Universite Catholique de Louvain
Faculté des Sciences Appliquées
Département d'ingénierie informatique

Apprentissage de la coordination dans les systèmes multi-agents

Promoteur : Professeur P. Van Roy
Co-promoteur : S. Gualandi

Mémoire présenté en vue
de l'obtention du grade
d'ingénieur civil informaticien
par

Jean-Yves Lawson
François Mairesse

Louvain-la-Neuve
Année académique 2003-2004

Nous tenons à remercier notre co-promoteur Stefano Gualandi, pour ses critiques constructives et ses encouragements tout au long de ce mémoire.

Nous remercions également notre promoteur Peter Van Roy pour ses suggestions pertinentes.

Enfin, nous souhaitons remercier toutes les personnes qui ont contribué à l'amélioration du code et à la relecture de ce travail.

Table des Matières

Abstract . . . 9
Résumé . . . 10

Introduction **11**

1 Concepts théoriques **13**
1.1 Concepts de base . . . 13
1.1.1 Agent . . . 13
1.1.2 Environnement . . . 15
1.1.3 Définitions . . . 16
1.1.4 Machine Learning . . . 16
1.1.5 Le machine learning dans les systèmes multi-agents . . . 17
1.2 Etat de l'art . . . 20
1.2.1 Algorithmes d'apprentissage . . . 20
1.2.2 Améliorations du Q learning . . . 23
1.2.3 Interactions entre agents . . . 24
1.2.4 Conclusion . . . 26
1.3 Introduction au Q learning . . . 28
1.3.1 Les hypothèses . . . 28
1.3.2 Objet de l'apprentissage . . . 28
1.3.3 Algorithme de base . . . 29
1.3.4 Convergence . . . 31
1.3.5 Exemple . . . 31
1.3.6 Améliorations . . . 33
1.3.7 Politiques d'exploration . . . 34
1.3.8 Q learning non-déterministe . . . 35

2 Mise en oeuvre des techniques existantes **37**
2.1 Présentation du problème . . . 37
2.1.1 Scénario . . . 37
2.1.2 Hypothèses de modélisation . . . 38
2.1.3 Extension de la modélisation . . . 38
2.1.4 Motivations . . . 40
2.2 Modélisation du problème . . . 42
2.2.1 Agent unique . . . 42
2.2.2 Simulateur . . . 43
2.2.3 Communication entre agent et simulateur . . . 44
2.2.4 Coordination de plusieurs agents . . . 45
2.2.5 Agents indépendants . . . 46
2.2.6 Agents collaboratifs . . . 48

2.3 Résultats expérimentaux 50
2.3.1 Agent unique 50
2.3.2 Agents indépendants 55
2.3.3 Agents collaboratifs 63
2.3.4 Obstacles dynamiques 77
2.3.5 Impact des paramètres 84
2.3.6 Limitations 86
2.3.7 Conclusion 87

3 Généralisation d'une fonction Q de deux agents **89**
3.1 Description de la méthode JAG 89
3.1.1 Avantages 90
3.1.2 Limitations 93
3.2 Résultats expérimentaux 94
3.3 Conclusion 98

4 Travail ultérieur **99**
4.1 Approximation de la fonction Q 99
4.1.1 Exemple d'application 99
4.1.2 Limitations 101
4.2 Configuration automatique des paramètres 101
4.2.1 Learning Momentum 101
4.2.2 Algorithme génétique 102
4.3 Apprentissage de comportements primitifs 103
4.4 Raffinement de graphe 103
4.5 Apprentissage de sous-goals 106
4.6 Initialisation basée sur un algorithme du plus court chemin 106
4.7 Propagation des modifications de la table $\hat{Q}$ 107

Conclusion **109**

Références **112**

A Concepts de programmation utilisés **114**

B Manuel de l'utilisateur **116**
B.1 Installation 116
B.2 Utilisation de l'interface 117
B.2.1 Panneau de configuration 117
B.2.2 Panneau de simulation 121
B.2.3 Panneau de statistiques 123

Liste des Figures

1.1 Architecture générique d'un agent. 14
1.2 Exemple de politique obtenue après le premier épisode. 32
1.3 Exemple de politique obtenue après le second épisode. 32
1.4 Exemple de politique optimale. 33

2.1 Illustration du problème général. 38
2.2 Description du problème dans le cas de quatre agents évoluant dans une grille de taille 10×10 [12]. 39
2.3 Représentation d'un environnement complexe en utilisant un graphe. 40
2.4 Graphe orienté unidirectionnel limitant les actions possibles dans chaque état. 40
2.5 Synchronisation des messages reçus au court d'un même intervalle de temps, dans le cas de deux agents envoyant chacun 4 messages à des instants différents. Le message 2 est considéré comme étant isolé alors que les messages 6 et 3 sont simultanés. 45
2.6 Diagramme de messages représentant la communication entre agent et simulateur. 46
2.7 Collision entre deux agents A_1 et A_2 ne pouvant pas être apprise dans le cas de fonctions Q indépendantes et d'une politique d'exploration aléatoire. . . 47
2.8 Taille de l'espace état-action en fonction du nombre d'agents dans le cas d'une table jointe. 49
2.9 Environnement de test pour un agent. 50
2.10 Moyennes des longueurs des chemins obtenus après chaque épisode par un agent unique apprenant par exploration aléatoire d'un environnement comportant des obstacles. 52
2.11 Moyennes des collisions occasionnées par un agent unique apprenant par exploration aléatoire dans un environnement comportant des obstacles. . . . 52
2.12 Moyennes des longueurs des chemins obtenus par un agent unique apprenant par exploration probabiliste un environnement comportant des obstacles. . 53
2.13 Moyennes des collisions occasionnées par un agent unique apprenant par exploration probabiliste dans un environnement comportant des obstacles. . 54
2.14 Scénario de test pour deux agents. 55
2.15 Moyennes des longueurs des chemins obtenus pour deux agents indépendants explorant aléatoirement un environnement sans obstacle. 56
2.16 Nombre moyen d'agents entrant en collision lors de l'apprentissage pour deux agents indépendants dans un environnement sans obstacle. 57
2.17 Moyennes des longueurs des chemins dans le cas de deux agents indépendants dans un environnement sans obstacle. Q learning non-déterministe. 57
2.18 Effet du Q learning non-déterministe sur le nombre moyen de collisions. . . 58

2.19 Effet des données des capteurs de proximité sur la courbe des longueurs des chemins. . . . 59
2.20 Effet des données des capteurs de proximité sur la courbe du nombre d'agents entrant en collision. . . . 59
2.21 Scénario 1*b*. Influence du choix probabiliste des actions d'exploration sur la moyenne des longueurs des chemins dans le cas d'agents indépendants. . . . 60
2.22 Scénario 1*b*. Influence du choix probabiliste des actions d'exploration sur le nombre moyen de collisions dans le cas d'agents indépendants. . . . 61
2.23 Scénario de test pour deux agents dans un environnement avec des obstacles. 61
2.24 Moyennes des longueurs des chemins trouvés par deux agents indépendants dans un environnement contenant des obstacles. . . . 62
2.25 Nombre moyen des collisions occasionnées par deux agents dans un environnement contenant des obstacles. . . . 63
2.26 Moyennes des longueurs des chemins trouvés par deux agents indépendants dans un environnement comportant des obstacles, avec exploration probabiliste. . . . 64
2.27 Nombre moyen de collisions occasionnées par deux agents indépendants dans un environnement comportant des obstacles, avec exploration probabiliste. . 64
2.28 Moyennes des longueurs des chemins trouvés par deux agents collaboratifs dans un environnement sans obstacle. . . . 66
2.29 Nombre moyen des collisions subies par deux agents dans un environnement sans obstacle. . . . 66
2.30 Nombre moyen des collisions subies par deux agents collaboratifs dans un environnement sans obstacle. . . . 67
2.31 Moyennes des longueurs des chemins trouvés par deux agents collaboratifs dans un environnement sans obstacle avec exploration probabiliste. . . . 67
2.32 Moyennes des collisions subies par deux agents collaboratifs dans un environnement sans obstacle avec exploration probabiliste. . . . 68
2.33 Moyennes des collisions subies par deux agents collaboratifs dans un environnement sans obstacle avec exploration probabiliste (zoom). . . . 68
2.34 Scénario de test pour quatre agents. . . . 69
2.35 Moyennes des longueurs des chemins obtenus par quatre agents collaboratifs explorant aléatoirement un environnement sans obstacle. . . . 70
2.36 Nombre moyen des collisions, sur 20 apprentissages, occasionnées par quatre agents collaboratifs explorant aléatoirement un environnement sans obstacle. 70
2.37 Moyennes des longueurs des chemins obtenus par quatre agents collaboratifs explorant de manière probabiliste l'environnement sans obstacle. . . . 71
2.38 Nombre moyen de collisions occasionnées par quatre agents collaboratifs explorant de manière probabiliste l'environnement sans obstacle. . . . 71
2.39 Moyenne des longueurs des chemins sous-optimaux trouvés par deux agents collaboratifs dans un environnement contenant des obstacles. . . . 72
2.40 Nombre moyen des collisions occasionnées par deux agents collaboratifs dans un environnement contenant des obstacles. . . . 73
2.41 Comparaison des moyennes des longueurs des chemins entre les approches probabiliste et aléatoire avec le scénario 2*b*. . . . 73
2.42 Comparaison des moyennes des nombres de collisions entre les approches probabiliste et aléatoire avec le scénario 2*b*. . . . 74
2.43 Scénario de test pour quatre agents avec obstacles. . . . 74

2.44 Moyennes des longueurs des chemins empruntés par quatre agents collaboratifs dans un environnement comportant des obstacles. 75
2.45 Nombre moyen de collisions occasionnées par quatre agents collaboratifs dans un environnement comportant des obstacles. 76
2.46 Nouvel environnement de test, la position des obstacles est modifiée. 77
2.47 Moyennes des longueurs des chemins empruntés après chaque épisode par deux agents utilisant les données des capteurs de proximité. 78
2.48 Nombre moyen de collisions occasionnées par deux agents utilisant les données des capteurs de proximité dans un environnement avec obstacles. 78
2.49 Scénario $2a$. Exemple de chemins optimaux trouvés par les agents utilisant les données des capteurs de proximité. 79
2.50 Utilisation des données des capteurs de proximité. 79
2.51 Moyennes, sur 20 apprentissages avec des obstacles dynamiques, des longueurs des chemins. 80
2.52 Nombre moyen de collisions, sur 20 apprentissages avec des obstacles dynamiques. 81
2.53 Moyenne des courbes des meilleurs chemins trouvés pendant une série d'apprentissages avec des obstacles aléatoires et le Q learning non-déterministe. . . . 81
2.54 Moyenne des courbes du nombre d'agents entrant en collision pendant une série d'apprentissages avec des obstacles aléatoires et le Q learning non-déterministe. 82
2.55 Exemple de chemins optimaux dans une configuration modifiée. 82
2.56 Exemple de chemins non optimaux dans une configuration modifiée. 83
2.57 Trois agents changeant de position. La fonction de récompense favorise les chemins individuels de longueur minimale. 85
2.58 Trois agents changeant de position. La fonction de récompense favorise les chemins de même longueur. 85
2.59 Comparaison des fonctions d'assignation des récompenses. 86
2.60 Deux agents collaboratifs dans le cas d'une température de Boltzmann constante valant 1.0. 87

3.1 Généralisation d'une politique apprise par deux agents à quatre agents. . . 91
3.2 Comparaison des tailles de l'espace état-action dans le cas de quatre agents entre une table $\hat{Q}$ jointe classique et une table jointe $\hat{Q}$ apprise par deux agents. 92
3.3 Longueur du chemin dans le cas de deux agents dans un environnement exploré aléatoirement et comportant des obstacles. 94
3.4 Nombre de collisions dans le cas de deux agents dans un environnement exploré aléatoirement avec des obstacles. 95
3.5 Longueur du chemin dans le cas de quatre agents dans un environnement exploré aléatoirement sans obstacle. 96
3.6 Nombre de collisions dans le cas de quatre agents dans un environnement exploré aléatoirement sans obstacle. 97
3.7 Longueur des chemins dans le cas de quatre agents dans un environnement exploré aléatoirement comportant des obstacles. 97
3.8 Nombre de collisions dans le cas de quatre agents dans un environnement exploré aléatoirement comportant des obstacles. 98

4.1 Exemple d'approximation d'une fonction Q. 100
4.2 Différentes étapes d'un algorithme génétique. 102

4.3 Graphe complet d'un environnement, le noeud D est la position initiale de l'agent. . . . 104
4.4 Graphe simplifié initial comprenant quatre noeuds. . . . 104
4.5 Graphe final de l'agent après raffinement. . . . 105
4.6 Exemple de sous-goals. . . . 106
4.7 Exemple de propagation partielle des nouvelles valeurs de $\hat{Q}$ d'un épisode d'apprentissage. . . . 108
4.8 Comparaison des longueurs des chemins obtenues avec des agents indépendants, collaboratifs et la méthode JAG. . . . 110
4.9 Comparaison du nombre moyen de collisions obtenu avec des agents indépendants, collaboratifs et la méthode JAG. . . . 110

B.1 Panneau de configuration permettant l'ajustement des paramètres généraux de l'apprentissage. . . . 117
B.2 Panneau de configuration permettant l'ajustement des paramètres relatifs à l'algorithme du Q learning. . . . 118
B.3 Panneau de configuration permettant l'ajustement des paramètres des graphiques du panneau de statistiques. . . . 119
B.4 Panneau de configuration permettant l'ajustement des paramètres relatifs à l'environnement d'apprentissage. . . . 120
B.5 Panneau de simulation permettant de visualiser la politique apprise par les agents. . . . 122
B.6 Panneau de simulation permettant de lancer plusieurs apprentissages à la suite afin d'effectuer des statistiques. . . . 123
B.7 Panneau de statistiques décrivant la longueur du chemin obtenu et le nombre total de collisions en fonction de l'itération d'apprentissage. . . . 124

Liste des Tables

1.1 Algorithme du Q learning dans le cas d'actions et récompenses déterministes. 30

2.1 Performances et limites de l'implémentation. 87

3.1 Algorithme JAG de décision d'un agent parmi N_{agents} utilisant une table $\hat{Q}$ pour deux agents. 90

3.2 Comparaison de taille de l'espace état-action dans le cas de quatre agents entre une table $\hat{Q}$ jointe classique et une table jointe $\hat{Q}$ apprise par deux agents. 92

Abstract

A lot of research has been done in the case of a single agent acting in a static world. Efficient solutions using machine learning have been successfully implemented. We are interested in the coordination of agents in multi-agent systems. The complexity of those systems makes machine learning techniques efficient alternatives to solve such problems.

This work focuses on the use of *reinforcement learning* techniques for the coordination of agents moving in an unknown and dynamic environment. The agents have to learn from experience how to coordinate their actions to maximize their global reward. They also have to learn how to cope with environmental uncertainties.

The *Q learning* algorithm has been widely used for its simplicity and ability to cope with the lack of environmental model. Using this algorithm, collaborative agents were implemented and a new coordination algorithm called *JAG* was designed. The new method is less vulnerable to the state space explosion we encountered while investigating other coordination methods. The algorithm breaks down the learning of coordination between several agents into smaller and more tractable tasks implying only two agents. The sub-solutions are then generalized to solve the initial problem. The experimental results confirm our expectations.

In this *mémoire*, we compare the performances of independent learner and joint action learner methods. Then, we give an overview and the results of the *JAG* method we designed and implemented. The method takes advantage of the good coordination achieved with the joint action learner, while reducing the state-action space.

The implementation part was done with the *Mozart* programming system. It is an advanced development platform based on Oz, a multi-paradigm programming language. Mozart comes with powerful functionalities that have been found useful in the realization of our multi-agent system.

Résumé

Beaucoup de recherches ont été réalisées pour des problèmes mettant en oeuvre un agent unique dans un environnement statique. Des algorithmes basés sur les techniques d'apprentissage des machines ont été implémentés et fournissent de bons résultats. Nous nous intéressons aux problèmes de coordination dans les systèmes multi-agents. La complexité de ceux-ci ouvre de nouvelles perspectives sur l'utilisation des mécanismes d'apprentissage des machines.

Ce travail se concentre sur l'utilisation des techniques de *reinforcement learning* pour la coordination de plusieurs agents dans un monde inconnu et dynamique. Les agents doivent apprendre au fil des expériences à se coordonner afin de maximiser les récompenses reçues. Cet apprentissage doit également permettre de gérer les incertitudes de l'environnement.

L'algorithme *Q learning* est largement utilisé pour sa simplicité et sa capacité à gérer les environnements inconnus. Sur base de celui-ci, des agents collaboratifs ont été implémentés et un nouvel algorithme de coordination nommé *JAG* a été élaboré. La nouvelle approche est moins vulnérable à l'explosion de l'espace d'états rencontrée lors de l'étude d'autres méthodes. L'algorithme décompose tout d'abord l'apprentissage de la coordination entre plusieurs agents en des tâches de moindre complexité impliquant uniquement deux agents. Les sous-solutions obtenues sont alors généralisées pour résoudre le problème initial. Les résultats expérimentaux confirment nos prévisions.

Dans le présent mémoire, une analyse de la qualité de la coordination d'agents indépendants et d'agents collaboratifs est réalisée. Le travail comporte également une évaluation de la méthode *JAG*. Cette méthode tire profit du bon degré de coordination atteint par les agents collaboratifs, tout en gardant un espace d'états de taille réduite.

L'implémentation a été réalisée en utilisant *Mozart*. Il s'agit d'une plateforme de développement basée sur Oz, un langage de programmation multi-paradigmes. *Mozart* fournit des primitives puissantes qui se sont avérées utiles lors de la réalisation du système multi-agents.

Introduction

Depuis longtemps les hommes ont compris que la subdivision du travail constitue une façon efficace de réaliser des tâches de grande complexité. Ce n'est que récemment que les scientifiques tentent d'appliquer ce principe à des agents informatiques. L'utilisation de telles entités intelligentes de manière coopérative rend possible la réalisation des applications résolvant des problèmes de grande envergure.

Une coopération efficace entre agents nécessite un bon degré de coordination. C'est sur ce dernier aspect que se focalise le présent travail : la coordination dans un système multi-agents. L'objectif visé est que chaque entité réalise des actions adéquates de façon à accomplir une tâche individuelle ou commune. Des algorithmes permettent de calculer les meilleures actions à effectuer si l'environnement est connu à l'avance et est statique. Dans le cas d'un environnement incertain et dynamique, il devient plus difficile d'obtenir un comportement adéquat en toute circonstance. Ce travail se concentrera donc sur des agents, dépourvus de connaissances sur le monde extérieur, dont le but est d'apprendre par expérience. L'intelligence acquise ici est adaptative, les agents doivent utiliser les informations reçues de manière à se comporter correctement dans un environnement susceptible d'évoluer.

Ce type de coordination est nécessaire dans le cas où l'environnement n'est pas totalement prévisible. On peut citer par exemple le cas d'une équipe de robots devant explorer la planète Mars, ou encore des véhicules autonomes se déplaçant dans de mauvaises conditions climatiques. L'application de ce travail peut être généralisée à une multitude d'autres cas. L'incertitude est en effet toujours présente quel que soit le domaine d'investigation.

L'exposé du résultat de nos recherches est structuré de la manière suivante.

Le premier chapitre décrit les concepts théoriques utilisés dans ce travail et définit le vocabulaire propre à l'apprentissage dans un système multi-agents. Il comporte également un résumé de l'état de l'art en matière de coordination d'agents, ainsi qu'une comparaison des différentes méthodes présentées. Nous avons opté pour le *Q learning* qui permet de trouver une estimation de l'utilité des actions possibles dans chaque état. Cette estimation est construite à partir des récompenses reçues du monde exterieur. Si les récompenses sont fournies de façon adéquate, les agents apprennent à se coordonnner dans un environnement qui leur est initialement inconnu.

Le chapitre 2 détaille le problème sur lequel se concentre plus particulièrement ce mémoire. Il s'agit d'assurer la coordination de plusieurs agents dans un environnement bidimensionnel comportant des obstacles. Les agents doivent se déplacer de leur point de départ à leur objectif en empruntant le chemin le plus court tout en évitant les collisions. Le chapitre comporte une description précise de l'intégralité du problème, suivie des différents éléments de modélisation qui ont été mis en oeuvre pour le résoudre efficacement. Dans un premier temps, l'analyse s'est centrée sur le cas d'agents indépendants estimant leur fonction d'utilité individuelle en se basant uniquement sur leurs propres récompenses. Le cas des agents collaboratifs cherchant à estimer une fonction d'utilité jointe a ensuite été étudié. Les résultats expérimentaux obtenus en appliquant ces techniques existantes sont analysés en détail dans ce chapitre.

Les algorithmes étudiés possèdent chacun des inconvénients importants. Les agents cherchant à maximiser leurs récompenses individuelles n'arrivent pas à se coordonner efficacement. Ils ne tiennent en effet pas compte des autres entités évoluant dans l'environnement lors de leur apprentissage. Les agents collaboratifs se coordonnent correctement, mais le temps nécessaire à leur apprentissage croît exponentiellement avec le nombre d'agents. Notre contribution principale décrite dans le chapitre 3 remédie à ce problème. Il s'agit d'une amélioration de l'utilisation de l'algorithme *Q learning* qui réduit fortement le temps nécessaire à l'apprentissage dans le cas d'un grand nombre d'agents. Elle se base sur la fonction d'utilité jointe de deux agents et la généralise à un nombre quelconque d'entités. Ainsi, même si un nombre important d'agents utilise la fonction d'utilité obtenue, le temps d'apprentissage restera du même ordre que celui de deux agents collaboratifs. Les résultats expérimentaux de cette méthode présentés à la section 3.2 sont très prometteurs : pour un même nombre d'agents notre algorithme converge plus rapidement que celui basé sur l'estimation d'une fonction d'utilité jointe classique.

Enfin, dans l'optique d'une extension ultérieure de notre travail, le chapitre 4 propose différentes idées susceptibles d'améliorer les résultats actuels.

Chapitre 1

Concepts théoriques

Ce mémoire se situe dans le domaine du *multi-agent learning*, qui est l'intersection du *machine learning* et des *multi-agent systems*. Ces derniers sont tous deux des branches de l'intelligence artificielle. Informellement, le *multi-agent learning* peut être défini comme suit [19] :

> *Le multi-agent learning est l'apprentissage d'une tâche par plusieurs agents. Cet apprentissage est possible uniquement grâce à l'existence d'autres agents.*

La première section définit les éléments de vocabulaire propres au domaine abordé et qui seront employés dans les chapitres suivants. Suit une mise en perspective de divers travaux réalisés dans ce domaine afin à motiver le choix de l'algorithme. Enfin, une introduction au *Q learning*, l'algorithme sur lequel est basé la résolution du problème de la coordination entre agents, est réalisée.

1.1 Concepts de base

Il est important de clarifier et fixer les termes spécifiques utilisés dans le travail. Ci après, les notions d'agent et d'environnement sont présentées. Nous définissons ensuite des concepts propres au *machine learning* et aux *multi-agent systems*.

1.1.1 Agent

Le grand nombre de définitions de la notion d'agent provenant de la littérature scientifique témoigne de la difficulté à définir précisément ce concept. Parmi celles-ci, retenons :

- Un agent est défini comme une unité autonome interactive faisant partie intégrante d'un environnement dans lequel il se meut ; cet environnement peut contenir d'autres entités qui interagissent afin d'atteindre un but [13] ;
- Un agent est n'importe quelle entité qui observe son environnement via des capteurs et agit sur cet environnement via des actuateurs [11] ;
- Un agent est un système informatique, situé dans un environnement, capable d'exécuter des actions de manière autonome afin de réaliser les objectifs pour lesquels il a été créé [20].

Une tâche est généralement affectée à un agent. Un critère permettant de déterminer la qualité de réussite d'un agent est nécessaire. Il s'agit de la mesure de performance [11]. Par exemple, pour un agent *démineur* une mesure de performance est la quantité de bombes trouvées et désamorcées par heure.

Pendant toute sa durée de vie un agent effectue continuellement les trois fonctions suivantes, illustrées par la figure 1.1 :

- Observation (via les capteurs)
- Interprétation de ces perceptions et choix des actions possibles (via l'unité de traitement)
- Réalisation des actions choisies (via les actuateurs)

Cette suite d'actions permet à un agent d'agir sur son environnement en fonction des observations effectuées et de l'interprétation des résulats.

