « Cyber-physical systems in manufacturing ». In : *Cirp Annals* 65.2 (2016), p. 621-641.

[29] T. E. Oliphant. *A guide to NumPy*. T. 1. Trelgol Publishing USA, 2006.

[30] D. Papp, Z. Ma et L. Buttyan. « Embedded systems security : Threats, vulnerabilities, and attack taxonomy ». In : *2015 13th Annual Conference on Privacy, Security and Trust (PST)*. ieee. 2015, p. 145-152.

[31] D. Ramani. « A Short Survey On Memory Based Reinforcement Learning ». In : *arXiv preprint arXiv :1904.06736* (2019).

[32] J. Reback, W. McKinney, J. Den Van Bossche, T. Augspurger, P. Cloud, A. Klein et al. « pandas-dev/pandas : Pandas 1.0. 3 ». In : *Zenodo* (2020).

[33] G. A. Rummery. « Problem solving with reinforcement learning ». Thèse de doct. Citeseer, 1995.

[34] R. Sathya et A. Abraham. « Comparison of supervised and unsupervised learning algorithms for pattern classification ». In : *International Journal of Advanced Research in Artificial Intelligence* 2.2 (2013), p. 34-38.

[35] K. Srinivasa, G. Siddesh et H. Srinidhi. « Getting Started with Visualization in Python ». In : *Network Data Analytics*. Springer, 2018, p. 333-337.

[36] A. L. Strehl, L. Li et M. L. Littman. « Reinforcement Learning in Finite MDPs : PAC Analysis. » In : *Journal of Machine Learning Research* 10.11 (2009).

[37] M. L. Waskom. « Seaborn : statistical data visualization ». In : *Journal of Open Source Software* 6.60 (2021), p. 3021.

[38] W. Wolf. « Cyber-physical systems ». In : *IEEE Annals of the History of Computing* 42.03 (2009), p. 88-89.

[39] M. Zimmer. « Apprentissage par renforcement développemental ». Thèse de doct. Université de Lorraine, 2018.