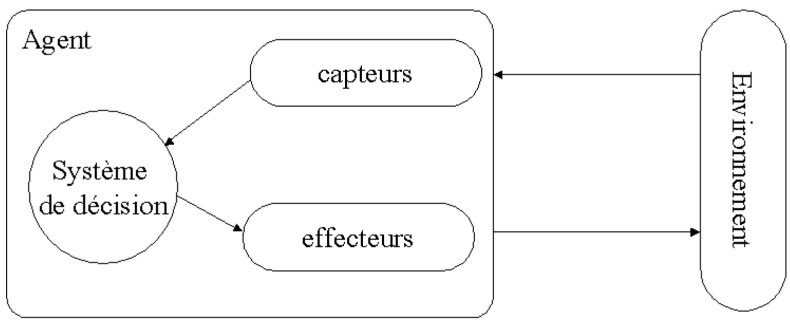


Figure 1.1: Architecture générique d'un agent.

Caractérisation des agents

Un certain nombre d'attributs permettent de caractériser le comportement d'un agent dans son environnement. Voici quelques propriétés pertinentes :

Rationalité [1] : capacité à agir dans l'optique d'accomplir sa mission, sans s'écarter de son objectif. Cette capacité dépend de la qualité et de la quantité d'informations à la disposition de l'agent.

Adaptation : capacité à apprendre, à s'améliorer avec l'expérience et à s'adapter à de nouvelles situations.

Autonomie : habilité à contrôler son propre comportement et à agir sans intervention humaine.

Réactivité : capacité à observer l'environnement et à agir en fonction des données reçues.

[1]Un agent rationnel idéal est une entité qui pour toutes les séquences d'observations possibles devrait exécuter des actions qui maximisent sa mesure de performance ; ce choix étant réalisé sur base des informations fournies par la séquence d'observations et les connaissances intégrées à l'entité [11].

Proactivité : capacité à prendre des initiatives pour satisfaire les objectifs, en anticipant les données futures.

Mobilité : capacité à se mouvoir d'un endroit à l'autre dans l'environnement.

Capacité à collaborer : habilité à travailler avec d'autres agents pour remplir un but.

Ainsi, un simple agent *thermostat* uniquement relié au système de chaufferie est autonome, réactif, non mobile, non adaptatif, non collaboratif et non proactif. Pour lui donner une certaine proactivité, il faudrait que l'agent connaisse un modèle de diffusion de la température, possède des capteurs pour détecter les ouvertures et fermetures de portes, etc. Le but du présent travail est de fournir un algorithme permettant à des agents mobiles de devenir adaptatifs, réactifs, proactifs et capables de collaborer.

1.1.2 Environnement

L'environnement dans lequel se déplacent les agents est important. Celui-ci fournit en effet les informations nécessaires aux capteurs du robot et ses propriétés déterminent les effets des actions des agents. On peut le définir comme un système duquel l'agent doit apprendre, au fil des expériences, les éléments nécessaires à la réalisation de son objectif. L'agent fait partie de ce système et peut le modifier.

Caractérisation de l'environnement

En toute généralité, un environnement peut être caractérisé par un certain nombre d'attributs. Chacun de ces systèmes possède donc un sous ensemble de ces propriétés :

Complètement observable vs. partiellement observable : un environnement est dit complètement observable si les capteurs de l'agent lui donnent accès à *l'état complet* de l'environnement. Dans le cas contraire, il est partiellement observable. Un environnement complètement observable simplifie le travail des agents car la quantité d'information inconnue est réduite. Mais cette hypothèse est rarement réaliste.

Déterministe vs. stochastique : un environnement est déterministe si le prochain état de l'environnement dépend *uniquement* de son état courant et de l'action de l'agent. La propriété déterministe réduit également la complexité de la tâche à résoudre car le résultat de chaque action est fixé. Un modèle peut donc facilement être construit.

Épisodique vs. séquentiel : un épisode est une séquence de paires $perception - action$. Dans un environnement séquentiel les décisions actuelles affectent les décisions futures, ce n'est par contre pas le cas de l'environnement épisodique où l'agent peut se contenter d'optimiser chaque épisode sans se soucier des incidences futures de ses actions. Chaque épisode étant en effet indépendant des autres.

Statique vs. dynamique : un environnement est dit statique si ses modifications ne résultent que des actions des agents. Un environnement dynamique comporte d'autres processus pouvant l'affecter, et peut donc être modifié indépendamment des actions des agents. Un monde statique peut donc être modélisé plus facilement car la durée du processus de décision des agents n'affecte pas le résultat, pour autant qu'elle soit identique pour chacun d'eux.

Discret vs. continu : un environnement ayant un nombre fini d'états et d'actions est discret. Par conséquent, un environnement continu comporte une infinité d'états ou d'actions.

Agent unique vs. multi-agents : cette propriété traduit la capacité à faire interagir plusieurs agents dans le même environnement. Cette interaction peut être compétitive ou collaborative. Les agents compétitifs essaient d'atteindre leur objectif individuellement, alors que les agents collaborants agissent dans un même but.

1.1.3 Définitions

Voici un ensemble de définitions de concepts fondamentaux tels qu'ils seront utilisés tout au long de ce travail.

Actions : ensemble des opérations que peut effectuer un agent afin de modifier son environnement. L'agent lui-même fait partie de cet environnement, il peut donc modifier sa propre position.

État : propriétés d'un agent caractérisant la partie de l'environnement qu'il occupe.

Agent : entité rationnelle qui, pour toutes les séquences d'observations possibles, exécute des actions qui maximisent sa mesure de performance. Ce choix est réalisé sur base des informations fournies par la séquence d'observations et les connaissances intégrées à l'entité [11].

Apprentissage : processus par lequel une entité améliore ses connaissances au fil des expériences. Une définition du machine learning est donnée à la section 1.1.4.

Communication : échange d'informations entre différents agents. Les informations échangées sont pertinentes dans le sens où le récepteur de l'information ne peut acquérir ces données tout seul. La communication peut être explicite ou implicite. Dans le cas d'une communication explicite, l'échange est effectué directement entre les agents. Dans celui d'une communication implicite, un agent récupère l'information à partir des modifications de l'environnement effectuées par d'autres agents.

Goal : état que l'agent doit apprendre à atteindre au fil des expériences.

Récompense : information reçue suite à une action.

Interaction : communication, action, perturbation de l'état ou récompense d'un agent due à l'environnement.

1.1.4 Machine Learning

Le machine learning traite de l'apprentissage de systèmes informatiques indépendants et peut être défini de la façon suivante [8]:

> *On dit d'un programme informatique qu'il apprend une classe de tâches T à partir d'une expérience E en se basant sur une mesure de performance P, si ses performances lors de la réalisation de tâches de T, mesurées par P, s'améliorent avec l'expérience E.*

Cette définition peut être rendue plus concrète en l'appliquant au cas d'un agent devant trouver le chemin le plus court d'un point à un autre, et ce dans un environnement particulier. Ainsi dans cet exemple :

- T est la classe des tâches qui consistent à trouver le chemin le plus court entre deux points ;
- P est la longueur du chemin obtenu ;
- E est le déplacement dans l'environnement.

Une présentation complète du *machine learning* peut être trouvée dans [8] et [11].

Reinforcement learning

Plus précisément, notre travail se situe dans une branche du machine learning nommée *reinforcement learning*. Ce dernier est une classe d'algorithmes qui déterminent la façon dont un agent autonome, pouvant observer et intéragir avec son environnement, apprend à choisir de manière rationnelle les actions optimales pour atteindre son but. Suite à chaque action, l'agent peut recevoir une récompense ou une pénalité qui lui indique si l'action est désirable ou non. Par exemple, un agent qui apprend à jouer aux échecs reçoit une récompense positive quand il gagne la partie, négative lorsqu'il la perd et nulle dans tous les autres états. L'agent doit donc apprendre, à partir de ces informations étalées dans le temps, à choisir la séquence d'actions produisant la plus grande récompense totale.

Le choix du reinforcement learning est motivé par la propriété de cette classe d'algorithmes qui permet à l'agent d'apprendre par lui-même un modèle de l'environnement. Avant de commencer, l'agent ne connaît ni son but ni les actions à éviter. Au fil de ses expériences, il apprendra les actions à faire pour maximiser ses récompenses. Le modèle de l'environnement est donc construit par l'agent lui-même. Dans le cas où le modèle est connu, un ensemble fixe de règles caractérisant le comportement de l'agent est suffisant. Ceci est un point important, puisqu' un comportement intelligent requiert souvent un nombre considérable de règles de contrôle. L'apprentissage est ici géré par un instructeur donnant les récompenses appropriées à l'agent.

Le reinforcement learning est également souhaitable dans le cas d'un environnement dynamique. En effet, les récompenses varient au cours du temps et l'agent s'adaptera progressivement pour apprendre les règles de transition de l' environnement. Dans le cadre de ce travail, l'algorithme utilisé est le *Q learning*. Il est présenté dans la section 1.3.

1.1.5 Le machine learning dans les systèmes multi-agents

Contrairement au machine learning classique, un agent est ici considéré non pas en tant qu'individu mais en tant qu'entité pouvant intéragir avec d'autres entités. En se basant sur [19], des caractéristiques spécifiques à l'apprentissage dans un système multi-agents sont présentées brièvement ci-dessous.

Catégories d'apprentissage

Dans un système multi-agents, l'apprentissage est *centralisé* s'il est réalisé dans son entièreté par un seul agent et qu'il ne requiert aucune interaction explicite avec les autres agents de l'environnement. Dans un processus centralisé un agent agit donc comme s'il était seul. Par contre, un apprentissage réparti au sein de plusieurs agents est dit *décentralisé*. Les agents peuvent avoir des capacités semblables ou différentes, et une interaction explicite entre agents est nécessaire pour accomplir leur objectif.

Systèmes multi-agents

Les *multi-agent systems* traitent de l'organisation et de la structure des agents dans un environnement. Informellement, il s'agit de systèmes dans lesquels plusieurs agents interagissent pour résoudre des problèmes et dont le but est de répondre à la question: *"Quand, comment interagir et avec qui?"*

Une présentation complète des systèmes multi-agents peut être trouvée dans [20].

En toute généralité, les systèmes multi-agents sont caractérisés par les propriétés suivantes :

- Chaque agent possède une information incomplète sur l'environnement ;
- Le contrôle des interactions est décentralisé ;
- Les informations sont décentralisées ;
- Les processus sont asynchrones.

Outre ces propriétés, trois aspects clés permettent de différencier les systèmes multi-agents [18]. Le premier est l'*environnement* occupé par le système, il a été présenté à la section 1.1.2.

Le second aspect concerne les caractéristiques des *agents*. Outre celles mentionnées à la section 1.1.1, une propriété importante est le nombre d'agents présents dans le système. On peut également analyser le nombre de goals de ces agents, ainsi que la *compatibilité* de ces goals entre eux. Ces goals peuvent êtres *contradictoires* ou *complémentaires*. Enfin, l'*uniformité* des agents est un critère essentiel. Ils peuvent être *homogènes* si les agents sont tous identiques, ou *hétérogènes* , s'ils ont des propriétés différentes.

Les *interactions* entre les agents constituent le dernier aspect. Elles peuvent avoir lieu entre ces derniers ou avec l'environnement, et diffèrent suivant plusieurs dimensions. Le premier élément à citer concerne le *type* des interactions. Entre agents, elles peuvent être *compétitives* ou *collaboratives*. Dans le premier cas, les agents cherchent à maximiser leurs performances individuelles tout en minimisant celles des autres, tandis que dans le second, leur but est d'obtenir le meilleur résultat global. Il faut ensuite mentionner la *raison* des interactions qui peuvent être dues au hasard ou *goal-orientées*, elles résultent alors de la volonté qu'ont les agents d'atteindre leur objectif.

La plupart des systèmes actuels sont distribués, larges, hétérogènes et complexes. Les systèmes multi-agents jouent et joueront un rôle important dans les technologies futures car ils fournissent un cadre théorique et concret permettant de modéliser et résoudre des problèmes complexes [18]. En effet :

- Le parallélisme, l'efficacité, la robustesse et la flexibilité de ces systèmes sont des propriétés très utiles dans les domaines complexes où des solutions centralisées sont impraticables ;
- Le concept de système multi-agents est en accord avec le principe qui stipule qu'intelligence et interaction sont intimement liées. Les systèmes multi-agents réalisent ainsi ce couplage dans les deux sens puisque d'une part, l'interaction des agents dans un tel système leur permet d'améliorer leur intelligence et, d'autre part, cette intelligence augmente leur capacité à interagir ;
- La technologie actuelle fournit une plate-forme solide pour la réalisation des éléments théoriques propres aux systèmes multi-agents.

Les caractéristiques évoquées ci-dessus sont les raisons pour lesquelles une structure multi-agents a été choisie pour ce travail.

La section suivante présente une sélection d'algorithmes utilisés pour la coordination d'agents. On y trouve également une comparaison entre les différentes méthodes et une motivation des choix en faveur de l'algorithme *Q learning*.

1.2 Etat de l'art

Cette section met en perspective une sélection de documents scientifiques liés à l'apprentissage de la coordination d'agents. Les algorithmes actuels de coordination dans un système multi-agents sont comparés afin de pouvoir déterminer lequel est le plus approprié dans le cadre du présent travail. Différents types d'interactions entre agents sont ensuite analysées, cet aspect étant un élément déterminant pour la qualité de l'apprentissage.

1.2.1 Algorithmes d'apprentissage

Reinforcement learning

Il existe deux schémas d'apprentissage. L'un est basé sur l'apprentissage de la fonction action-valeur et le second sur celui de la fonction d'utilité des états :

- Le premier consiste à apprendre en construisant une fonction d'utilité des états. Cette fonction renvoie l'utilité d'être dans un état particulier. Elle est ensuite employée pour effectuer les actions engendrant une utilité finale maximale ;
- Le second schéma consiste à apprendre la fonction action-valeur qui renvoie l'utilité d'une action particulière étant donné un état initial.

Le premier schéma nécessite un modèle de l'environnement, car le choix d'une action optimale nécessite la connaissance de l'état induit par l'action, afin de déterminer si l'état résultant est également optimal. L'apprentissage utilisant la fonction action-valeur ne nécessite par contre pas de modèle et il pourra être construit à l'aide de cette fonction.

Trois approches sont envisageables pour apprendre la fonction d'utilité:

- Méthode des moindres carrés (Least Mean Square ou LMS) : cette approche utilise les récompenses totales observées dans un état lors d'une séquence d'entraînement comme étant l'utilité de cet état. Le modèle de l'environnement est utilisé pour effectuer une décision lors du choix des actions. Le défaut de cette méthode est qu'elle ne tient pas compte de l'interconnexion des états. Les utilités de deux états adjacents ne sont en effet pas indépendantes ;
- Programmation dynamique adaptative (Adaptive Dynamic Programming ou ADP) : des statistiques du modèle sont utilisées afin de raffiner le calcul des utilités ;
- Différence temporelle (Temporal Difference ou TD) : cette méthode est basée sur le même principe que l'ADP, à l'exception des statistiques du modèle qui sont celles observées lors de l'apprentissage. Cela présente un avantage qu'aucun modèle ne soit nécessaire. Cependant, une connaissance préalable du modèle de l'environnement entraîne une convergence plus rapide.

Appelons la fonction action-valeur Q. Elle fournit une estimation (Q-valeur) de l'utilité d'une action pour un état donné. Ces Q-valeurs peuvent être apprises à partir de récompenses provenant de l'environnement. Elles permettent de prendre des décisions en l'absence de modèle. Le Q learning est un algorithme itératif qui permet l'apprentissage de ces Q-valeurs. Lors des épisodes d'apprentissage de l'algorithme, l'agent explore aléatoirement l'environnement. Si cette exploration est assez longue, un modèle peut être construit en même temps que la table $\hat{Q}$.

Une application de cet algorithme à la coordination d'une équipe de robots est présentée ci-dessous, suivie d'autres techniques de coordination de robots et d'une comparaison entre chaque méthode. Enfin, des améliorations du Q sont exposées.

Mise en oeuvre du Q learning

Le Q learning est utile lorsque le modèle du monde extérieur n'est pas connu du robot. Son efficacité est testée dans un scénario qui consiste en une équipe de robots qui doivent collecter un ensemble de mines et les rapporter à un endroit précis. Chacun des robots peut choisir entre trois rôles (*foreur*, *soldat* ou *mécanicien*) régissant ses actions.

L'algorithme est utilisé pour déterminer le rôle approprié d'un robot en fonction de son état. Des récompenses lui sont données quand il ramène une mine et lorsqu'il répare un autre robot immobilisé. L'efficacité d'une équipe est mesurée par le nombre d'itérations nécessaires à la collecte de toutes les mines. Il est montré que le nombre d'itérations décroît exponentiellement avec le nombre de robots. De plus, une équipe dotée du Q learning peut faire mieux qu'une équipe paramétrée manuellement, pour autant que les taux d'apprentissage et d'exploration soient adéquats.

Une description détaillée du fonctionnement du Q learning peut être trouvée à la section 1.3.

Case-Based Reasoning

Le Case-Based Reasoning (CBR) est une approche générale permettant à des agents de résoudre des problèmes spécifiques en se basant sur leur expérience [1, 9]. Les connaissances sont stockées dans une base de données sous forme de solutions de problèmes appelées *cases*. Un nouveau problème est résolu en utilisant une adaptation de problèmes similaires dont les solutions sont connues. La base de données contenant les solutions est interrogée à chaque fois que l'agent rencontre un nouveau problème. Les comportements des agents peuvent être définis sous forme d'une pondération de sous-comportements. Sur base des paramètres stockés, un nouveau jeu de paramètres définissant le comportement à adopter est renvoyé. Le CBR peut être utilisé, au même titre que le Learning Momentum, pour configurer en temps réel les paramètres d'un algorithme. Dans le cas de plusieurs agents, la base de données devient distribuée et les agents l'interrogent par envoi de messages.

Les étapes du CBR se synthétisent comme suit :

1. Retrouver des cas similaires en mémoire ;
2. Proposer une solution ;
3. L'adapter au cas présent ;
4. L'évaluer ;
5. Revenir à l'étape 3 ou enregistrer la solution finale.

Learning Momentum

Le Learning Momentum (LM) est une méthode de configuration automatique de paramètres. Il a été efficacement mis en oeuvre dans [2, 1] pour la sélection automatique de comportements prédéfinis. Le but était de faire apprendre une équipe de robots. En robotique, comme la plupart du temps l'environnement est inconnu des robots ou non modélisable (un terrain rempli d'obstacles par exemple), cet algorithme permet de calculer et d'ajuster en cours d'exécution les paramètres définissant le comportement d'un robot.

Le principe du LM consiste à modifier les paramètres d'un *behavior-based controller* (un robot agissant selon des comportements prédéfinis) en fonction des stimuli externes. Ces modifications ont pour objectif l'amélioration des performances de l'agent en question. Cette technique peut faciliter l'apprentissage des robots agissant selon des comportements prédéfinis en trouvant les paramètres optimaux. Le principe peut se résumer comme suit: le robot se comporte de la même manière tant que les performances sont bonnes, dans le cas contraire il est préférable de changer de comportement.

Le contrôleur doit être capable d'identifier la situation dans laquelle se trouve le robot et de maintenir une table bidimensionnelle dont la première dimension représente les situations possibles et la seconde les paramètres ajustables. Chaque entrée de la table contient la variation à ajouter aux paramètres correspondants. Le comportement final du robot est une somme pondérée des comportements de base.

L'étude qualitative a été réalisée en mettant en oeuvre une équipe de robots dans le même scénario que celui défini à la section sur le Q learning. Le scénario comporte trois types de robots, dont deux pour lesquels le comportement est totalement défini et un troisième dont le comportement global est à affiner en temps réel afin de maximiser les performances de l'équipe (robot LM). Ce dernier robot modifie ses paramètres en fonction de l'état dans lequel les autres robots se trouvent, de l'environnement et de son propre état.

Les résultats montrent qu'avec un nombre réduit de robots, les équipes dotées du LM sont plus efficaces que les équipes configurées manuellement. Cependant, avec un grand nombre de robots les équipes dotées uniquement de robots au comportement fixé sont meilleures. Cela s'explique par le fait que dans la situation d'abondance de ressources que sont les robots, il n'est pas nécessaire de placer ces robots de manière stratégique.

Comparaison des algorithmes d'apprentissage

Plusieurs méthodes pour l'apprentissage de la coordination par plusieurs agents ont été présentées précédemment. Pour ce mémoire le choix se porte sur le Q *learning* car il permet un apprentissage sans modèle. Il n'est donc pas nécessaire de communiquer à l'agent des règles définissant son comportement. Cela s'avère très utile puisque le nombre de règles à définir pour assurer un comportement complexe et adaptatif peut s'avérer très grand dans le cas de la coordination de plusieurs agents.

En outre, le Q learning permet aux agents de s'adapter à un environnement qui change dans le temps. Dans le cas de règles d'apprentissage ou de sous-comportements définis, il peut être nécessaire de les modifier si l'environnement varie. Dans le cas du Q learning, par contre, les agents verront leurs récompenses et leurs états changer et des entrées de la table

$\hat{Q}$ seront progressivement ajoutées ou modifiées. Les agents passeront ainsi d'une politique optimale à une autre pour autant que le nombre d'apprentissage dans le nouvel environnement soit suffisant. Il est nécessaire dans ce cas d'utiliser la version non-déterministe de l'algorithme qui peut s'adapter à un environnement changeant. Il s'agit du second avantage du Q learning : il permet d'obtenir des agents adaptatifs.

1.2.2 Améliorations du Q learning

La section 1.3.6 comporte également quelques idées mentionnées en [8] permettant de perfectionner l'algorithme Q learning de base. Un ensemble d'améliorations plus générales sont décrites dans cette section.

Apprentissage de primitives de haut niveau

La complexité du Q learning augmente exponentiellement avec le nombre d'états qui caractérisent l'agent. Une technique pour améliorer son efficacité tout en diminuant le nombre d'états possibles est l'apprentissage de primitives de haut niveau. Le Q learning peut être utilisé pour l'apprentissage de comportements prédéfinis [7, 6]. Il est aussi possible d'apprendre des comportements de bas niveau et d'utiliser un second algorithme d'apprentissage pour les coordonner afin d'obtenir des comportements plus complexes (Hierarchical Reinforcement Learning). La granularité des actions possibles est alors moins fine que dans le cas de primitives de bas niveau. Le Q learning a été efficacement utilisé comme mécanisme d'échange de rôle (*role-switching*), dans le cadre d'un scénario de forage où chaque robot apprend le rôle qu'il doit remplir en fonction des stimuli externes [6].

Le but visé par une telle approche est de diminuer le nombre d'états possibles et par conséquent le temps de calcul de l'algorithme, en regroupant les actions sous forme de comportements. Ces primitives peuvent être par exemple *patrouiller*, *intercepter*, etc.

La politique optimale issue de l'apprentissage d'une équipe de deux robots a été testée sur de vrais robots, le comportement résultant fut concluant. Les résultats montrent la capacité du Q learning à coordonner une équipe de robots sans communication directe en utilisant des primitives de haut niveau (*role-switching* ou *behavior selection*). Ils soulignent également l'importance d'une bonne attribution des récompenses.

L'algorithme a également été testé en présence d'obstacles. Il a été montré que le nombre de simulations nécessaires à la convergence augmente exponentiellement avec le pourcentage de surface couverte par des obstacles. Au delà de 30% d'obstacles, les performances deviennent mauvaises. Une implémentation réelle a été effectuée, dans laquelle le robot a appris en quinze itérations à coordonner ses différentes actions en temps réel.

Il est également possible d'utiliser le Q learning hiérarchique en divisant un problème donné en sous-problèmes et autant d'agents chargés de les résoudre [4]. Il faut ajouter un agent principal destiné à apprendre la fonction $Q(s, i)$, où i est l'agent à choisir dans l'état s. Cette dernière fonction est moins complexe que la fonction $Q(s, a)$ traditionnelle étant donné que l'espace des actions est grandement réduit. Les résultats des expériences obtenus avec cette méthode sont meilleurs que ceux obtenus avec le Q learning traditionnel.

Estimateur de progrès

Dans le Q learning classique, les récompenses sont attribuées quand le goal est atteint. Il en est de même pour les punitions qui sont attribuées quand une mauvaise décision est prise. Une nouveauté intéressante est l'utilisation de deux types de récompenses : des estimateurs de progrès et des fonctions de récompense hétérogènes. Ces dernières fournissent des récompenses en fonction des évènements perçus par les capteurs et des états des robots. Les estimateurs de progrès sont des fonctions continues dans le temps qui mesurent le chemin parcouru pour atteindre un objectif. Une de ces fonctions peut par exemple retourner la distance à une cible. Ce type de récompense rend l'algorithme plus robuste pour plusieurs raisons :

- Les estimateurs de progrès sont moins dépendants du bruit des capteurs car ils introduisent une certaine connaissance du domaine ;
- Ils permettent d'encourager l'exploration en achevant l'apprentissage d'un sous-comportement qui doit encore progresser. Sans ces estimateurs le robot pourrait choisir des comportements alternatifs sans terminer aucun apprentissage de sous-comportement ;
- Ils permettent également de relativiser les récompenses dues à des succès engendrés par des coincidences.

Les résultats expérimentaux montrent que l'apprentissage combinant les fonctions hétérogènes et les estimateurs de progrès est plus efficace que le Q learning traditionnel. Cela s'explique par le fait que quand le comportement d'un robot dépend de celui des autres, le Q learning de base éprouve des difficultés à trouver une séquence ordonnée de comportements efficaces.

Approximation de la fonction Q

L'expérience réalisée en [4] comporte un robot doté de 7 senseurs pouvant prendre une dizaine de valeurs différentes ainsi que réaliser une dizaine d'actions. L'espace état-action comporte un comporte des millions d'entrées. Un premier problème est la mémoire requise pour stocker ces données. A cela s'ajoute le problème du temps nécessaire à la visite de toutes ces entrées lors de l'application de l'algorithme. Une solution est de représenter la fonction Q par un réseau de neurones et non par une table. Si le nombre d'actions est réduit, il est efficace d'associer à chaque action un réseau prenant un état en entrée et une Q-valeur en sortie. Ainsi, il suffit de comparer la sortie des réseaux pour obtenir la Q valeur maximale. Afin que les réseaux puissent mieux séparer les entrées, chaque valeur des capteurs est convertie en binaire et les valeurs d'entrée sont réduites à 0 ou 1. Dans l'expérience réalisée, cela revient à 57 entrées pour chaque réseau. Une couche cachée comportant 10 neurones a fourni les meilleurs résultats.

Contrairement aux réseaux de neurones classiques, l'apprentissage ne se fait pas sur des exemples d'entrée/sortie corrects (état s, $Q(s, a)$), mais sur les estimations successives de Q.

1.2.3 Interactions entre agents

Suite à l'analyse des différents algorithmes d'apprentissage, il est important de s'intéresser à la modélisation de la communication entre agents ainsi qu'à leur coopération. Ces aspects sont en effet essentiels pour obtenir un apprentissage de qualité en un temps réaliste.

Techniques de communication

Il existe différentes manières de communiquer entre des agents apprenant un comportement optimal [15] :

- Les agents se communiquent des informations instantanées, par exemple des triplets (état, action, récompense) ;
- Les agents se communiquent des séquences de ces triplets, des parcours de l'état initial à l'état final par exemple ;
- Les agents se communiquent la fonction définissant le comportement appris (politique de déplacement).

Les expériences réalisées dans [15] utilisent le Q learning afin de faire apprendre à des chasseurs à trouver leur proie dans une grille de déplacement carrée. Dans un premier temps, on considère qu'un chasseur peut capturer sa proie seul.

Le premier test est réalisé avec un chasseur doté d'un éclaireur qui lui transmet ce qu'il voit. Malgré l'augmentation de l'espace d'état-action du chasseur, les résultats sont meilleurs qu'en l'absence de l'éclaireur.

Une autre alternative est que deux chasseurs utilisent la même fonction de décision, qui est ainsi successivement mise à jour par les deux agents. Les résultats obtenus sont meilleurs que dans le cas de deux chasseurs indépendants.

Il se peut cependant que les chasseurs puissent avoir des comportement complémentaires et apprendre des parties différentes de l'espace d'état-action. Des tests ont été réalisés avec des agents remplaçant leur fonction de décision par la moyenne de celles de tous les agents, a une certaine fréquence. A nouveau la convergence est plus rapide que dans le cas d'agents indépendants mais néanmoins un peu plus lente que dans celui d'agents utilisant la même fonction. On peut noter que la fréquence de mise à jour optimale dépend du champ de vision des chasseurs. De plus, si la taille de la fonction de décision et la fréquence de mise à jour sont trop gandes, les coûts de communication deviennent très importants.

Une autre alternative est qu'une fois la proie capturée, les agents se transmettent un épisode contenant tous leurs déplacements, afin que l'agent partenaire puisse rejouer cet épisode et l'apprendre. Les agents doublent ainsi leur apprentissage. Le coût de communication est relativement faible comparé à l'échange de fonction de décision. De plus, cet échange peut se faire entre agents hétérogènes, tant qu'ils sont capables d'interpréter les mêmes épisodes. Enfin, un agent pourrait apprendre d'un “expert” qui aurait déjà appris la tâche. C'est cette dernière technique qui donne les meilleurs résultats.

Le cas d'une action conjointe est enfin analysé. Les deux chasseurs doivent capturer la proie en même temps. Pour résoudre ce problème on étend l'espace d'état d'un chasseur en incluant la position relative du partenaire de même que ce qu'il perçoit. Malgré un espace d'état bien plus grand et donc un apprentissage plus lent, les résultats après un certain nombre d'itérations sont meilleurs que pour deux agents indépendants.

Stratégies de coordination

Dans le cadre du Q learning, différentes stratégies de choix d'action sont analysées en [5]. Elles concernent le cas de plusieurs agents coopératifs, indépendants, et n'ayant pas connaissance des actions des autres.

Dans le cas de deux agents, le but de chacun est de trouver l'action qui lui fournit la plus grande récompense, sachant que la valeur de celle-ci dépend également de l'action de l'autre agent. Les récompenses associées aux actions combinées des agents peuvent être représentées par une table où chaque colonne correspond à une action particulière du premier agent, et chaque rangée à une action du second.

Le premier test effectué se base sur le "jeu de l'escalade" qui est caractérisé par une forte pénalité en cas de non-coordination. L'agent reçoit ainsi une récompense négative s'il choisit l'action optimale alors que l'autre choisit la mauvaise. Le second scénario de test est le "jeu de la pénalité" dans lequel plusieurs actions peuvent conduire à une récompense jointe optimale. Toutefois, les agents sont fortement pénalisés s'ils ne s'accordent pas sur la même action.

Dans le cas d'une stratégie de coordination optimiste, l'agent choisit une action en supposant que l'autre sélectionne la meilleure action correspondante. Cette stratégie permet de résoudre les deux problèmes mentionnés mais pas dans leur version stochastique. La version stochastique est caractérisée par le fait qu'une action peut donner lieu à des récompenses différentes. La récompense moyenne tend cependant vers une valeur fixe, ce qui permet de comparer ce type de scénarios aux précédents. La non-convergence est causée par le fait que, pour une action donnée, la stratégie optimiste tiendra compte de la récompense la plus élevée et non de la valeur moyenne.

Une alternative qui permet de résoudre les problèmes partiellement stochastiques est l'utilisation de l'heuristique *Frequency of the Maximum Q-value* (FMQ) [5]. Au lieu de se baser directement sur les Q-valeurs pour choisir l'action appropriée, la stratégie FMQ prend en compte la fréquence à laquelle une action a produit sa récompense maximale. Pour une action donnée, le critère de sélection est remplacé par une somme pondérée de sa Q-valeur et du produit entre sa récompense maximale et le nombre de fois qu'elle a été obtenue.

Les tests réalisés montrent que la stratégie FMQ permet de résoudre le problème partiellement stochastique, mais pas le problème stochastique complet, dans lequel chaque action peut engendrer des récompenses différentes.

1.2.4 Conclusion

Divers algorithmes d'apprentissage de la coordination ont été analysés dans cette section. Pour ce travail c'est le Q learning qui a été choisi. Les raisons de ce choix sont les suivantes :

1. Il ne nécessite pas de modèle de l'environnement ;
2. Sa simplicité de mise en oeuvre permettra de se focaliser sur la qualité de l'apprentissage ;
3. Des résultats prometteurs ont déjà été obtenus avec cette technique.

Plusieurs améliorations du Q learning utilisées pour la coordination ont été présentées. L'aspect de la communication a également été abordé et il en découle qu'un apprentissage de la coordination basé sur des actions jointes donne de meilleurs résultats, dans le cas d'agents coopératifs, que celui basé sur actions individuelles. La section suivante présente l'algorithme Q learning de base.

1.3 Introduction au Q learning

L'algorithme Q learning ainsi que le développement mathématique qui permet de le comprendre sont présentés ci-après. Le contenu est basé sur le chapitre 13 de [8]. Dans un premier temps, une énumération des hypothèses sur lesquelles se base le raisonnement est présentée. Suit alors la démarche mathématique qui mène à l'algorithme dans le cas déterministe [16]. La question de la convergence sera ensuite traitée, suivie d'un exemple d'exécution de l'algorithme. Par après, une analyse des améliorations possibles ainsi que des différentes politiques d'exploration sera effectuée. L'adaptation de l'algorithme Q learning au cas d'un environnement non-déterministe cloturera la section.

1.3.1 Les hypothèses

Un agent est défini à la section 1.1.1 comme étant une entité pouvant observer et agir sur son environnement à l'aide d'un ensemble d'actions possibles. Plusieurs hypothèses sont nécessaires à l'utilisation du Q learning dans un système comprenant un ou plusieurs agents :

- L'ensemble S contient les états possibles de l'agent, et l'ensemble A les actions que l'agent peut effectuer. Pour un temps t, l'agent va analyser son état courant s_t appartenant à S. Il choisit alors une action a_t à effectuer dans l'ensemble A ;
- Lors de chaque action, l'environnement renvoie à l'agent une récompense $r(s_t, a_t)$ qui peut être un nombre négatif, nul ou positif. L'agent prend également connaissance de son état résultant $s_{t+1} = \delta(s_t, a_t)$;
- Les fonctions r et δ ne sont pas forcément connues de l'agent. Ce dernier ne possède pas de modèle de l'environnement et ne sait donc pas vers quel état une action particulière va le mener ;
- Le problème doit être modélisé sous la forme d'un Processus de Décision Markovien (communément appelé Markov Decision Process ou MDP). Ce qui signifie que les fonctions $\delta(s_t, a_t)$ et $r(s_t, a_t)$ dépendent uniquement de l'état et de l'action effectuée au temps t, et non des états et actions antérieures ;
- Pour commencer, le cas d'un MDP déterministe est envisagé. C'est-à-dire que l'exécution d'une action particulière dans un état donné engendre toujours la même récompense et le même état résultant.

1.3.2 Objet de l'apprentissage

La tâche de l'agent est d'apprendre une politique $\pi : S \to A$ lui indiquant quelle action choisir lorsqu'il se trouve dans un état particulier. Il doit trouver la politique π qui engendre la plus grande récompense cumulée au cours du temps. La récompense cumulée $V^\pi(s_t)$ d'une politique arbitraire π est définie comme suit :

$$\begin{aligned} V^\pi(s_t) &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots \\ &= \sum_{i=0}^{\infty} \gamma^i r_{t+i} \end{aligned} \tag{1.1}$$

La séquence de récompenses r_{t+i} provient de l'application de la politique π à partir de l'état s_t, et ce i fois. Le facteur γ est une constante déterminant la valeur relative d'une récompense reçue au temps suivant par rapport à une récompense immédiate ($0 \leq \gamma < 1$). Une récompense obtenue après i périodes de temps sera donc diminuée d'un facteur γ^i lors de sa prise en compte au temps initial. Au plus γ sera proche de 1, au plus les récompenses futures auront de l'importance par rapport à ce que l'agent reçoit immédiatement. Ce facteur provient du fait qu'il est préférable de recevoir une récompense au temps présent que par après.

L'agent doit donc apprendre la politique π qui maximise $V^\pi(s)$ pour tout état s. Il s'agit de la *politique optimale* π^* définie comme étant :

$$\pi^* = \operatorname*{argmax}_{\pi} V^\pi(s), (\forall s) \tag{1.2}$$

Pour simplifier la notation, la récompense cumulée maximale $V^{\pi^*}(s)$ qu'un agent peut recevoir à partir d'un état s sera notée $V^*(s)$. Cette dernière est obtenue en suivant la politique optimale à partir de l'état s.

L'agent doit apprendre la politique optimale $\pi^* : S \rightarrow A$ à partir des récompenses immédiates $r(s_t, a_t)$. La meilleure action a à effectuer dans un état s est celle qui maximise la somme de la récompense immédiate $r(s, a)$ et de la récompense cumulée maximale à partir de l'état résultant. Ce dernier terme est multiplié par le facteur γ car la récompense de l'état résultant se situe dans le futur. Ainsi

$$\pi^*(s) = \operatorname*{argmax}_{a}(r(s,a) + \gamma V^*(\delta(s,a))) \tag{1.3}$$

L'agent peut donc connaître la politique optimale π^* s'il a une connaissance parfaite des fonctions de récompense r et de transition d'état δ , mais ce n'est pas le cas du problème détaillé ici. Il est donc nécessaire de définir une fonction d'utilité $Q(s, a)$ qui renvoie la récompense cumulée maximale pouvant être obtenue si on effectue l'action a dans l'état s :

$$Q(s,a) = r(s,a) + \gamma V^*(\delta(s,a)) \tag{1.4}$$

L'équation 1.3 peut donc être reformulée de la façon suivante :

$$\pi^*(s) = \operatorname*{argmax}_{a} Q(s,a) \tag{1.5}$$

Pour suivre la politique optimale, l'agent doit donc choisir dans l'état s l'action a qui maximise la fonction Q. Le but de l'algorithme du Q learning consiste à estimer cette fonction.

1.3.3 Algorithme de base

Le Q learning estime la fonction Q par approximation itérative. A partir des équations 1.2 et 1.5 il se déduit que la récompense cumulée maximale pour un état s est la plus grande valeur que peut prendre la fonction Q dans cet état. Cette valeur maximale correspond à une action particulière a'.

$$V^*(s) = \max_{a'} Q(s, a') \tag{1.6}$$

L'équation 1.4 peut donc être écrite sous la forme

$$Q(s, a) = r(s, a) + \gamma \max_{a'} Q(\delta(s, a), a') \tag{1.7}$$

Cette définition récursive de la fonction Q est à la base du Q learning, car elle permet l'approximation itérative. Désormais, le symbole $\hat{Q}$ désignera l'estimation de la fonction Q lors de l'apprentissage. Pour représenter l'estimation $\hat{Q}$, l'algorithme utilise une table où chaque entrée est représentée par une paire d'état-action (s, a). Chacune des entrées (s, a) contient l'estimation actuelle $\hat{Q}(s, a)$ de la véritable valeur de la fonction Q. La valeur d'une entrée est appelée *Q-valeur*.

La table peut être initialisée avec des valeurs aléatoires ou nulles. L'agent va successivement observer son état courant s, choisir une action a, l'exécuter et enfin observer la récompense $r(s, a)$ obtenue et l'état résultant $s' = \delta(s, a)$. Il va ensuite mettre à jour l'entrée (s, a) dans la table selon la formule suivante :

$$\hat{Q}(s, a) \leftarrow r(s, a) + \gamma \max_{a'} \hat{Q}(s', a') \tag{1.8}$$

Cette règle de mise à jour est basée sur l'équation 1.7, mais la valeur de la fonction Q pour l'état résultant s' est remplacée par l'estimation actuelle $\hat{Q}(s', a')$. L'agent n'a donc pas besoin de connaître les fonctions $\delta(s, a)$ et $r(s, a)$ pour modifier la table. La table 1.1 récapitule l'algorithme de base du Q learning proposé par Watkins [16], dans le cas d'actions et de récompenses déterministes.

Algorithme du Q learning

- Pour chaque état s et action a, initialiser l'entrée de la table $\hat{Q}(s, a)$ à zéro.
- Observer l'état initial s
- Répéter infiniment:
  1. Sélectionner une action a et l'exécuter
  2. Recevoir une récompense immédiate r
  3. Observer le nouvel état s'
  4. Mettre à jour l'entrée de la table $\hat{Q}(s, a)$ de la façon suivante:

     $$\hat{Q}(s, a) \leftarrow r(s, a) + \gamma \max_{a'} \hat{Q}(s', a')$$

  5. Remplacer l'état s par l'état résultant s'

Table 1.1: Algorithme du Q learning dans le cas d'actions et récompenses déterministes.

1.3.4 Convergence

Il a été démontré que l'algorithme proposé dans la table 1.1 converge bien vers la fonction Q sous certaines conditions [17]. Ces conditions de convergence sont les suivantes :

- Le problème est modélisable par un MDP déterministe ;
- Les valeurs des récompenses immédiates sont bornées. Cela signifie qu'il existe une constante positive c telle que que pour tout état s et action a, $|r(s, a)| < c$;
- L'agent sélectionne ses actions de façon à visiter chaque paire état-action possible infiniment souvent. Cela veut dire que lorsque la séquence d'actions effectuée tend vers l'infini, pour toute action a valide dans un état s l'agent répétera a à partir de s avec une fréquence non-nulle.

Cette dernière condition est la plus restrictive, les différentes approches pour tenter de la remplir seront abordées ultérieurement. Une intuition de la démonstration du théorème de convergence peut cependant être esquissée. L'idée principale est que l'entrée de la table $\hat{Q}(s, a)$ comportant la plus grande erreur voit son erreur réduite d'un facteur γ lorsqu'elle est mise à jour. Ceci provient du fait que la nouvelle valeur est la somme de la récompense immédiate sans erreur et de l'estimation erronée $\hat{Q}$ réduite par un facteur γ.

1.3.5 Exemple

Cette section illustre l'algorithme d'apprentissage à l'aide d'un environnement représenté par une grille de taille 2×3. L'agent est initialement positionné en $(1, 1)$. Lors de chaque déplacement il effectue une action aléatoire et reçoit une récompense de 100 quand il arrive à son goal g en $(1, 3)$. Le goal est ici un état absorbant, c'est-à-dire qu'une fois son goal atteint l'agent ne peut plus faire d'action. Pour poursuivre l'apprentissage, il convient donc de réinitialiser l'agent à sa position initiale et de recommencer le parcours de l'environnement. Les actions possibles sont les déplacements vers le haut, le bas, la gauche et la droite. La facteur de réduction γ est fixé à 0.9. La figure 1.4 décrit la politique optimale à trouver. Dans ce schéma, chaque flèche correspond à une entrée (s, a) dans la table et la valeur associée est celle de l'estimation $\hat{Q}$ pour cette entrée.

Initialement, les valeurs de la table $\hat{Q}(s, a)$ sont fixées à zéro. L'agent va parcourir l'environnement en exécutant un série d'actions successives. Avant d'atteindre son goal, il obtient lors de chaque déplacement une récompense immédiate nulle. Il laisse donc les entrées de la table à zéro. Lorsque l'agent est dans un état s_1 adjacent au goal et qu'il effectue une action a_1 le menant à ce goal g, il reçoit une récompense immédiate de 100. Il remplace alors la valeur nulle de l'entrée $\hat{Q}(s_1, a_1)$ par 100. Seule la récompense $r(s_1, a_1)$ est prise en compte car le goal est un état absorbant sans récompense future. La figure 1.2 illustre les valeurs de la table $\hat{Q}$ après le premier épisode, dans le cas où ces valeurs ont été initialisées à zéro.

L'agent recommence ensuite un épisode à partir de sa position initiale. Tant que le robot ne se retrouve pas dans un état adjacent à g ou s_1, les valeurs de la table $\hat{Q}$ restent nulles. Si l'agent arrive dans un état s_2 adjacent à s_1 et qu'il effectue une action a_2 le menant en s_1, il reçoit toujours une récompense immédiate r nulle. Mais comme la valeur de la table $\hat{Q}$ pour l'état s_1 et l'action a_1 est de 100, l'entrée correspondant à l'état s_2 et l'action a_2 sera mise à jour de la façon suivante :

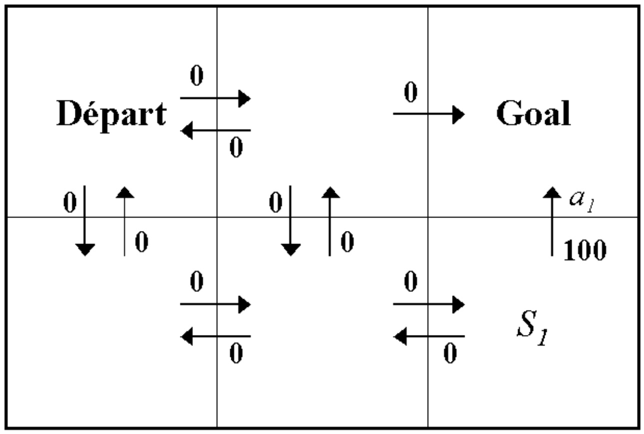


Figure 1.2: Politique obtenue après le premier épisode, avec $\gamma = 0.9$ et $r(s,a) = 100$ pour tout s et a tels que $\delta(s,a) = g$ et $r(s,a) = 0$ sinon. Les valeurs de la table $\hat{Q}$ sont initialisées à zéro. La valeur de l'action a_1 dans l'état s_1 menant au goal est fixée à 100.

$$
\begin{aligned}
\hat{Q}(s_2, a_2) &= r(s_2, a_2) + 0.9 \max_{a'} \hat{Q}(s_1, a') \\
&= 0 + 0.9 \max(100, 0) \\
&= 90
\end{aligned}
$$

La figure 1.3 représente l'état de la table $\hat{Q}$ à la suite de cet épisode.

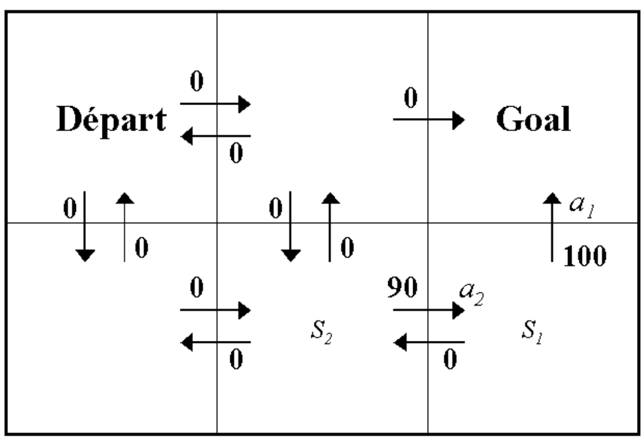


Figure 1.3: Politique obtenue après le second épisode, la valeur de l'action a_2 dans l'état s_2 est fixée à 90, les autres restent nulles.

Après un certain nombre d'épisodes, l'agent aura visité un nombre suffisant de paires état-action et la table $\hat{Q}(s,a)$ contiendra les valeurs correspondant à la politique optimale réprésentée par les flèches de la figure 1.4. Une fois cette politique optimale déterminée, l'agent peut se déplacer de façon à maximiser ses récompenses futures. Il suffit pour cela qu'à chaque état s l'agent effectue l'action a telle que la valeur de $\hat{Q}(s,a)$ dans la table soit maximale. On constate que dans le cas de la figure 1.4, l'agent en $(1,1)$ va d'abord se déplacer vers la droite (la valeur de l'action *droite* est de 90 contre 72 pour l'action *bas*) puis une nouvelle fois vers la droite (valeur de 100 pour l'action *droite* contre 81 pour *bas* et *gauche*), atteignant ainsi son goal en seulement deux déplacements.

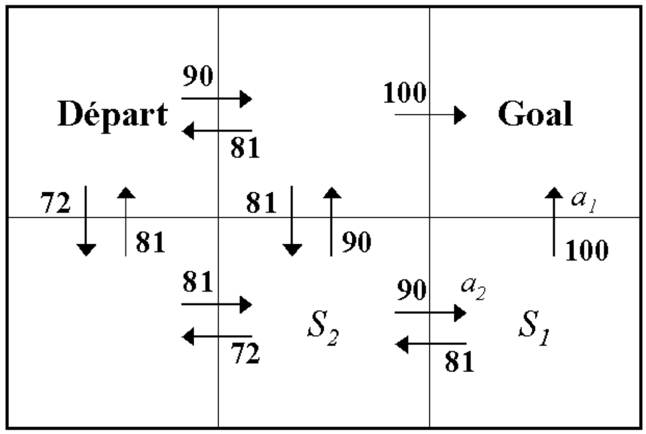


Figure 1.4: Politique optimale pour cet environnement avec $\gamma = 0.9$ et $r(s,a) = 100$ pour tout s et a tels que $\delta(s,a) = goal$ et $r(s,a) = 0$ sinon.

1.3.6 Améliorations

Une hypothèse du théorème de convergence est que chaque paire état-action doit être visitée une infinité de fois. Par conséquent une séquence d'actions aléatoires peut convenir. On peut de même changer l'ordre dans lequel les déplacements sont effectués sans affecter la convergence. L'exemple de la section 1.3.5 montre que lors du premier épisode, seule une entrée de la table $\hat{Q}(s,a)$ est modifiée, de même durant l'épisode suivant au maximum deux entrées sont modifiées.

Il est possible de faire converger l'algorithme plus rapidement en stockant les déplacements d'un épisode et en inversant l'ordre chronologique des mises à jour de la table $\hat{Q}(s,a)$. Lors du premier épisode de l'exemple de la section 1.3.5, on peut commencer par mettre à jour l'entrée (s_1, a_1) correspondant au dernier déplacement. On remplace alors la valeur nulle de cette entrée par 100. Si on met ensuite à jour l'entrée de l'avant-dernier déplacement, on constate que l'on remplace la valeur nulle de cette entrée par 90. Ceci provient du fait que la table $\hat{Q}$ contient déjà la valeur 100 ajoutée précédemment. En effectuant toutes les mises à jour dans l'ordre inverse du parcours, on obtient des valeurs de la table non nulles pour chaque action effectuée lors de l'épisode, et non plus uniquement pour la dernière. L'algorithme va ainsi converger plus rapidement vers la politique optimale, mais cela nécessite que l'agent stocke tous les déplacements lors de chaque épisode.

Dans le cas où le coût des opérations consistant à effectuer une action et observer le nouvel état est important, il peut être intéressant de ré-entraîner périodiquement l'agent sur des épisodes stockés. Cela pourrait sembler superflu, mais il faut se rappeler que la mise à jour d'une entrée dépend des valeurs de $\hat{Q}(s', a')$ de l'état résultant s'. Si la valeur de la meilleure action a' dans l'un de ces états résultants a changé entre-temps, le nouvel apprentissage à partir du même épisode modifiera les entrées menant à l'état s'.

1.3.7 Politiques d'exploration

Différentes politiques de choix des actions successives à effectuer lors de l'apprentissage sont envisagées. Deux d'entre elles sont présentées ci-dessous.

Mouvements aléatoires

Il s'agit du système de choix le plus simple. L'agent choisit une action aléatoirement parmi ses actions possibles. Cette politique d'exploration est très facile à mettre en oeuvre et permet de visiter toutes les paires d'état-action infiniment souvent pour un nombre infini d'itérations. Cependant, le nombre de déplacements nécessaires à l'agent pour trouver son goal peut être très grand. On en dénombre plusieurs centaines pour un environnement de taille 10×10. Cette méthode peut nécessiter un temps d'exploration très long puisque l'agent doit trouver son goal aléatoirement lors de chaque épisode de l'apprentissage.

Approche probabiliste

Une autre méthode consiste à tenir compte de l'estimation $\hat{Q}$ obtenue au cours de l'apprentissage pour séléctionner dans un état s une action a maximisant $\hat{Q}(s, a)$. L'agent apprend ainsi à partir de parcours plus proches de l'optimalité que dans le cas aléatoire. Ceci réduit par conséquent le temps nécessaire pour trouver son goal lors de chaque épisode.

Un inconvénient de cette technique est que, si l'agent choisit à chaque fois l'action maximisant l'estimation $\hat{Q}$, il ne visitera pas toutes les paires état-action possibles et effectuera continuellement le même parcours sous-optimal. Une solution est d'utiliser une approche probabiliste, dans laquelle la probabilité de choisir une action est proportionnelle à l'estimation de sa valeur $\hat{Q}$. Il faut cependant que toutes les actions aient une probabilité non-nulle d'être choisies. Il s'agit là de trouver un compromis entre l'*exploitation* des estimations $\hat{Q}$ connues et l'*exploration* des paires état-action non-visitées. Ce problème peut être résolu en assignant à chaque action une probabilité suivant la fonction de distribution de Boltzmann (également appelée *soft max*). La probabilité $P(a_i|s)$ de choisir dans un état s une action a_i parmi j actions possibles est alors définie de la façon suivante [16, 14] :

$$P(a_i|s) = \frac{e^{\frac{\hat{Q}(s,a_i)}{T}}}{\sum_j e^{\frac{\hat{Q}(s,a_j)}{T}}} \tag{1.9}$$

Le terme $e^{\frac{\hat{Q}(s,a_i)}{T}}$ est toujours positif, et la somme des probabilités de choix de chaque action $\sum_j P(a_j|s)$ est égale à 1. Le facteur T représente la *température*, elle permet de contrôler l'exploration par rapport à l'exploitation des estimations $\hat{Q}$. Si T est élevée, les écarts entre les expressions $e^{\frac{\hat{Q}(s,a_i)}{T}}$ pour les actions possibles a_i seront moins grands que dans le cas d'une faible température. L'effet des grandes valeurs de $\hat{Q}$ sera donc atténué. On constate que pour $T = \infty$ on revient dans le cas du choix aléatoire. Une température nulle correspond à la situation de l'exploitation pure : l'agent choisit les actions ayant une valeur $\hat{Q}$ maximale. Il convient dès lors de faire décroître la température vers zéro lors de chaque itération. Ceci favorise la visite de toutes les paires état-action en début d'apprentissage pour bénéficier par après des connaissances acquises, dans le but d'accélérer l'exécution de l'algorithme.

1.3.8 Q learning non-déterministe

Jusqu'à présent, seul le cas où les récompenses et les états résultants sont observés de façon déterministe a été présenté. Si à deux moments différents, une action a effectuée dans un état s peut donner lieu à des récompenses et des états résultants différents, le problème doit être modélisé par un MDP non-déterministe.

Pour une politique π donnée, l'agent ne doit plus apprendre la récompense cumulée V^π définie en 1.1 mais la moyenne des récompenses cumulées reçues en suivant cette politique un nombre infini de fois. L'équation suivante est donc une généralisation de l'équation 1.1.

$$V^\pi(s_t) = E\left[\sum_{i=0}^{\infty} \gamma_i r_{t+i}\right] \tag{1.10}$$

La politique optimale π^* est toujours la politique qui maximise $V^\pi(s)$ pour tout état s. On peut généraliser la formule définissant la fonction Q de la manière suivante:

$$\begin{aligned} Q(s,a) &= E\left[r(s,a) + \gamma V^*(\delta(s,a))\right] \\ &= E\left[r(s,a)\right] + \gamma \sum_{s'} P(s'|s,a) V^*(s') \end{aligned} \tag{1.11}$$

$P(s'|s,a)$ représente la probabilité que l'action a effectuée dans l'état s produise l'état s'. On peut à nouveau exprimer la valeur de la fonction Q récursivement. La fonction $Q(s,a)$ dans le cas non-déterministe est donc la moyenne des différentes valeurs obtenues dans le cas déterministe.

$$Q(s,a) = E\left[r(s,a)\right] + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a') \tag{1.12}$$

Il s'agit maintenant déterminer une règle de mise à jour de la table $\hat{Q}(s,a)$, car celle définie par l'équation 1.8 ne converge plus. On suppose en effet que l'agent reçoit des récompenses différentes lorsqu'il effectue plusieurs fois l'action a dans l'état s et on constate alors que la règle 1.8 va modifier les valeurs des entrées de la table $\hat{Q}$ même si celles-ci ont été initialisées aux valeurs correctes de Q. Pour remédier à cela, il faut modifier la règle de mise à jour de façon à ce que la nouvelle valeur soit une pondération de l'estimation $\hat{Q}$ courante et de la valeur du cas déterministe. Ainsi, si $\hat{Q}_n$ réprésente l'estimation de Q à l'itération n, la formule suivante permet aux estimations successives de converger vers la fonction Q :

$$\hat{Q}_n(s,a) \leftarrow (1-\alpha_n)\hat{Q}_{n-1}(s,a) + \alpha_n \left[r(s,a) + \gamma \max_{a'} \hat{Q}_{n-1}(s',a')\right] \tag{1.13}$$

avec

$$\alpha_n = \frac{1}{1 + visits_n(s,a)} \tag{1.14}$$

où $visits_n(s,a)$ est le nombre de fois que la paire état-action (s,a) a été visitée par l'agent, en incluant l'itération n. Le facteur α_n décroît au fil des itérations, et le terme de l'estimation précédente $\hat{Q}_{n-1}(s,a)$ prend davantage d'importance par rapport à l'estimation déterministe. La variation des mises à jour s'atténue donc jusqu'à ce qu'elle devienne négligeable. Il existe en outre d'autres formules faisant décroître α_n pour lesquelles la convergence est garantie. Il faut pour cela qu'elles vérifient, pour $0 \leq \alpha_n < 1$, que

$$\sum_{i=1}^{\infty} \alpha_{n(i,s,a)} = \infty \quad \text{et} \quad \sum_{\mathrm{i=1}}^{\infty} \left[\alpha_{\mathrm{n(i,s,a)}}\right]^2 < \infty \tag{1.15}$$

Dans ces formules $n(i,s,a)$ est l'itération correspondant à la $\mathrm{i}^{\text{ème}}$ fois que l'action a est effectuée dans l'état s.

Le chapitre suivant présente le problème de coordination sur lequel le Q learning sera testé. Il comporte également une description des différentes modélisations effectuées ainsi qu'une analyse des résultats expérimentaux.

Chapitre 2

Mise en oeuvre des techniques existantes

Le problème de coordination qui va être résolu est décrit dans ce chapitre, suivi des choix de modélisation effectués. Une première modélisation du comportement des agents consiste à leur assigner une fonction d'utilité Q individuelle à estimer. De tels agents sont appelés *indépendants*. La seconde se base sur une fonction Q globale déterminant l'utilité des états et actions conjointes. Ces agents sont dits *collaboratifs* [3]. Les résultats expérimentaux de ces deux méthodes sont repris au sein de la section 2.3.

2.1 Présentation du problème

Afin d'être confronté à des objectifs de complexité graduelle, le scénario initial présenté à la section 2.1.1 sera réduit à une version dans laquelle l'environnement est modélisé par une grille bidimensionnelle. Cette modélisation décrite à la section 2.1.2 constitue la spécification de base du mémoire. Cette section s'achèvera par une généralisation du modèle de l'environnement à un graphe.

2.1.1 Scénario

Le scénario initial est caractérisé par les propriétés suivantes:

Environnement : le monde réel à trois dimensions ;

Agents : avions ayant chacun un rôle spécifique, à savoir *scanner* et *canadair*. Ces agents peuvent communiquer entre eux et doivent se coordonner pour accomplir une tâche;

Actions : outre la capacité commune de voler, chaque agent dispose d'actions spécifiques à son rôle. Le scanner possède un système de vision lui permettant d'analyser une zone incendiée et de choisir une cible qu'il communiquera aux autres avions. Les canadairs ont des dispositifs permettant de pomper de l'eau d'un lac pour ensuite la larguer sur la cible spécifiée par le scanner ;

Objectifs des agents : se coordonner pour respecter un plan de vol et éteindre un feu de forêt. Les avions doivent donc suivre une trajectoire qui est confinée à l'intérieur d'un couloir de vol, choisir une configuration de formation efficace et, arrivés à destination, larguer de l'eau sur l'incendie.

La figure 2.1 illustre un tel scénario.

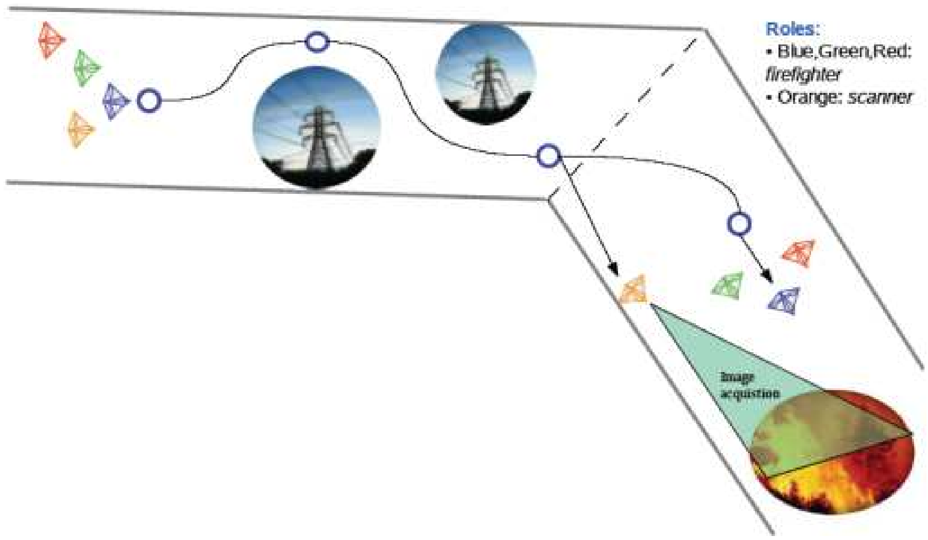


Figure 2.1: Illustration du problème général.

2.1.2 Hypothèses de modélisation

Le scénario ci-dessus peut dans un premier temps être modélisé en utilisant les hypothèses suivantes :

Environnement : l'environnement a été discrétisé à un espace à deux dimensions représenté par une grille de connexité quatre. Chaque case de la grille correspond à une position (x, y) pouvant être libre ou occupée par des agents et/ou des obstacles ;

Agents : les agents peuvent évoluer librement dans les limites de l'environnement. Ces agents partent d'une position initiale et doivent atteindre une position finale ;

Actions : pour se déplacer, les agents disposent de quatre actions : Nord, Sud, Est et Ouest. Ces actions produisent un déplacement de l'agent, à l'intérieur des limites de l'environnement, d'une case dans la direction correspondante ;

Objectif : la mission des agents consiste à coordonner leurs déplacements, de façon à éviter les collisions tout en empruntant le chemin le plus court de leur point de départ à leur goal.

La figure 2.2 illustre le problème dans le cadre d'un environnement de taille 10×10 et de quatre agents [12]. Les agents débutent aux positions A, B, C et D, et doivent arriver respectivement en A', B', C' et D'. Les parcours en pointillés correspondent à l'une des solutions du problème.

2.1.3 Extension de la modélisation

La modélisation présentée à la section 2.1.2 possède cependant quelques limitations :

Environnement : la représentation choisie pour modéliser l'environnement est limitative dans le sens où il est difficile de simuler un environnement de topologie complexe contenant, par exemple, des zones inaccessibles.

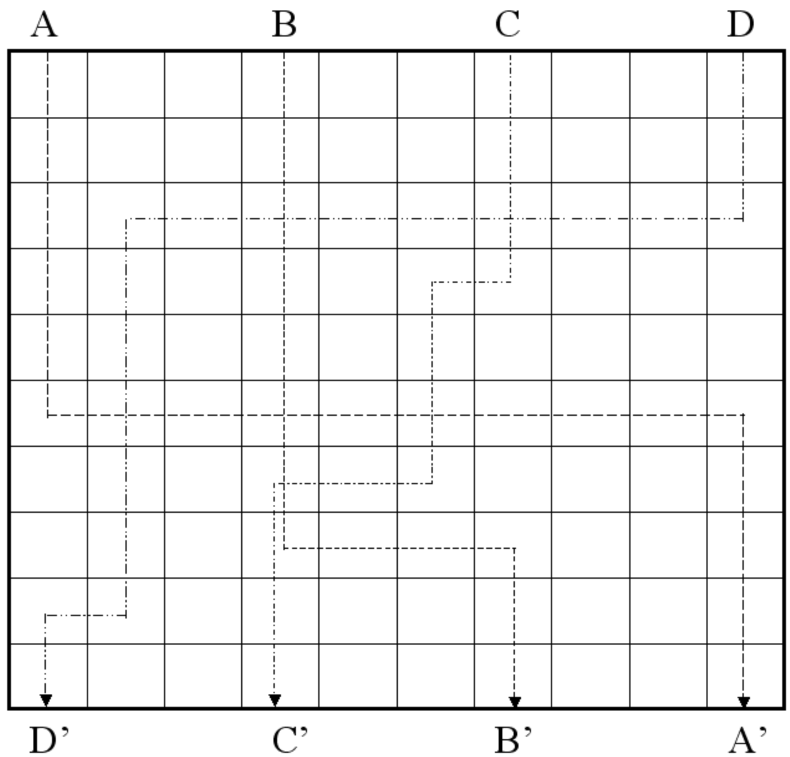


Figure 2.2: Description du problème dans le cas de quatre agents évoluant dans une grille de taille 10×10 [12].

Actions : jusqu'à présent les actions sont limitées à Nord, Sud, Est, Ouest et sont applicables à partir de n'importe quelle position dans l'environnement. Cependant, dans un monde complexe il existe certains états où toutes les actions ne sont pas applicables. Il est possible d'établir les contraintes sur les actions à partir de règles explicites, mais il est beaucoup plus simple de trouver une représentation de l'environnement qui tienne compte implicitement de ces règles.

La solution à ces limitations est d'utiliser un graphe pour modéliser le monde externe. Ainsi, les noeuds du graphe représentent les états possibles et les arêtes les actions permises à partir du noeud correspondant. On peut ainsi modéliser des environnements plus complexes, tout en limitant implicitement le choix des actions pour chaque état. Il n'est en effet pas nécessaire de rajouter des règles définissant les actions possibles en fonction de l'état, ces actions étant représentées par les arêtes du graphe. La figure 2.3 illustre un environnement dont certaines positions ont un nombre limité d'actions.

Dans le cas d'avions devant se coordonner, on peut par exemple se limiter à un graphe orienté unidirectionnel car les avions ne peuvent pas faire de demi-tour. On diminue ainsi considérablement la taille de l'espace état-action. La figure 2.4 illustre un exemple de graphe pour ce cas particulier.

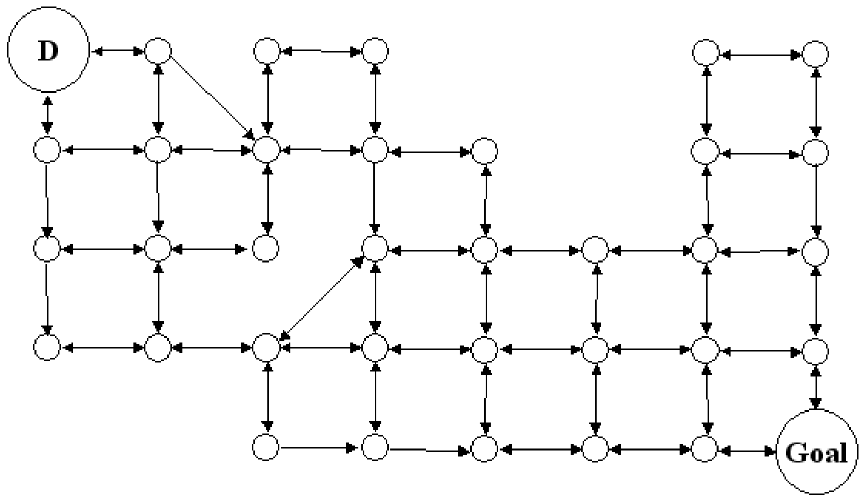


Figure 2.3: Représentation d'un environnement complexe en utilisant un graphe. Tous les états (noeuds) ne permettent pas les mêmes actions (arêtes sortantes).

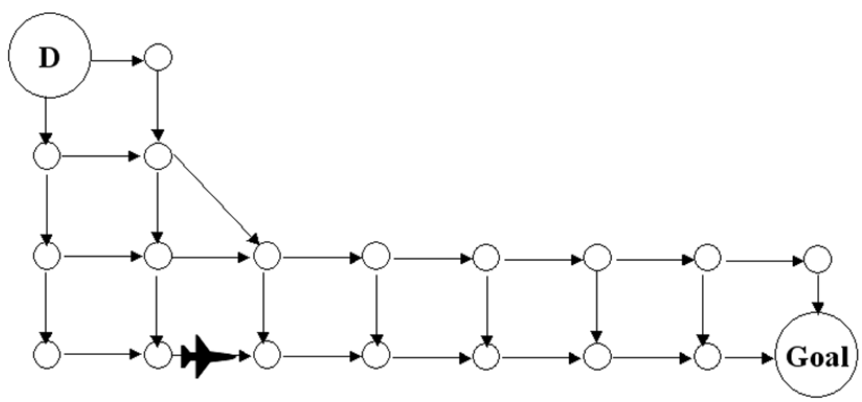


Figure 2.4: Graphe orienté unidirectionnel limitant les actions possibles dans chaque état.

2.1.4 Motivations

Le but de ce travail est d'implémenter un algorithme d'apprentissage de la coordination d'agents pour les environnements définis aux sections 2.1.2 et 2.1.3. Les algorithmes classiques d'intelligence artificielle permettent de trouver un chemin optimal, mais ils nécessitent la connaissance au préalable de l'environnement et ne fournissent une solution que pour un état initial particulier.

L'algorithme permettra à des agents d'apprendre à se coordonner quelles que soient leurs positions respectives. Ceci est important car les agents doivent pouvoir réagir correctement dans des situations imprévues. Il est par exemple important de pouvoir faire face à l'incertitude dans le cas d'avions déviant de leur position prévue à cause de perturbations climatiques.

Il est en outre important de réaliser une structure permettant de tester aisément différents apprentissages. Le programme implémenté permettra de spécifier la taille de la grille, le nombre d'itérations, la présence d'obstacles, la configuration des agents et enfin les paramètres spécifiques au Q learning. Ainsi, l'utilisateur pourra faire apprendre aux agents des différents scénarios. Ce programme fournira en temps réel des statistiques sur l'apprentissage, de façon à ce que l'utilisateur puisse évaluer la qualité de la politique apprise. Enfin, l'interface graphique inclura une représentation des mouvements des robots décrivant la politique obtenue.

En ce qui concerne les applications pratiques, l'algorithme de ce travail pourrait par exemple être utilisé par un ensemble de robots souhaitant apprendre en temps réel les mouvements optimaux maximisant leurs récompenses. Des robots autonomes peuvent s'attribuer ces récompenses à eux-mêmes lorsqu'ils détectent que leur objectif est atteint. Une autre application serait l'affichage dans un avion des déplacements optimaux à effectuer en fonction de la position des avions environnants.

2.2 Modélisation du problème

Afin de résoudre le problème énoncé à la section précédente, il est nécessaire de modéliser différents concepts. Une première modélisation de l'environnement a été effectuée à la section 2.1.2. Cette section décrit les choix de modélisation concernant trois aspects :

- Un agent pouvant se déplacer ;
- Un simulateur contenant les informations de l'environnement nécessaires à l'apprentissage ;
- La communication entre un agent et le simulateur.

2.2.1 Agent unique

L'environnement comporte un agent unique ainsi que des obstacles éventuels. Le but visé est de faire apprendre à l'agent un chemin *optimal* de son point de départ à son goal. On identifie deux critères d'optimalité:

- Une longueur de chemin minimale. A cette fin, l'agent reçoit une récompense positive lorsqu'il atteint son goal. L'algorithme va ainsi converger vers la politique permettant à l'agent de recevoir cette récompense le plus rapidement, ce qui correspond au chemin le plus court ;
- Aucune collision avec les obstacles éventuels. Ce critère est atteint en assignant des récompenses négatives lors de chaque collision avec un obstacle. Ainsi, les entrées $\hat{Q}(s,a)$ telles que l'état résultant $s' = \delta(s,a)$ comporte un obstacle auront une Q-valeur inférieure aux autres.

Un agent est une entité pouvant se déplacer dans un environnement. Il doit pour cela choisir une action à effectuer en tenant compte de son état courant. Nommons S l'ensemble des états possibles. Dans le cas d'un environnement de dimension $D \times D$ comme décrit dans la section 2.1, l'état d'un agent peut, dans un premier temps, être défini par sa position dans la grille. Si N est l'ensemble des entiers naturels, on a donc

$$S = \{(x,y) | \, (x,y) \in N^2 \text{ et } 1 \leq x, y \leq D\} \tag{2.1}$$

L'ensemble A des actions que l'agent peut effecuer se limite aux quatre directions cardinales :

$$A = \{\text{Nord, Sud, Est, Ouest}\} \tag{2.2}$$

On modélise alors l'agent par une fonction $Agent : S \to A$. Lors de l'apprentissage, cette fonction dépend de la politique d'exploration employée (voir section 1.3.7). Lors de l'utilisation de la politique de déplacement apprise, l'agent est responsable du choix de son action en se basant sur sa table $\hat{Q}$ et les informations fournies par le simulateur. Ce dernier communique à l'agent les actions valides $Actions(s)$ pour un état particulier s, telles que

$$\forall \, s \in S, \, Actions(s) = \{a \in A | \, (x', y') = \delta(s,a) \text{ et } 1 \leq x', y' \leq D\} \tag{2.3}$$

Avant chaque mouvement, l'agent reçoit son état courant s et l'ensemble $Actions(s)$ du simulateur et lui retourne l'action a telle que $\hat{Q}(s,a)$ est maximal. Si l'on applique la politique apprise, la fonction *Agent* se définit comme :

$$\forall\, s \in S,$$
$$\begin{cases} \text{si } \exists\, a \in A \text{ tel que } \hat{Q}(s,a) \text{ est défini: } Agent(s) = \text{argmax}_a\, \hat{Q}(s,a) \\ \text{sinon: } Agent(s) \text{ est un élément aléatoire de } Actions(s) \end{cases}$$

Le choix de la position dans la grille comme définition de l'état de l'agent engendre toutefois une limitation : la politique obtenue ne sera optimale que dans l'environnement d'apprentissage. L'agent va en effet se baser sur les positions des obstacles et de son goal pour trouver le chemin optimal. Si l'on change la position des obstacles, l'exécution de la politique apprise reproduira toujours le même chemin. Il est dès lors nécessaire de changer la définition de l'état de l'agent pour pouvoir trouver une politique applicable dans des environnements différents. On peut par exemple définir l'état d'un agent comme sa position et le contenu des cases adjacentes dans l'environnement. Il faut pour cela émettre l'hypothèse que l'agent possède des capteurs lui permettant de déterminer si un obstacle est présent dans un rayon d'une case autour de lui. Définissons $Env(s)$ comme un quadruplet représentant le contenu des cases adjacentes à s dans les directions Nord, Sud, Est et Ouest (par exemple $Env((1,3)) = (\text{occupée, libre, libre, libre})$). Le nouvel ensemble d'états S_{env} est défini comme :

$$S_{env} = \{(s, Env(s)) |\, s \in S\} \tag{2.4}$$

Grâce à cette définition de l'état, un agent est capable de prendre des décisions en fonction de ce qui se trouve à proximité de lui. Il peut ainsi apprendre à contourner une case occupée par un obstacle dynamique ou un autre agent, si toutefois l'apprentissage s'est effectué avec des obstacles différents d'une itération à une autre. Ce problème sera abordé dans la section 2.3.4.

2.2.2 Simulateur

Le but du simulateur est de fournir les informations sur l'environnement nécessaires à l'apprentissage de l'agent. Son objectif n'est donc pas de reproduire fidèlement un environnement, mais de simuler une interaction basique entre l'agent et le monde externe. Il est responsable des services suivants :

- Stocker l'état de l'agent et un modèle de l'environnement ;
- Communiquer à l'agent son état lorsqu'il le demande. Il fait ainsi office de capteur dans le cas d'un état contenant l'environnement proche de l'agent. Il transmet également les actions valides pour un état donné ;
- Exécuter l'action choisie par l'agent. Ce point est important car il permet d'introduire le caractère non-déterministe d'un environnement réel. En effet, même si l'agent choisit l'action qui lui semble la plus appropriée dans un état particulier, il se peut que l'environnement soit perturbé et que le résultat de l'action ne soit pas celui escompté. On peut par exemple imaginer le cas de robots dont les mouvements seraient perturbés par un vent important ;

- Récompenser l'agent positivement lorsqu'il atteint son goal et négativement en cas de collision.

Le simulateur peut être modélisé par la fonction

$$Simulateur : Env \times Message \rightarrow Réponse$$

Env désigne ici l'ensemble des environnements globaux possibles et $Message$ contient les messages que peut envoyer l'agent. Il existe deux types de messages possibles: une requête d'état et l'action choisie par l'agent. Selon le type de message, la sortie *Réponse* du simulateur sera respectivement l'état courant de l'agent ou la valeur de la récompense correspondant à l'action effectuée.

Le simulateur possède les caractéristiques suivantes :

- Les agents constituant des entités indépendantes, le simulateur peut recevoir des messages des différents agents simultanément. Cette caractéristique dépend toutefois de l'implémentation sous-jacente à la communication entre les agents et le simulateur ;
- La séquence de messages reçue n'est pas ordonnée. Le simulateur ne connaît pas d'avance l'agent qui enverra le message suivant ;
- Il traite les messages des agents de manière asynchrone. Cela signifie que le simulateur ne doit pas répondre instantanément aux messages qu'il reçoit. Il peut donc en traiter plusieurs à la fois ;
- La synchronisation entre plusieurs agents est assurée de la manière suivante : le simulateur définit de courts intervalles de temps de façon à considérer les messages reçus dans une même période comme simultanés.

En ce qui concerne cette dernière caractéristique, il est intéressant de noter l'importance de l'intervalle de temps déterminant la simultanéité des messages. Si celui-ci est trop court, tous les messages reçus sont considérés comme des évènements arrivant à des temps différents. La coordination est alors inefficace car la modélisation n'est plus assez réaliste. Les agents pourront en effet obtenir une information de l'environnement toujours cohérente et n'entreront jamais en collision. Si par contre l'intervalle de temps est trop long, les agents attendront longtemps la réponse à leur message ce qui peut fortement ralentir la simulation. On note en outre qu'un agent ne peut envoyer deux messages dans un même intervalle de temps car il doit obtenir la réponse à son premier message avant de pouvoir envoyer le second. La figure 2.5 illustre la synchronisation des messages envoyés par deux agents.

2.2.3 Communication entre agent et simulateur

Cette section résume l'ensemble des interactions entre un agent et le simulateur. L'agent i commence par envoyer un message $sense_{i,t_0}$ au temps t_0. Le simulateur lui renvoie son état s_{i,t_1} correspondant au temps t_1 où le message est reçu. L'agent choisit ensuite une action en se basant sur son état s_{i,t_1}, et envoie au temps t_2 le message $action_{i,t_2}$ correspondant. Enfin, si l'apprentissage est en cours, le simulateur communique à l'agent la valeur de la récompense r relative à l'action effectuée. Ces quatres étapes sont répétées jusqu'à ce que

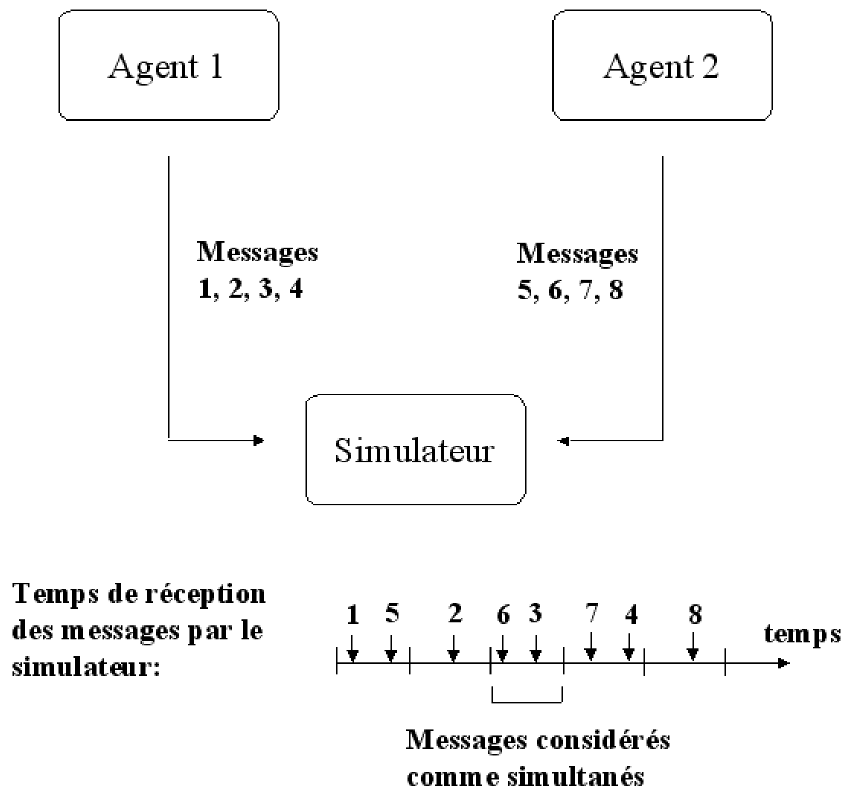


Figure 2.5: Synchronisation des messages reçus au court d'un même intervalle de temps, dans le cas de deux agents envoyant chacun 4 messages à des instants différents. Le message 2 est considéré comme étant isolé alors que les messages 6 et 3 sont simultanés.

l'agent trouve son goal. Le diagramme de messages représenté à la figure 2.6 résume les interactions entre les deux modules.

2.2.4 Coordination de plusieurs agents

Cette section est consacrée à la modélisation de la coordination de plusieurs agents se déplaçant simultanément, il s'agit du problème central du mémoire. Chaque agent possède un point de départ et un goal différent. Le but est ici d'apprendre un ensemble de chemins optimaux, mais il est nécessaire de redéfinir le critère d'optimalité :

- La somme des longueurs des chemins doit être minimale. A cette fin, une récompense est donnée lorsqu'un agent trouve son goal. On peut également envisager le cas où l'on ne minimise que la longueur du chemin le plus long. Ceci est obtenu en ne donnant une récompense que lorsque le dernier agent atteint son objectif ;
- Aucune collision n'est permise avec les obstacles éventuels. De la même manière que dans le cas de l'agent unique, une punition sera infligée en cas de collision ;
- Aucune collision n'est admise entre les agents. Une telle collision apparaît lorsque plusieurs agents partagent la même position dans l'environnement. Ceci peut être résolu en assignant une récompense négative aux agents lors d'une collision.

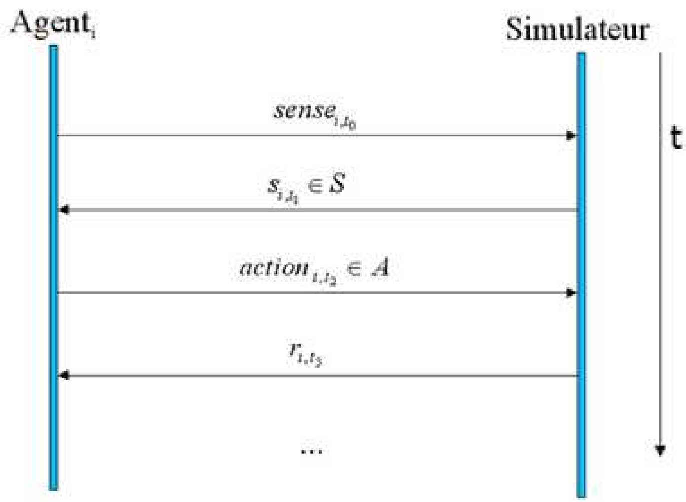


Figure 2.6: Diagramme de messages représentant la communication entre agent et simulateur.

Deux méthodes de coordination entre les agents vont maintenant êtres décrites. Dans la première, les agents sont dits *indépendants* car ils essaient chacun de déterminer leur propre fonction d'utilité Q. La seconde approche utilise une seule fonction Q globale *jointe* impliquant le partage des récompenses et des pénalités entre tous les agents.

2.2.5 Agents indépendants

Dans ce cas de figure, les agents cherchent à estimer leur propre fonction Q. Ils stockent donc chacun une table $\hat{Q}$ caractérisant leur politique apprise. L'ensemble des actions A est identique à celui défini par l'équation 2.2 dans le cas d'un agent seul. Chaque agent va apprendre un chemin optimal vers son goal. Si l'algorithme déterministe est utilisé, un risque de collision existe lors de la superposition des divers chemins optimaux appris. Il est donc nécessaire d'utiliser l'ensemble d'états S_{env} défini à l'équation 2.4 permettant aux agents de détecter la proximité d'autres objets. L'état des agents est alors constitué de leur position et de ce qui se trouve dans les positions adjacentes à celle-ci. Dans le cas d'un robot, cela revient à ce qu'il se déplace dans une grille en se basant sur sa position et sur ses capteurs dont la portée est d'une case. Ils apprendront ainsi à ne pas se déplacer vers une case adjacente occupée. La fonction Q_i que l'agent i cherche à estimer est définie comme :

$$Q_i : S_{env} \times A \rightarrow QVal$$

L'ensemble des valeurs possibles $QVal$ de la table $\hat{Q}$ peut être assimilé à l'ensemble des réels R.

Cette modélisation pose toutefois un problème de coordination au niveau des collisions entre agents. Les agents reçoivent en effet les informations de leurs capteurs à un temps t et l'action correspondante est exécutée à un temps t' tel que $t' > t$. Deux agents distants de deux unités de position détectent à partir de leurs capteurs que leur environnement

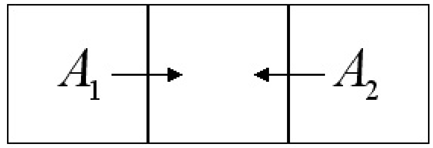


Figure 2.7: Collision entre deux agents A_1 et A_2 ne pouvant pas être apprise dans le cas de fonctions Q indépendantes et d'une politique d'exploration aléatoire.

immédiat est inoccupé. Il se peut donc qu'ils choisissent d'aller à la même position vacante et qu'ils entrent en collision. Ce type de collision est illustré par la figure 2.7.

Dans le cas de l'algorithme déterministe basé sur des mouvements aléatoires (section 1.3.7), des agents indépendants munis de capteurs ne peuvent apprendre à éviter ce type de collisions. Du point de vue d'un agent isolé, ces collisions entraînent un non-déterminisme. En effet, pour une même action a et un même état s composé de la position et de l'occupation des cases adjacentes, un agent recevra une récompense nulle si tout se passe bien et une récompense négative s'il entre en collision comme dans la figure 2.7. L'algorithme déterministe prend uniquement en compte les dernières récompenses reçues pour chaque action effectuée. L'effet d'une collision non-déterministe d'une action a dans un état s sera donc nul si au cours d'un épisode suivant l'action a dans s ne produit pas de collision, l'ancienne valeur de la table $\hat{Q}$ étant dès lors écrasée par la nouvelle.

Toujours dans le cas d'une politique de déplacement aléatoire, ce problème est partiellement résolu par l'algorithme non-déterministe caractérisé par la règle de mise à jour 1.13. Cet algorithme tient en effet compte de toutes les récompenses reçues au cours du temps pour chaque paire état-action. Les actions qui ont une plus grande probabilité de mener à une collision seront donc pénalisées par rapport aux autres. Cependant, pour des positions éloignées des points de départ des agents ces probabilités de collision seront faibles et identiques. Les agents ne sauront donc pas détecter quelle action aura engendré davantage de collisions qu'une autre et ils n'auront pas appris à éviter les collisions illustrées à la figure 2.7.

Dans le cas de l'algorithme non-déterministe combiné avec la politique d'exploration probabiliste décrite à la section 1.3.7, l'apprentissage des agents est effectué en favorisant les actions dont la valeur de l'estimation $\hat{Q}$ est élevée. Les épisodes d'apprentissage successifs vont par conséquent converger vers des parcours similaires. Si dans un premier temps ce parcours génère des collisions de type non-déterministe, les actions responsables de celles-ci seront fortement pénalisées car ces mêmes collisions seront répétées d'un épisode à un autre. Les agents finiront par éviter ces collisions et converger vers un chemin optimal.

Un dernier inconvénient des agents indépendants est qu'ils n'apprennent à s'éviter que lorsqu'ils sont à proximité l'un de l'autre. Ceci est dû au fait que les seules informations dont ils disposent sur les autres agents sont fournies par leurs capteurs. Ils sont donc incapables de trouver un chemin qui les maintienne à une distance suffisante pour minimiser les risques de collision.

Les résultats de la mise en oeuvre de cette modélisation feront l'objet d'une analyse et d'une vérification au sein de la section 2.3.2.

2.2.6 Agents collaboratifs

Cette approche consiste à l'estimation, par les agents, d'une même fonction Q jointe [3]. Cette fonction prend toujours en entrée un état s et une action a, mais ici s désigne l'état de tous les agents et a les actions effectuées simultanément. Dans le cas de I agents se déplaçant en même temps, les ensembles des états S_{joint} et des actions A_{joint} sont respectivement :

$$S_{joint} = S^I \quad \text{et} \quad A_{joint} = A^I$$

S et A sont les ensembles d'états et d'actions dans le cas d'un agent seul décrits dans la section 2.2.1. La fonction Q jointe estimée est désormais définie comme suit :

$$Q : S_{joint} \times A_{joint} \rightarrow QVal$$

En apprenant cette fonction jointe, un agent va partager avec les autres les récompenses qu'il a obtenues. Les agents vont ainsi apprendre à se coordonner en effectuant les actions qui maximisent la fonction d'utilité globale. Cette approche permet d'éviter les collisions illustrées par la figure 2.7, car ces dernières sont maintenant déterministes. Lorsque ce type de collision survient, l'entrée de la table $\hat{Q}$ correspondant aux actions jointes effectuées se verra infligée une pénalité par rapport aux autres entrées. Contrairement au cas d'agents indépendants, l'objectif visé par l'apprentissage est de maximiser les récompenses communes et non individuelles.

Lorsque les agents ont appris la politique jointe, ils doivent se baser sur la même table pour se déplacer. Ils commencent par demander chacun l'état global $s \in S_{joint}$ au simulateur. Ils peuvent alors déterminer à quelle action jointe $a \in A_{joint}$ correspond l'entrée (s, a) possédant la plus grande valeur dans la table $\hat{Q}$. Lorsqu'un agent a déterminé cet ensemble d'actions a, il lui reste à choisir l'action qui lui correspond parmi celles de a et à la communiquer au simulateur.

L'inconvénient principal de cette méthode par rapport à celle de la section précédente réside dans le nombre important de paires état-action à visiter afin de faire converger l'algorithme vers la politique optimale. Examinons le cas de N_{agents} agents pouvant effectuer $N_{actions}$ actions dans une grille de dimension $D \times D$. Si, de plus, les états $s \in S_{joint}$ ne contiennent que les positions respectives des agents, il existe alors pour chaque agent $N_{actions} \cdot D^2$ paires état-action possibles. Dans le cas de N_{agents} agents, le nombre total de paires état-action est :

$$N_{état-action} = (N_{actions} \cdot D^2)^{N_{agents}} \tag{2.5}$$

Si l'on considère 4 agents pouvant effectuer 4 actions dans un environnement de taille 10×10, on obtient 25,6 milliards de paires état-action. Le théorème mentionné à la section 1.3.4 stipule que la convergence est garantie si chacune de ces paires est visitée infiniment souvent. Dans cet exemple pourtant simple on se rend compte qu'une unique visite de tous les états et actions nécessite déjà un temps très long. De plus, ce nombre de paires état-action correspond également à la taille maximale de la table $\hat{Q}$ vers laquelle l'algorithme converge. La quantité de mémoire requise pour stocker cette table peut dès lors être très grande. La figure 2.8 illustre la taille de l'espace état-action en fonction du nombre d'agents pour différentes tailles d'environnement.

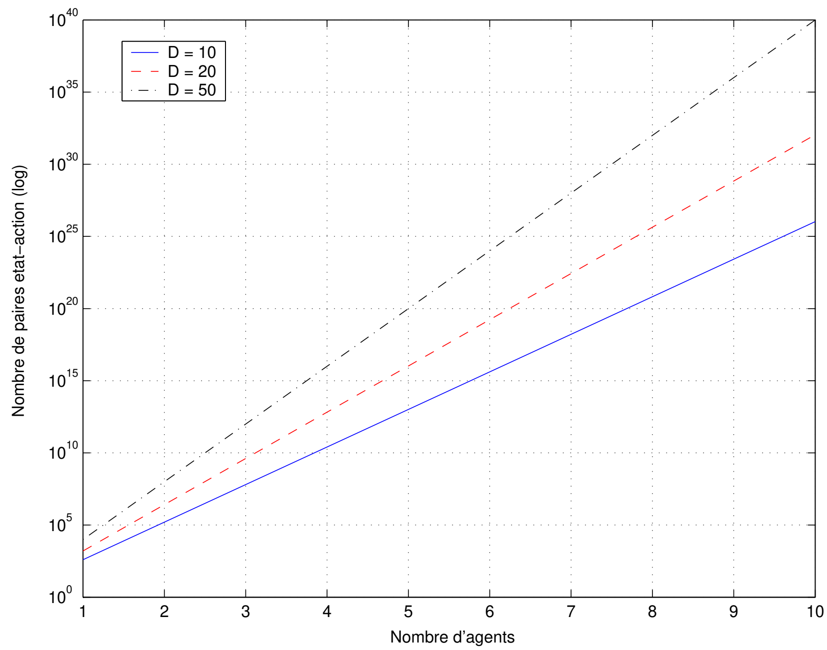


Figure 2.8: Taille de l'espace état-action en fonction du nombre d'agents dans le cas d'une table jointe (échelle logarithmique). L'environnement est une grille de taille $D \times D$. Les agents peuvent effectuer 4 actions possibles, et leur état est uniquement représenté par leur position.

On note que l'utilisation d'une fonction Q jointe implique que chaque agent connaisse l'état de tous les autres au moment du choix de l'action jointe de valeur maximale dans la table $\hat{Q}$. Dans un environnement réel, cela revient à émettre l'hypothèse que les agents peuvent communiquer entre eux à tout moment pour connaître l'état de chacun.

Le chapitre 3 propose une solution se basant sur une table $\hat{Q}$ apprise avec deux agents et applicable à un plus grand nombre d'agents. Cette approche permet de limiter le nombre de paires état-action à visiter si $N_{agents} > 2$.

2.3 Résultats expérimentaux

Les résultats obtenus lors de l'exécution de différents scénarios sont présentés et analysés dans cette section. Ces scenarios mettent en oeuvre les modélisations vues dans la section 2.2.

Tout au long de cette section, le terme *épisode* ou *itération* représentera une succession de déplacements d'agents allant de leur point de départ à leur goal, suivie de l'application du Q learning à ce parcours. Le terme *apprentissage* désignera quant à lui une séquence d'épisodes.

2.3.1 Agent unique

Ce premier test porte sur l'apprentissage par un seul agent d'une politique optimale de déplacement dans un environnement statique. Cet environnement comporte des zones d'obstacles, donnant lieu à des récompenses négatives si l'agent s'y aventure. Les résultats obtenus permettent de se faire une idée des performances de l'implémentation réalisée dans ce travail.

Scénario et résultats

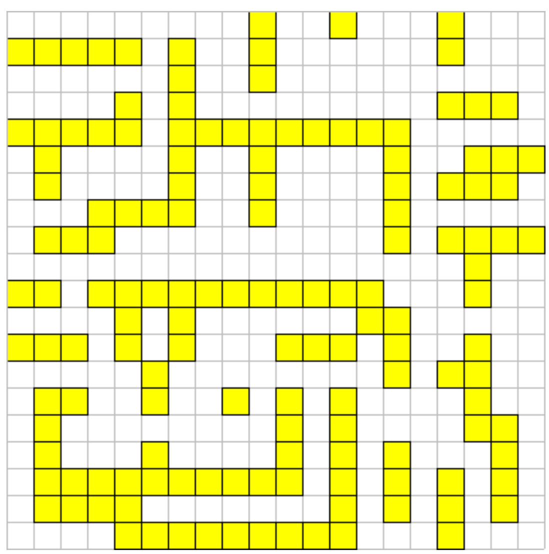


Figure 2.9: Environnement de test pour un agent. Les obstacles sont les cases jaunes.

Les paramètres utilisés sont :

Environnement : une grille bidimensionnelle de taille 20×20.

Agent : un seul agent dans l'environnement. Il peut effectuer les actions Nord, Sud, Est, Ouest et son état est caractérisé par le couple $< Position, Goal >$. Chaque apprentissage débute à la position $(1,1)$ et se termine dès que le robot a atteint la position $(20,20)$.

Fonction d'assignation des récompenses : l'agent reçoit une récompense positive de 20.0 quand il atteint son goal et une punition de -10.0 lorsqu'il entre en collision avec un obstacle.

Fonction d'exploration : l'agent explore dans un premier temps l'environnement de manière aléatoire. Par la suite, l'exploration probabiliste décrite à la section 1.3.7 est testée, pour une valeur initiale de la température fixée à 100.0. Cette valeur est décrémentée au fil des épisodes. Ceci donne lieu à une exploration importante en début d'apprentissage et une forte utilisation des connaissances vers la fin.

Objectif : apprendre une politique optimale fournissant un système de choix des actions à effectuer afin de minimiser la longueur du chemin parcouru jusqu'au goal tout en évitant les obstacles.

La figure 2.9 illustre ce scénario.

Résultats expérimentaux

Les résultats obtenus correspondent à des moyennes calculées à l'aide des résultats de 20 apprentissages d'une durée de 200 épisodes chacun.

Exploration aléatoire La figure 2.10 représente les moyennes des longueurs des chemins appris en fonction du nombre d'épisodes d'apprentissage. La figure 2.11 illustre le nombre de collisions moyen par rapport à ces mêmes épisodes. On y constate que les courbes convergent vers leurs valeurs optimales respectives, c'est-à-dire vers une valeur de 39 en 195 itérations pour la courbe des longueurs des chemins et vers la valeur zéro en 81 itérations pour le nombre de collisions *agent-obstacle*. L'agent a donc appris une politique optimale en 195 épisodes.

Exploration probabiliste Les résultats obtenus en utilisant une exploration probabiliste sont comparés avec ceux de l'exploration aléatoire dans les figures 2.12 et 2.13. Ces graphiques montrent les moyennes sur 20 apprentissages d'une durée de 200 épisodes chacun. On y remarque que les nouvelles courbes d'apprentissage de l'agent convergent plus rapidement vers leurs valeurs optimales respectives. La courbe de longueur des chemins atteint sa valeur optimale en 30 épisodes contre 195 pour la politique d'exploration aléatoire. Le nombre de collisions atteint une valeur nulle en 8 épisodes dans le cas probabiliste alors qu'il lui en fallait 81 dans le cas de mouvements aléatoires. Ceci est la conséquence de l'utilisation des connaissances acquises lors de l'exploration de l'environnement.

Conclusion

L'algorithme d'apprentissage d'un agent dans un environnement statique fournit de bons résultats si l'environnement est exploré aléatoirement. Une exploration probabiliste, se basant sur les connaissances acquises, améliore substantiellement l'apprentissage.

La section suivante analyse les perfomances de l'algorithme dans le cas où plusieurs agents du même type que celui étudié ci-dessus sont placés ensemble dans ce même environnement.

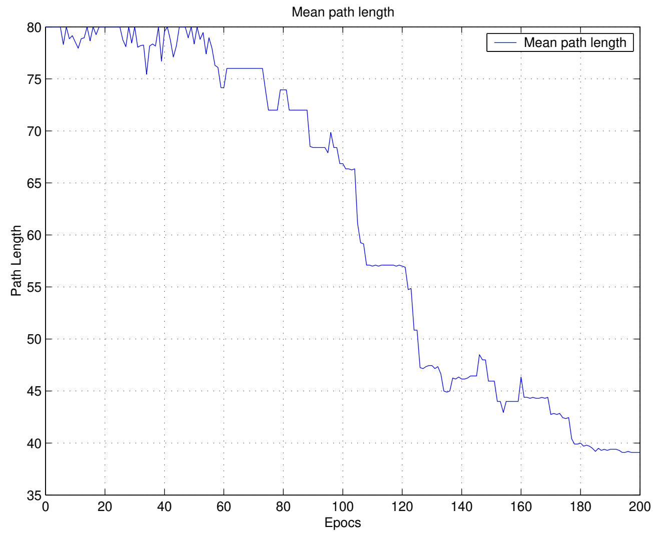


Figure 2.10: Moyennes sur 20 apprentissages des longueurs des chemins obtenus après chaque épisode par un agent unique. Ce dernier apprend par exploration aléatoire d'un environnement comportant des obstacles. On observe une convergence rapide vers la valeur optimale.

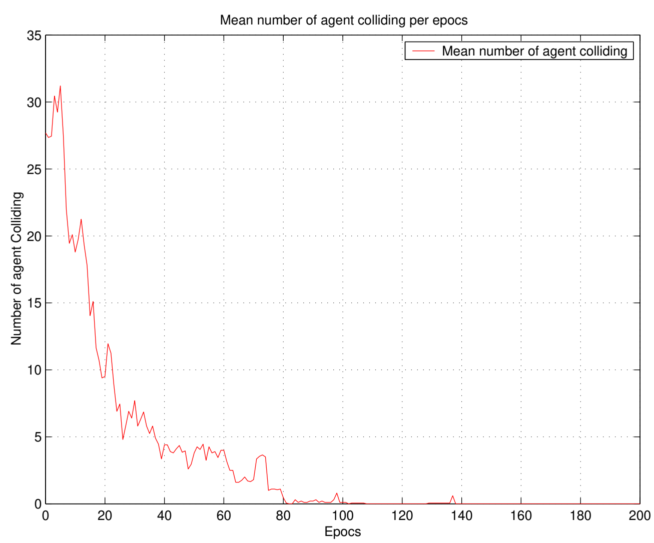


Figure 2.11: Moyennes sur 20 apprentissages des collisions occasionnées par un agent unique qui apprend par exploration aléatoire dans un environnement comportant des obstacles. La valeur optimale de zéro collision est atteinte.

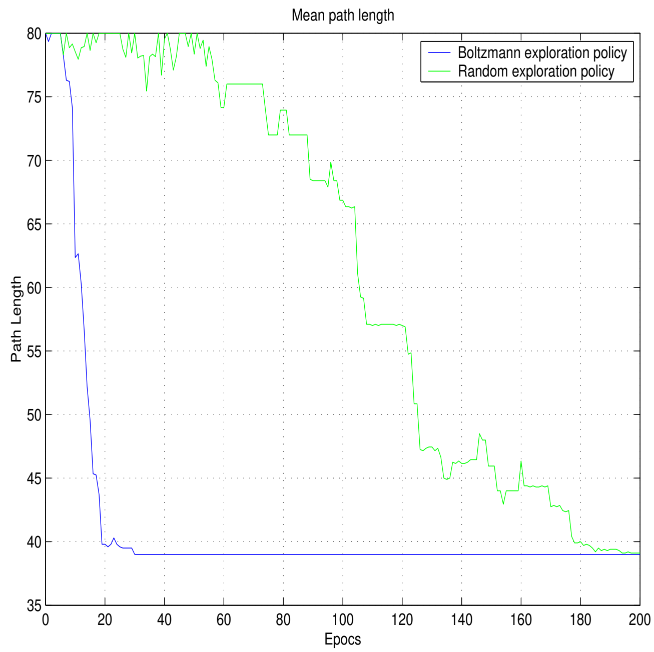


Figure 2.12: Moyennes sur 20 apprentissages des longueurs des chemins obtenus après chaque épisode par un agent unique qui apprend par exploration probabiliste un environnement comportant des obstacles. La convergence vers la valeur optimale 39 est plus rapide que dans le cas de l'exploration aléatoire.

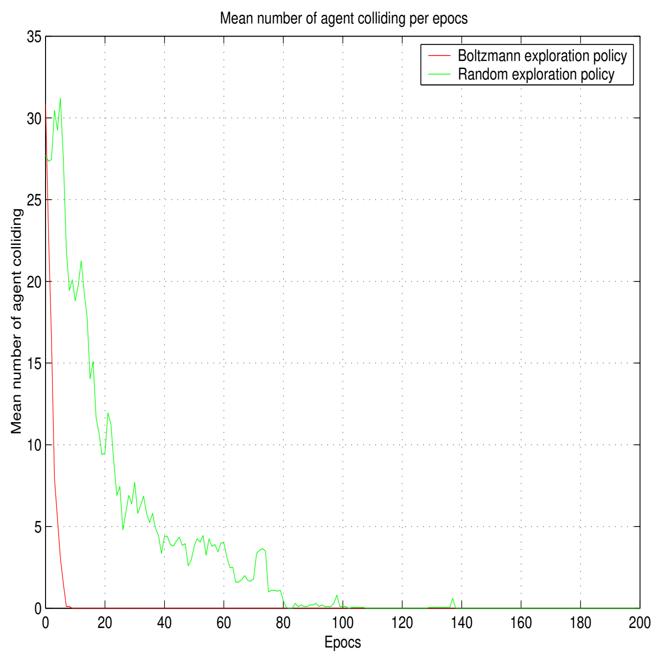


Figure 2.13: Moyennes sur 20 apprentissages du nombre de collisions occasionnées après chaque épisode par un agent unique. Ce dernier apprend par exploration probabiliste dans un environnement comportant des obstacles. La convergence vers zéro est plus rapide que dans le cas d'une exploration aléatoire.

2.3.2 Agents indépendants

Comme expliqué dans la section 2.2.5, la première approche du problème de l'apprentissage de plusieurs agents fut d'introduire des agents indépendants dans le même environnement. Le but était de leur faire apprendre individuellement une politique optimale du point de vue d'un agent isolé. Cette section présente les résultats obtenus lors de la mise en oeuvre de cette méthode.

Scénario $1a$

Le premier scénario comporte deux agents ayant des goals différents dans un environnement sans obstacle.

Environnement : la figure 2.14 illustre cet environnement.

Agents : deux agents évoluent dans la grille. L'état de chaque agent est représenté par le couple $< Position, Goal >$. L'un débute son apprentissage à la position $(1, 1)$ et doit atteindre la position $(5, 5)$. Le deuxième part de $(1, 5)$ et doit arriver en $(5, 1)$.

Fonction d'assignation des récompenses : si des agents entrent en collision, ceux impliqués dans la collision reçoivent une pénalité de -20.0. De même, chaque agent reçoit individuellement une récompense positive de $+20.0$ s'il atteint son goal.

Fonction d'exploration : les agents explorent l'environnement de manière aléatoire.

Le choix des paramètres et leur impact sur la qualité de l'apprentissage sera traité dans la section 2.3.5. La figure 2.14 illustre le scénario $1a$.

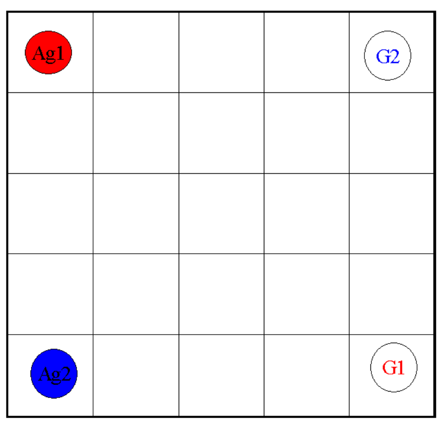


Figure 2.14: Scénario de test pour deux agents. L'agent Ag_1 part de la position $(1, 1)$ et doit atteindre son goal G_1 en $(5, 5)$. L'agent Ag_2 part de $(1, 5)$ et doit atteindre la position $(5, 1)$

Résultats expérimentaux

La figure 2.15 montre la moyenne, par épisode, des longueurs des chemins allant de la position initiale jusqu'à la position finale. La figure 2.16 représente la moyenne, par épisode, du nombre d'agents entrant en collision au fil de l'apprentissage. Ces moyennes ont été calculées sur base de 20 apprentissages, chacun étant composé de 500 épisodes.

Longueur des chemins Après moins de dix épisodes, la longueur moyenne des chemins trouvés descends rapidement à 9. On observe cependant une instabilité de ce chemin. La longueur de ce dernier oscille en effet entre 9 et 11. Cela s'explique par le fait que l'algorithme utilisé est déterministe et il suffit donc d'un épisode comportant beaucoup de collisions pour que les anciennes Q-valeurs soient remplacées par des valeurs moins bonnes (cfr section 2.2.5).

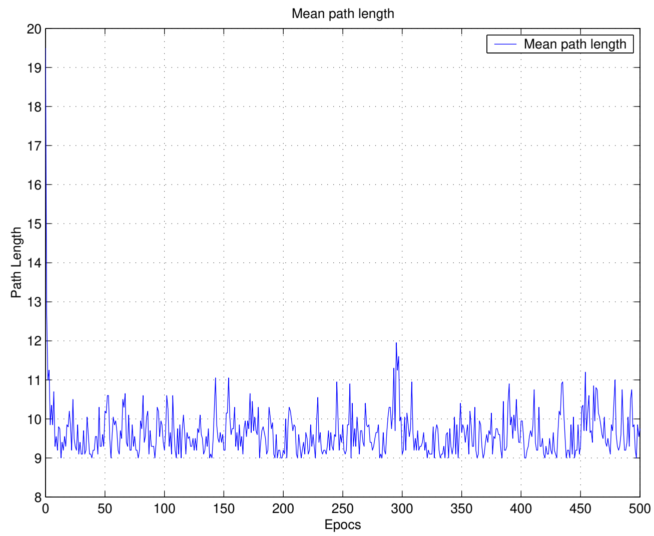


Figure 2.15: Scénario 1a. Moyennes sur base de 20 apprentissages des longueurs des chemins obtenus après chaque épisode par deux agents indépendants. Ces derniers apprennent par application du Q learning déterministe et explorent aléatoirement un environnement sans obstacle. On observe une instabilité du chemin et la convergence n'est pas atteinte.

Nombre de collisions Le nombre moyen d'agents ayant subi une collision chute rapidement après seulement 10 épisodes mais la convergence n'est pas atteinte. Ceci se traduit dans la figure 2.16 par des oscillations des valeurs de la courbe des collisions entre 2 et 3. Cette instabilité s'explique par le caractère non-déterministe des collisions (cfr section 2.2.5). Une amélioration éventuelle de ces résultats nécessite la modification de la configuration du scénario 1a. Les scénarios 1a' et 1a'' investiguent respectivement l'effet de l'utilisation de l'algorithme Q learning non-déterministe et l'effet des capteurs de proximité sur l'apprentissage.

Scénario 1a'

Un ensemble de simulations son exécutées en utilisant la configuration du scénario 1a ci-dessus avec l'algorithme Q learning non-déterministe.

La courbe de la figure 2.17 converge vers la valeur optimale et met en évidence un effet de *lissage* par rapport à celle de la figure 2.15. Comme cela avait été prévu dans la section 2.2.5, l'utilisation de l'algorithme non-déterministe stabilise l'apprentissage car il effectue une moyenne sur les récompenses reçues au fil des épisodes.

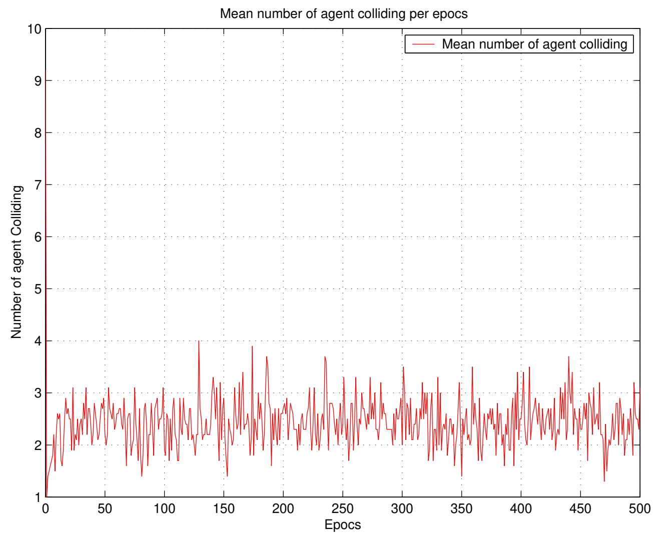


Figure 2.16: Scénario 1a. Nombre moyen d'agents rentrant en collision lors de l'apprentissage de deux agents indépendants dans un environnement sans obstacle. Les moyennes sont calculées sur base de 20 apprentissages. La courbe ne converge pas.

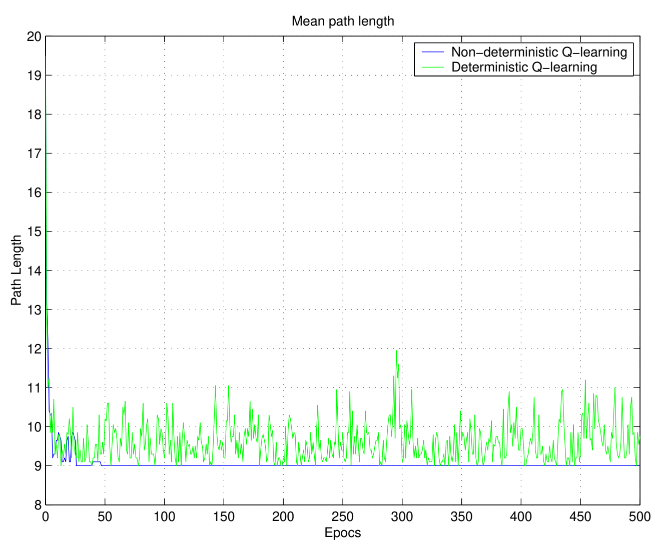


Figure 2.17: Scénario 1a'. Moyennes sur 20 apprentissages des longueurs des chemins dans le cas de deux agents indépendants dans un environnement sans obstacle, en appliquant l'algorithme non-déterministe. On observe une convergence vers la valeur optimale.

Il en est de même pour le nombre d'agents entrant en collision qui oscille autour de 1. Les collisions qui subsistent résultent du fait que les agents indépendants ne se voient pas à une distance supérieure à une case (cf. section 2.2.5).

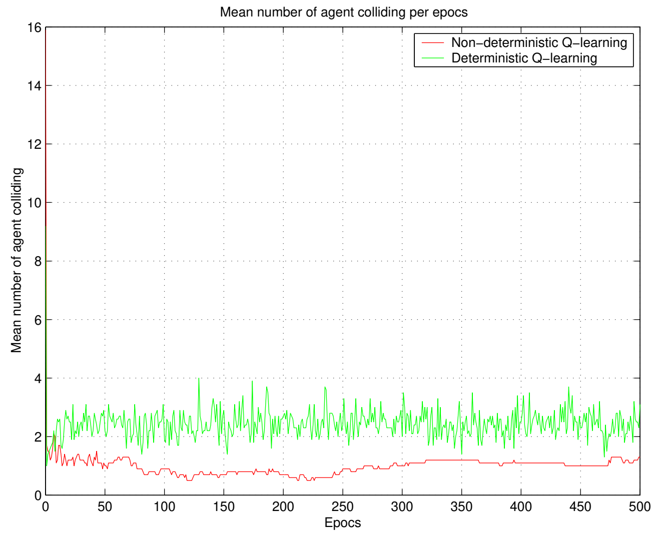


Figure 2.18: Scénario $1a'$. Effet du Q learning non-déterministe sur le nombre moyen de collisions. Deux agents indépendants dans un environnement sans obstacle.

Scénario $1a''$

Il s'agit d'exécuter un ensemble de simulations utilisant la configuration du scénario $1a$ ci-dessus où l'état d'un agent comprend les données des capteurs de proximité en plus de sa position. Nous avons dès lors

$$Etat =< Env, Position, Goal >$$

Aucune amélioration n'est observable. Ceci est la conséquence du fait que la configuration de l'environnement de test est une grille de taille 5×5. Les agents se déplaçant en même temps et étant initialement séparés par un nombre impair de cases, ils ne peuvent se retrouver dans des cases adjacentes. Les collisions "frontales" sont par conséquent impossibles, or le but des capteurs est d'aider à éviter ces collisions.

Scénario $1b$

Ce scénario utilise la configuration du scenario $1a$ ci-dessus avec une fonction d'exploration probabiliste :

Fonction d'exploration : il s'agit de la fonction d'exploration de Boltzmann avec une valeur initiale de 100.0 pour la température. Celle-ci décroit au fil des itérations d'un épisode. L'utilisation de cette valeur devrait permettre d'obtenir une exploration importante au début de l'apprentissage ainsi qu'une forte utilisation des connaissances vers la fin.

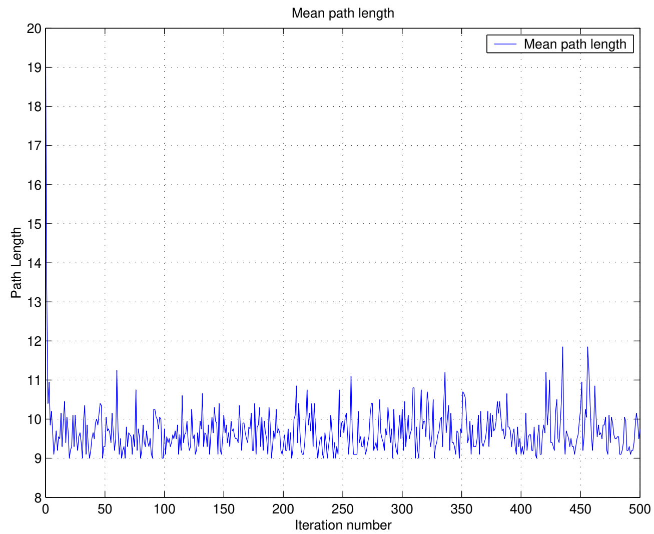


Figure 2.19: Scénario $1a''$. Effet des données des capteurs de proximité sur la courbe des longueurs des chemins. Aucune amélioration n'est observable.

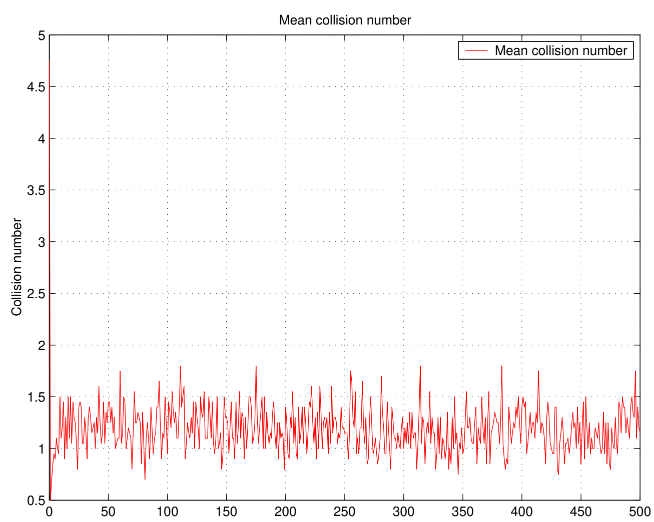


Figure 2.20: Scénario $1a''$. Effet des données des capteurs de proximité sur la courbe du nombre d'agents entrant en collision. Aucune amélioration ne se produit dans le cas d'une grille carrée de dimension impaire.

La figure 2.21 compare la courbe des longueurs des chemins dans le cas d'une exploration aléatoire (courbe verte) et d'une exploration probabiliste (courbe bleue). En raison de la grande valeur de la température, le début des courbes est semblable. Après 200 à 250 épisodes, les courbes diffèrent fortement : la longueur des chemins augmente dans le cas de la fonction d'exploration probabiliste. Ceci est dû au fait que la chute du nombre de collisions est compensée par un chemin plus long. On observe un phénomène de convergence de l'algorithme Q learning vers une valeur sous-optimale, la température étant trop faible pour explorer de nouveaux états (cfr section 1.3.7).

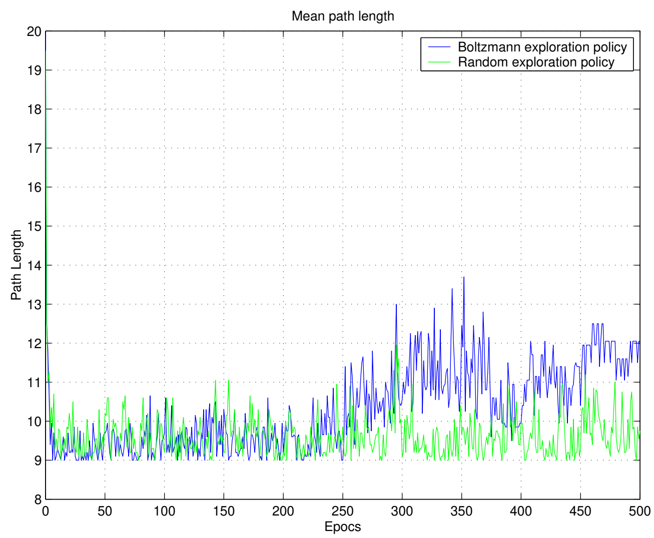


Figure 2.21: Scénario 1b. Influence du choix probabiliste des actions d'exploration sur la moyenne des longueurs des chemins dans le cas d'agents indépendants.

La courbe des collisions 2.22 évolue de manière opposée. On observe en effet une décroissance significative du nombre moyen de collisions à partir de 200 épisodes, contrairement à la courbe verte. Ce nombre moyen de collisions converge vers zéro. Cette convergence était impossible à atteindre dans le cas de l'exploration aléatoire. Dans le cas de l'exploration probabiliste, les actions menant aux plus grandes récompenses ont été choisies plus souvent ce qui explique la diminution du nombre de collisions (cf. section 1.3.7).

Scénario 2a

Les deuxièmes scénarios mettent en oeuvre deux agents ayant chacun des goals différents, dans un environnement contenant des obstacles. Cet environnement est illustré par la figure 2.23.

Environnement : une grille de taille 5×5 contenant des obstacles aux positions $(2, 2)$, $(2, 4)$,$(4, 1)$ et $(4, 5)$.

Agents : il y a deux agents dans l'environnement. L'un débute son apprentissage à la position $(1, 1)$ et doit atteindre la position $(5, 5)$. Le deuxième part de $(1, 5)$ et doit arriver en $(5, 1)$. L'état d'un agent est représenté par le couple $< Position, Goal >$.

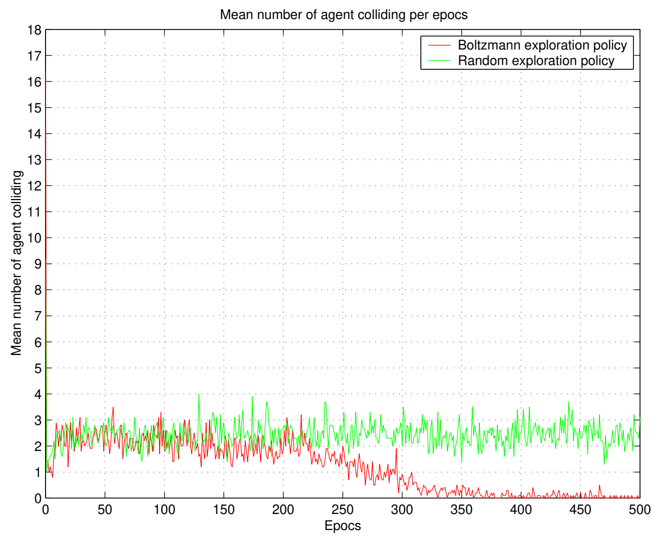


Figure 2.22: Scénario 1*b*. Influence du choix probabiliste des actions d'exploration sur le nombre moyen de collisions dans le cas d'agents indépendants.

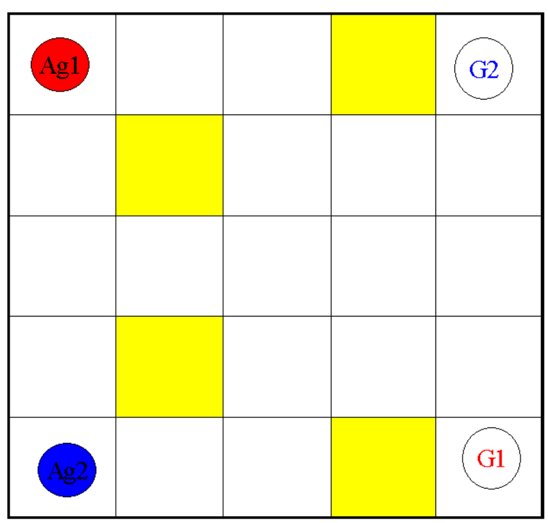


Figure 2.23: Scénario de test pour deux agents dans un environnement avec des obstacles. L'agent Ag_1 part de la position $(1, 1)$ et doit atteindre son goal G_1 en $(5, 5)$. De même que l'agent Ag_2 part de $(1, 5)$ et doit atteindre la position $(5, 1)$. Les zones en jaune constituent les obstacles.

Fonction d'assignation des récompenses : si des agents entrent en collision, ceux impliqués dans la collision reçoivent une récompense négative de -20.0. La punition reçue est de -10.0 lors d'une collision de l'agent avec un obstacle. Enfin, chaque agent reçoit individuellement une récompense de $+20.0$ lorsqu'il atteint son goal.

Fonction d'exploration : les agents explorent l'environnement de manière aléatoire.

Le choix des paramètres et leur impact sur la qualité de l'apprentissage sera traité ultérieurement dans la section 2.3.5.

Résultats et interprétation

Les conclusions tirées du scénario 1a (section 2.3.2) ci-dessus sont également applicables dans le cas présent. Les longueurs des chemins trouvés (figure 2.24) par les agents oscillent autour de la valeur optimale sans jamais y converger. De même, pour les collisions (figure 2.25), on observe un plus grand nombre de collisions dues à la présence d'obstacles, mais cela ne change en rien la mauvaise qualité de l'apprentissage obtenue en 2.3.2.

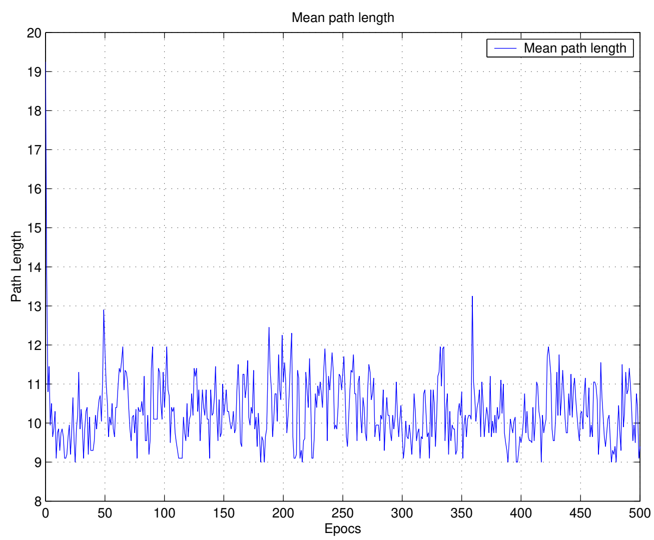


Figure 2.24: Scénario 2a. Moyennes sur base de 20 apprentissages des longueurs des chemins trouvés par deux agents indépendants dans un environnement contenant des obstacles. Aucune coordination n'est atteinte et la longueur des chemins trouvés est instable.

Scénario 2b

Fonction d'exploration : il s'agit de la fonction d'exploration de Boltzmann avec une valeur initiale de 100.0 pour la température. Elle est identique à celle définie dans le scénario 1b à la section 2.3.2.

Résultats et interprétation

L'utilisation de la fonction d'exploration probabiliste permet de trouver un chemin sous-optimal sans collision. On observe sur la figure 2.26 une stabilisation de la longueur des

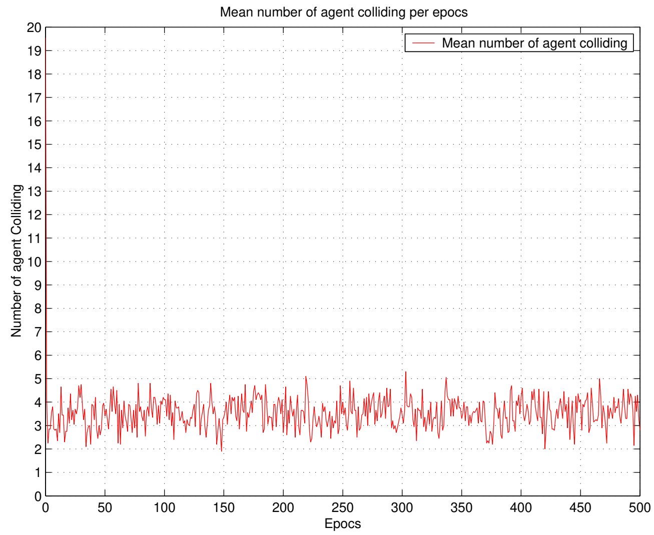


Figure 2.25: Scénario 2a. Nombre moyen de collisions, sur 20 apprentissages, occasionnées par deux agents dans un environnement contenant des obstacles. Aucune coordination n'est atteinte et le nombre de collisions entre agents et obstacles est instable.

chemins à une valeur moyenne sous-optimale de 9,3, et ce après 420 épisodes. Cette stabilisation de la longueur des chemins s'accompagne de la convergence du nombre de collisions *agent-obstacle* et *agent-agent* vers une valeur proche de 0,8. Cette convergence est illustrée par la figure 2.27.

Conclusion

L'apprentissage individuel des agents souffre des maux prévus lors de la modélisation du problème. Les résultats obtenus montrent en effet une incapacité des agents à obtenir une politique commune optimale. Cela se traduit par la non-convergence des courbes présentées ci-dessus. On note cependant que l'utilisation d'une fonction d'exploration probabiliste (section 2.3.2) ainsi que la prise en compte du non-déterminisme (section 2.3.2) améliorent la qualité de l'apprentissage sans pour autant fournir des valeurs optimales pour la courbe des longueurs des chemins *et* celle des collisions.

La section suivante présente les résultats obtenus lors de la mise en oeuvre de l'approche présentée dans la section 2.2.6 dans laquelle la coordination des agents est rendue explicite.

2.3.3 Agents collaboratifs

Cette section met en oeuvre une coordination explicite des agents, qui se base sur une table $\hat{Q}$ jointe, comme décrit dans la section 2.2.6. Ces scénarios sont similaires à ceux mis en oeuvre pour les agents individualistes détaillés dans la section 2.3.2 par la figure 2.14. Les modifications résident dans la fonction d'attribution des récompenses et dans le comportement des agents.

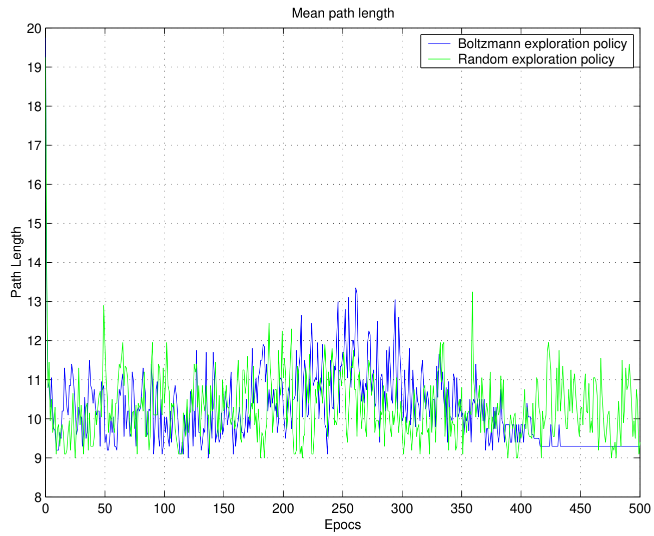


Figure 2.26: Scénario 2*b*. Moyennes sur 20 apprentissages des longueurs des chemins trouvés par deux agents indépendants dans un environnement comportant des obstacles. L'environnement est exploré en utilisant une fonction de décision probabiliste. On observe une stabilisation de la longueur des chemins à une valeur sous-optimale.

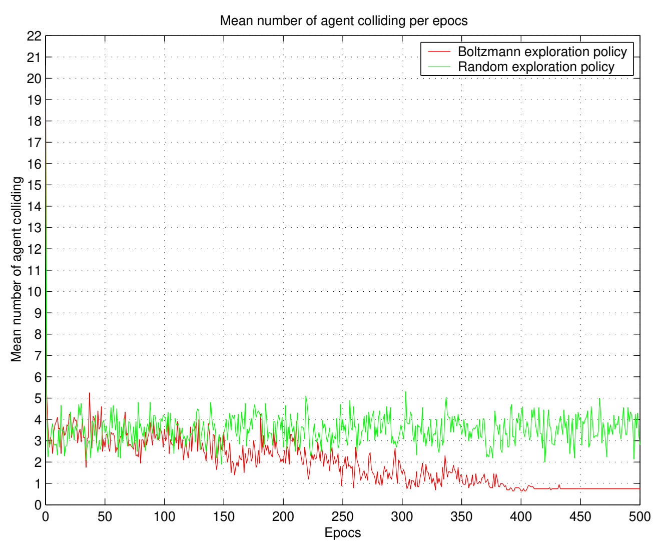


Figure 2.27: Scénario 2*b*. Nombre moyen de collisions, sur 20 apprentissages, occasionnées par deux agents indépendants, dans un environnement comportant des obstacles. L'environnement est exploré en utilisant une fonction de décision probabiliste. Stabilisation du nombre de collisions à une valeur sous-optimale.

Scénario $1a$

Agents : les agents sont collaboratifs dans le sens défini dans la section 2.2.6. L'état est représenté par le tuple $< Position_1, Goal_1, Position_2, Goal_2 >$

Fonction d'assignation des récompenses : si deux agents entrent en collision, ils reçoivent tous les deux une pénalité de -20.0. De même une récompense positive de $+20.0$ est donnée aux agents terminant leur parcours en dernier.

Objectif : les agents doivent maximiser leur récompense globale, c'est-à-dire la somme des récompenses reçues par chacun des agents. Par conséquent, la fonction d'assignation des récompenses privilégie une arrivée simultanée des agents à leur goal respectif, la récompense globale étant plus grande s'ils terminent leurs parcours en même temps.

Résultats et interprétation

La figure 2.28 illustre la courbe moyenne de la longueur des chemins allant de la position initiale jusqu'à la position finale. Le graphique 2.30 décrit la moyenne des nombres d'agents entrant en collision lors de chaque épisode. Ces moyennes sont calculées sur base de 20 apprentissages, chacun étant composé de 500 épisodes.

Dans le cas de deux agents collaboratifs dans un environnement sans obstacle, la moyenne des longueurs des chemins converge vers la valeur optimale après 100 épisodes. De même, la courbe des collisions entre agents s'annule après 10 épisodes. Comme prévu dans la section 2.2.6, les agents collaboratifs atteignent une meilleure qualité de coordination que les agents indépendants. Le temps de convergence est cependant plus long : 50 épisodes sont nécessaires pour atteindre un chemin de longueur 10, alors que cette même valeur est atteinte après seulement 2 épisodes dans le cas d'agents indépendants. Il est néanmoins important de rappeler que, dans ce dernier cas, la longueur des chemins ne se stabilise pas si la fonction d'exploration est aléatoire.

Scénario $1b$

Dans ce scénario, seule la fonction d'exploration change par rapport au scénario $1a$.

Fonction d'exploration : il s'agit de la fonction d'exploration de Boltzmann avec une valeur initiale de 100.0 pour la température. Elle est identique à celle définie à la section 2.3.2.

Résultats et interprétation

Les figures 2.31 et 2.32 illustrent les résultats de simulation et leur comparaison avec les courbes obtenues par exploration aléatoire. Les résultats sont très similaires mais les courbes relatives à l'exploration probabiliste fournissent des résultats légèrement meilleurs. Les courbes bleue et rouge se situent en effet la plupart du temps en dessous de la courbe verte.

Scénario $1c$ **et** $1d$

La configuration des scénarios $1c$ et $1d$ est identique à celle des scénarios respectivement $1a$ et $1b$, à l'exception du nombre d'agents qui est porté à quatre. La figure 2.34 illustre le scénario $1c$.

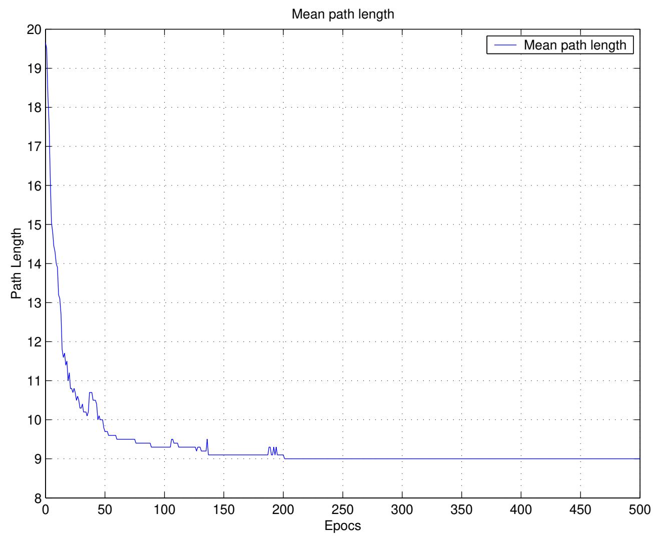


Figure 2.28: Scénario 1a. Moyennes sur 20 apprentissages des longueurs des chemins trouvés par deux agents collaboratifs dans un environnement sans obstacle. Exploration aléatoire et convergence après 100 épisodes vers le chemin optimal.

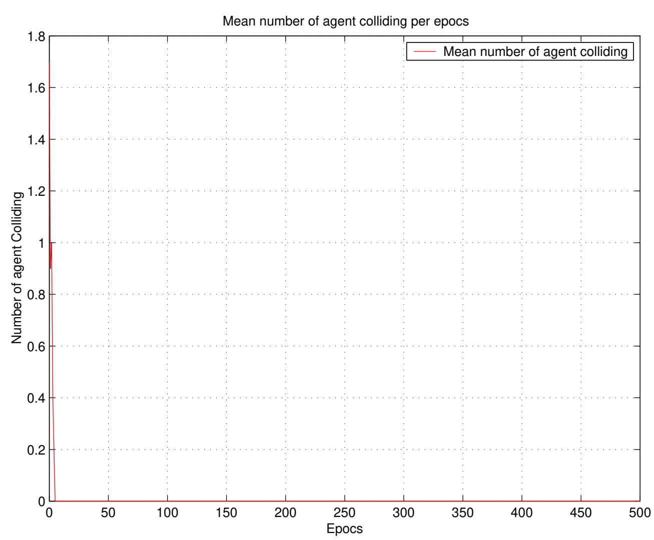


Figure 2.29: Scénario 1a. Nombre moyen de collisions, sur 20 apprentissages, subies par deux agents dans un environnement sans obstacle (scénario 1a). Exploration aléatoire et convergence vers zéro collision après 10 épisodes.

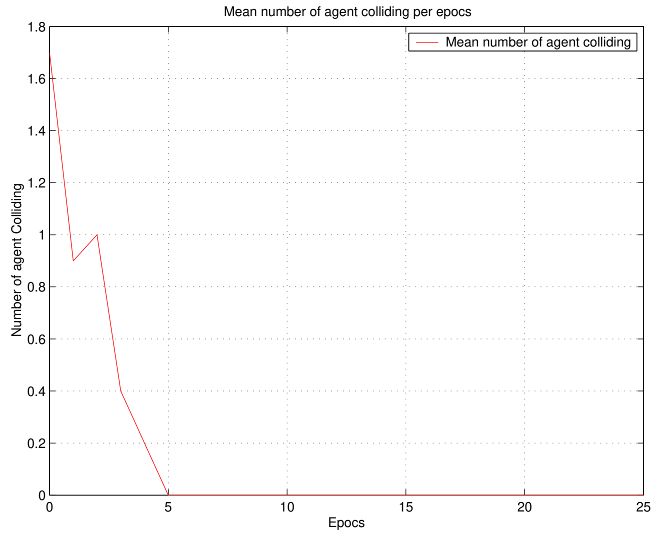


Figure 2.30: Scénario 1a. Nombre moyen de collisions, sur 20 apprentissages, subies par deux agents collaboratifs dans un environnement sans obstacle. Convergence vers zéro collision après 10 épisodes (zoom).

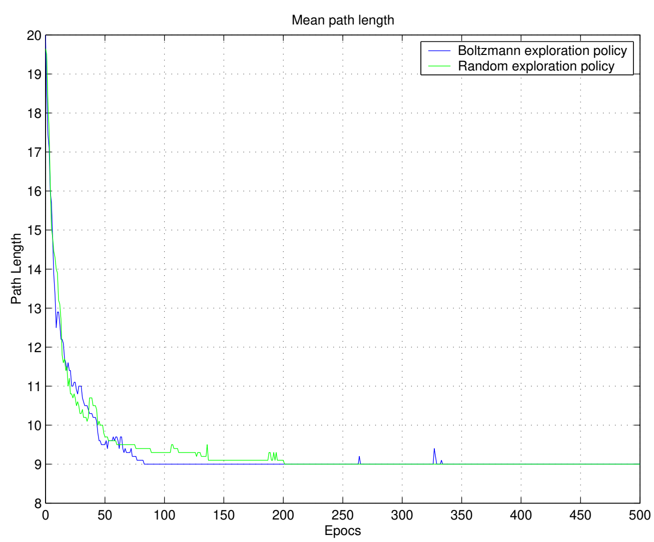


Figure 2.31: Scénario 1b. Moyennes sur 20 apprentissages des longueurs des chemins trouvés par deux agents collaboratifs dans un environnement sans obstacle. L'utilisation d'une fonction d'exploration probabiliste offre des résultats légèrement meilleurs que dans le cas d'une exploration aléatoire.

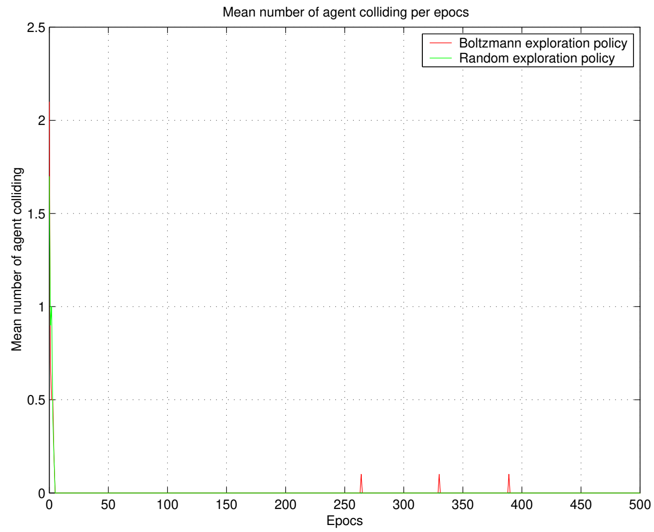


Figure 2.32: Scénario 1b. Moyennes des collisions subies par deux agents collaboratifs dans un environnement sans obstacle. L'utilisation d'une fonction d'exploration probabiliste offre des résultats légèrement meilleurs que dans le cas d'une exploration aléatoire.

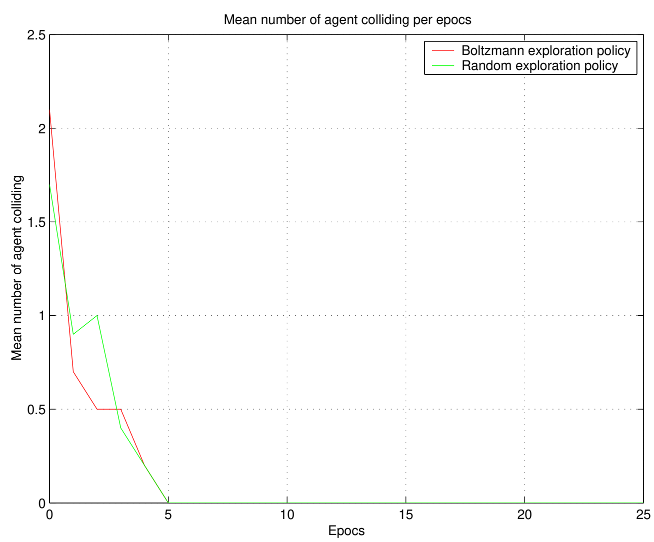


Figure 2.33: Scénario 1b. Moyennes sur 20 apprentissages du nombre de collisions subies par deux agents collaboratifs dans un environnement sans obstacle. L'utilisation d'une fonction d'exploration probabiliste offre des résultats légèrement meilleurs que dans le cas d'une exploration aléatoire (zoom).

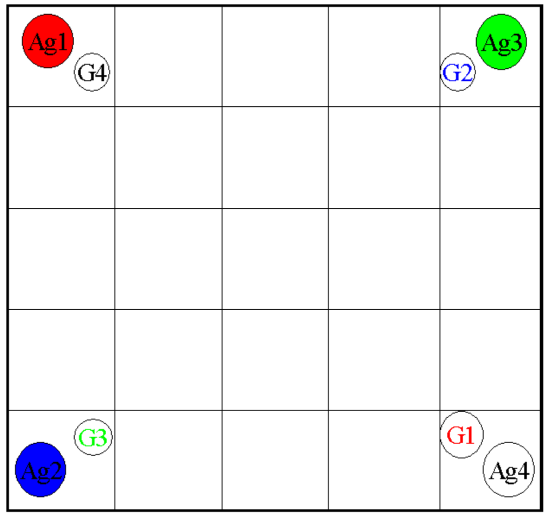


Figure 2.34: Scénario de test pour quatre agents. L'agent Ag_1 part de la position $(1,1)$ et doit atteindre son goal G_1 en $(5,5)$, l'agent Ag_2 part de la position $(1,5)$ et doit atteindre son goal G_2 en $(5,1)$, l'agent Ag_3 part de la position $(5,1)$ et doit atteindre son goal G_3 en $(1,5)$ et finalement l'agent Ag_4 part de $(5,5)$ et doit atteindre son goal G_4 en $(1,1)$.

Résultats et interprétation

Exploration aléatoire Les résultats obtenus avec une exploration aléatoire montrent que l'algorithme converge. Ils sont illustrés par les figures 2.35 et 2.36. Le nombre d'épisodes utilisés pour la simulation fut cependant insuffisant pour obtenir un chemin de longueur optimale (i.e. 9). Au niveau des collisions, la valeur nulle est cependant atteinte après 500 épisodes d'apprentissage. En ce qui concerne les longueurs des chemins, un millier d'épisodes supplémentaires devraient permettre d'obtenir la valeur optimale.

Exploration probabiliste Les figures 2.37 et 2.38 comparent les résultats ci-dessus avec ceux obtenus en utilisant une fonction d'exploration probabiliste. Les nouveaux résultats sont légèrement meilleurs que dans le cas de l'exploration aléatoire, ils confirment donc les prévisions. La courbe des longueurs des chemins (bleue) converge vers la valeur optimale et reste la plupart du temps en dessous de la courbe verte correspondante. L'effet de l'approche probabiliste est de réduire les oscillations par rapport à l'exploration aléatoire, la similarité des épisodes augmentant lorsque la température décroît. La courbe des collisions donne par contre de moins bons résultats que dans le cas de mouvements aléatoires.

Scénario $2a$

Des obstacles illustrés par la figure 2.23 sont introduits dans la configuration du scénario $1a$ ci-dessus et la fonction de récompense est redéfinie comme suit :

Fonction d'assignation des récompenses : si deux agents entrent en collision, ils reçoivent tous les deux une pénalité de -20.0. Par contre, si la collision s'est produite avec un obstacle, la punition est de -10.0. Enfin, une récompense positive de $+20.0$ est donnée aux agents achevant leur parcours en dernier.

Figure 2.35: Scénario 1*c*. Moyennes sur 20 apprentissages des longueurs des chemins obtenus par quatre agents collaboratifs. Ces derniers explorent aléatoirement un environnement sans obstacle. On observe une convergence des chemins vers un chemin optimal mais le nombre d'épisodes d'apprentissage reste insuffisant pour atteindre la valeur optimale.

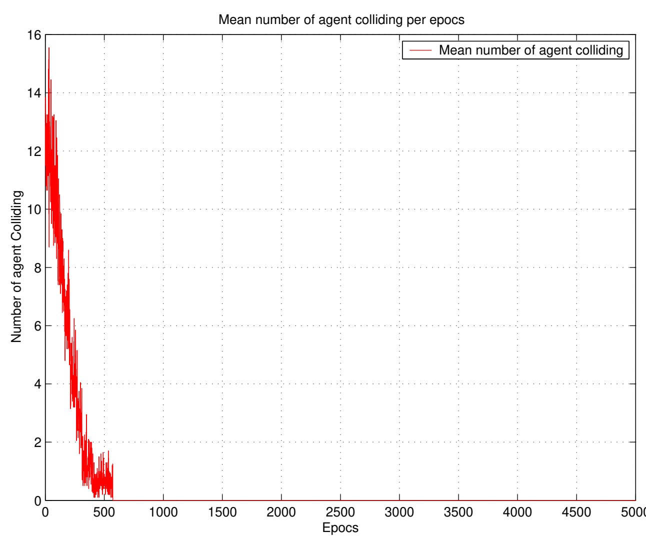


Figure 2.36: Scénario 1*c*. Nombre moyen de collisions, sur 20 apprentissages, occasionnées par quatre agents collaboratifs explorant aléatoirement un environnement sans obstacle. On constate une convergence du nombre de collisions vers la valeur optimale en 500 épisodes.

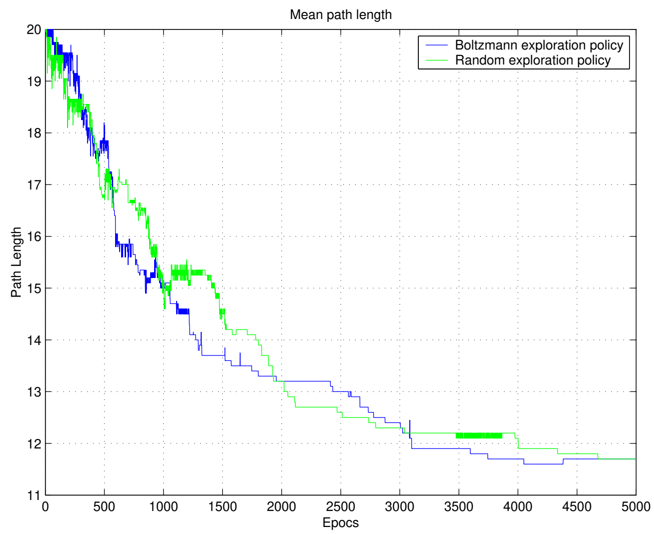


Figure 2.37: Scénario 1d. Moyennes, sur 20 apprentissages, des longueurs des chemins obtenus par quatre agents collaboratifs explorant de manière probabiliste l'environnement sans obstacle. Comparaison avec l'apprentissage basé sur l'exploration aléatoire. Les résultats obtenus avec une exploration probabiliste sont meilleurs.

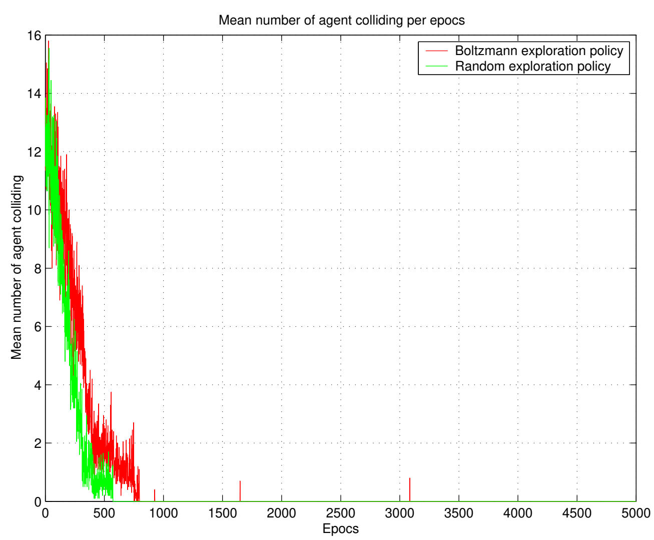


Figure 2.38: Scénario 1d. Nombre moyen de collisions, sur 20 apprentissages, occasionnées par quatre agents collaboratifs explorant de manière probabiliste l'environnement sans obstacle. Comparaison de l'apprentissage avec le cas d'une exploration aléatoire. Les résultats sont moins bons mais la convergence reste atteinte.

Objectif : maximisation de la récompense globale des agents, c'est-à-dire de la somme des récompenses reçues par chacun des agents. Cela implique la minimisation du nombre de collisions entre agents et avec les obstacles.

Résultats et interprétation

Exploration aléatoire Les résultats obtenus dans les figures 2.39 et 2.40 montrent qu'en présence de deux agents et d'obstacles dans l'environnement, l'algorithme est capable d'atteindre un degré de coordination permettant de suivre un chemin optimal sans collision.

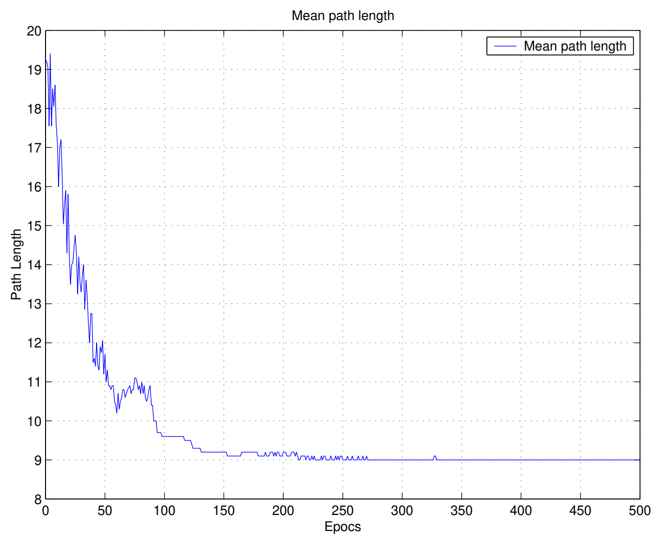


Figure 2.39: Scénario $2a$. Moyenne des longueurs des chemins sous-optimaux trouvés par deux agents collaboratifs dans un environnement contenant des obstacles. La coordination est efficace et la courbe tend vers la valeur optimale.

Scénario $2b$

Fonction d'exploration : il s'agit de la fonction d'exploration de Boltzmann avec une valeur initiale de 100.0 pour la température. Elle est identique à celle définie dans le scénario $1b$ à la section 2.3.2.

Les figures 2.41 et 2.42 illustrent les résultats obtenus. En ce qui concerne la moyenne des longueurs des chemins, on constate que la courbe correspondant à l'approche probabiliste atteint la longueur minimale 40 épisodes avant celle de l'exploration aléatoire. Les graphiques représentant le nombre de collisions par épisode convergent par contre de la même façon. La fonction d'exploration probabiliste produit dès lors un résultat global légèrement meilleur.

Scénario $2c$ et $2d$

La configuration des scénarios $2c$ et $2d$ est identique à celle des scénarios respectivement $2a$ et $2b$, à l'exception du nombre d'agents qui est porté à quatre. La figure 2.43 illustre le scénario $2c$ comportant des obstacles.

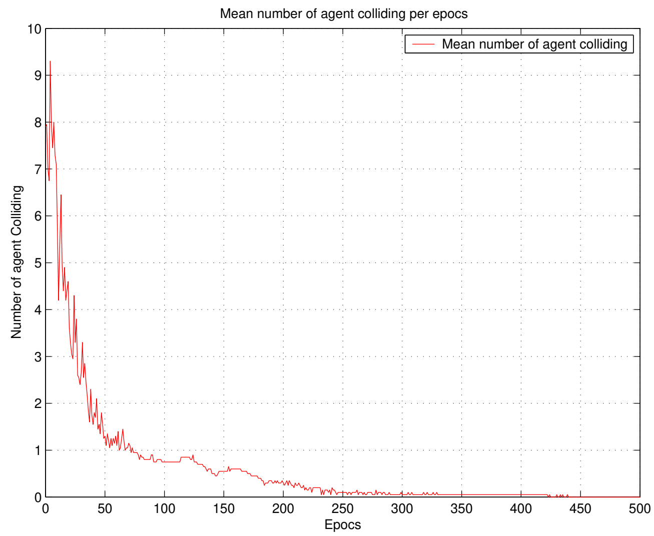


Figure 2.40: Scénario 2a. Nombre moyen de collisions, sur 20 apprentissages, occasionnées par deux agents collaboratifs dans un environnement contenant des obstacles. La coordination est efficace, on constate ainsi une convergence vers zéro de la courbe des collisions entre agents et avec les obstacles.

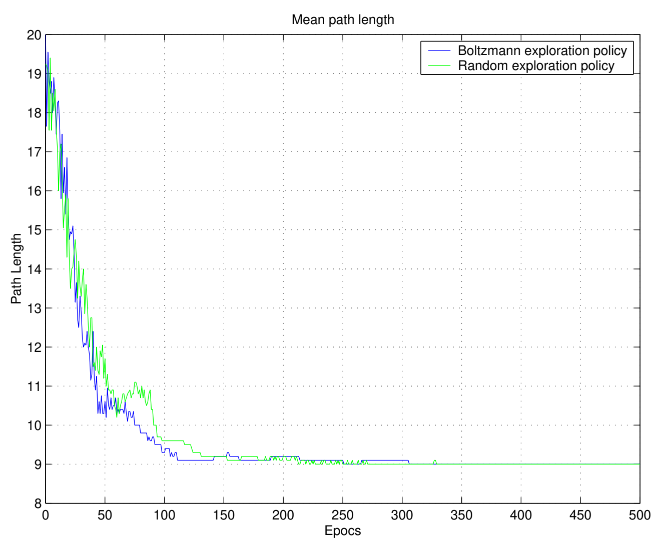


Figure 2.41: Deux agents collaboratifs dans le scénario 2b. Comparaison des moyennes des longueurs des chemins entre les approches probabiliste et aléatoire.

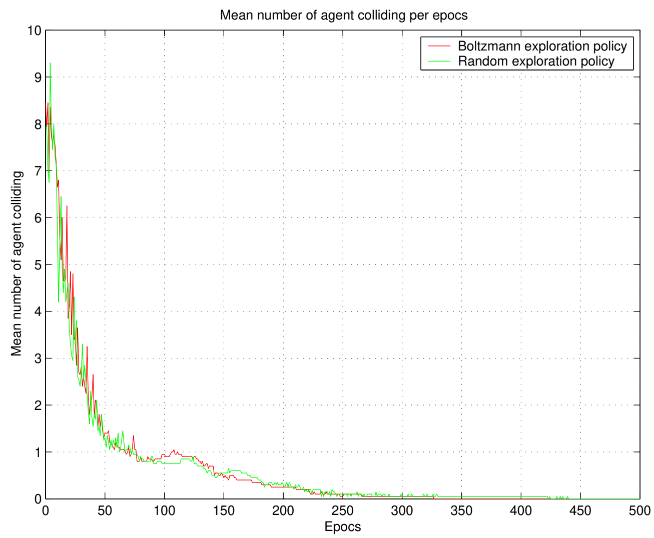


Figure 2.42: Deux agents collaboratifs dans le scénario $2b$. Comparaison des moyennes des nombres de collisions entre les approches probabiliste et aléatoire.

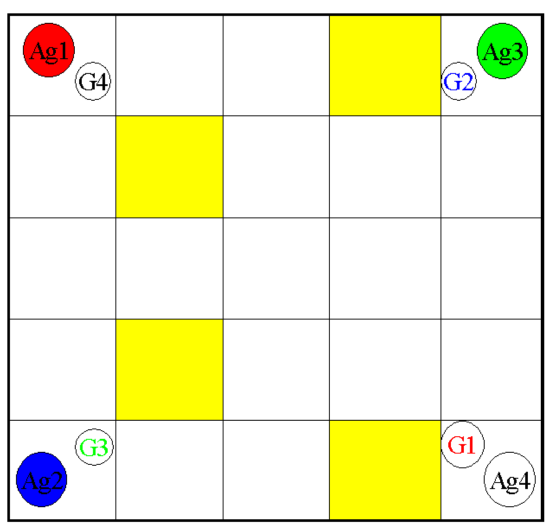


Figure 2.43: Scénario de test pour quatre agents avec obstacles. L'agent Ag_1 part de la position $(1,1)$ et doit atteindre son goal G_1 en $(5,5)$, l'agent Ag_2 part de $(1,5)$ et doit atteindre G_2 en $(5,1)$, l'agent Ag_3 part de la position $(5,1)$ et doit atteindre son goal G_3 en $(1,5)$ et l'agent Ag_4 part de $(5,5)$ et doit atteindre la position G_4 en $(1,1)$

Résultats et interprétation

Les résultats avec une exploration aléatoire illustrés par les figures 2.44 et 2.45 sont nettement inférieurs à ceux obtenus sans obstacle. Après 5000 itérations, l'algorithme ne produit en moyenne qu'un chemin de longueur 18, c'est-à-dire deux fois plus long que le chemin optimal. En outre, on constate qu'en moyenne 16 agents entrent en collision. Ceci est la conséquence des nombreuses récompenses négatives engendrées par les obstacles qui ralentissent la convergence. Une solution offrant de meilleurs résultats est d'augmenter le nombre d'épisodes ou d'utiliser une méthode permettant de visiter moins d'états pour une même qualité d'apprentissage.

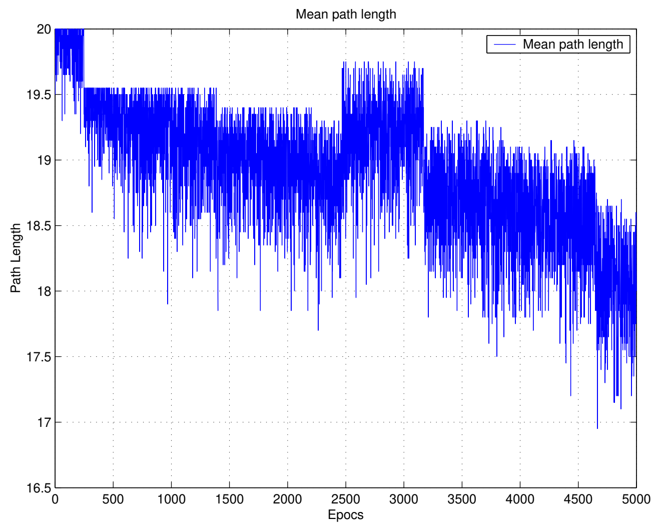


Figure 2.44: Scénario 2*c*. Moyennes, sur 20 apprentissages, des longueurs des chemins empruntés par quatre agents collaboratifs dans un environnement comportant des obstacles. Le nombre d'épisodes est insuffisant pour atteindre la valeur optimale.

Conclusion

Les agents collaboratifs atteignent d'une manière générale un meilleur degré de coordination que des agents indépendants. Lorsque l'on utilise quatre agents, l'explosion de l'espace d'états constitue une limitation sérieuse. Les tests effectués avec deux agents dans une grille de taille 5×5 nécessitent en effet en moyenne 8 heures pour fournir un résultat et dans le cas où l'on rajoute des obstacles le nombre d'épisodes devient insuffisant.

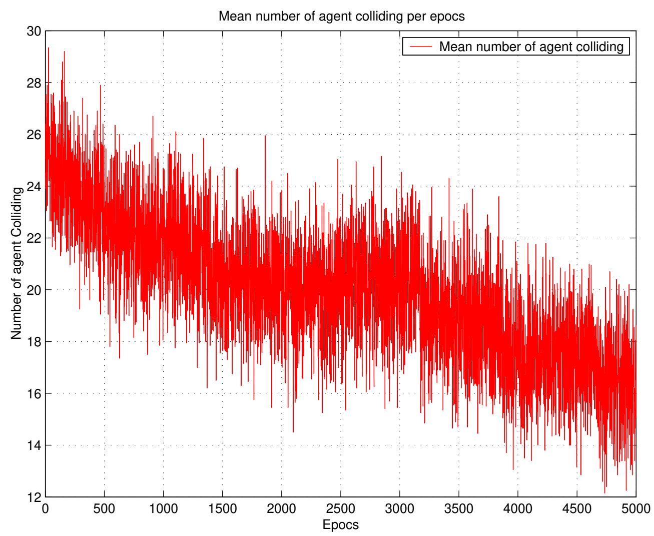


Figure 2.45: Scénario 2*c*. Nombre moyen de collisions, sur 20 apprentissages, occasionnées par quatre agents collaboratifs dans un environnement comportant des obstacles. Le nombre d'épisodes est insuffisant pour atteindre la valeur optimale.

2.3.4 Obstacles dynamiques

Lors des tests réalisés ci-dessus, la position des obstacles utilisés reste fixe tout au long de l'apprentissage. Les agents effectuent donc leurs apprentissages successifs dans un même environnement, contenant des obstacles toujours aux mêmes endroits. Mais qu'en est-il des performances si les agents sont mis dans une situation autre que celle qu'ils ont apprise? Qu'advient-il si nous faisons exécuter à des agents leur apprentissage dans un environnement fixé et qu'ensuite nous les mettons dans un environnement légèrement différent?

L'environnement d'apprentissage utilisé est illustré par la figure 2.23. La politique obtenue par les agents sera testée dans l'environnement de la figure 2.46. Il est nécessaire d'inclure une représentation de l'environnement proche de l'agent dans son état. Sans ces informations additionelles, un agent est incapable de voir si un nouvel obstacle se trouve à proximité afin d'agir en conséquence. Les paramètres de la simulation sont les suivants :

Environnements : l'apprentissage est effectué avec l'environnement décrit par la figure 2.23 et testé avec celui de la figure 2.46.

Agents : il y a deux agents dans l'environnement. L'un débute son apprentissage à la position $(1, 1)$ et doit atteindre la position $(5, 5)$. Le deuxième part de $(1, 5)$ et doit arriver en $(5, 1)$. L'état d'un agent est représenté par le tuple $< Position, Goal, Environnement >$.

Fonction d'assignation des récompenses : si deux agents entrent en collision, ils reçoivent tous les deux une pénalité de -20.0. De même, une récompense positive de $+20.0$ est donnée à ceux terminant leur parcours en dernier. Enfin, un agent reçoit une punition de -10.0 lorsqu'il entre en collision avec un obstacle.

Fonction d'exploration : les agents explorent l'environnement de manière aléatoire.

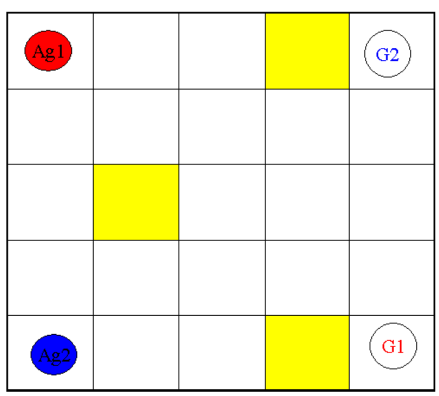


Figure 2.46: Nouvel environnement de test, la position des obstacles est modifiée.

Les figures 2.47 et 2.48 illustrent les résultats obtenus dans l'environnement d'apprentissage. On voit que les valeurs optimales pour la longueur des chemins et le nombre de collisions ont été atteintes. Un des chemins empruntés est dessiné dans la figure 2.49. On peut noter que les courbes d'apprentissage ne sont pas très différentes du cas où l'on n'inclut pas l'environnement proche dans l'état des agents.

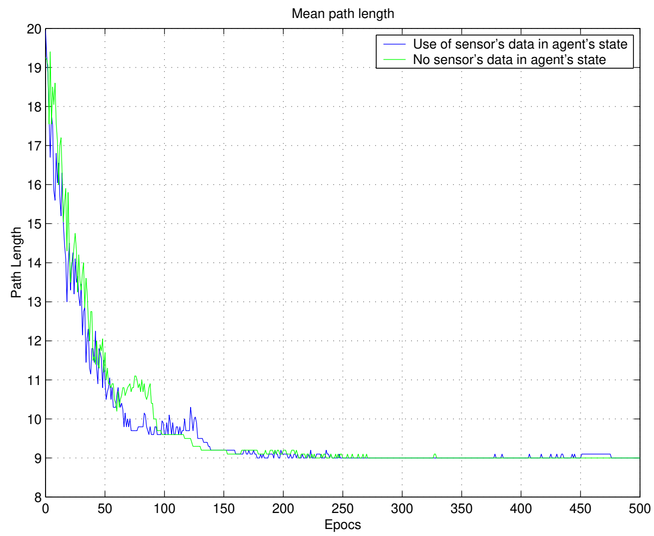


Figure 2.47: Scénario $2a$. Moyennes, sur 20 apprentissages, des longueurs des chemins empruntés après chaque épisode par deux agents utilisant les données des capteurs de proximité lors de l'apprentissage. On observe une convergence vers la valeur optimale 9.

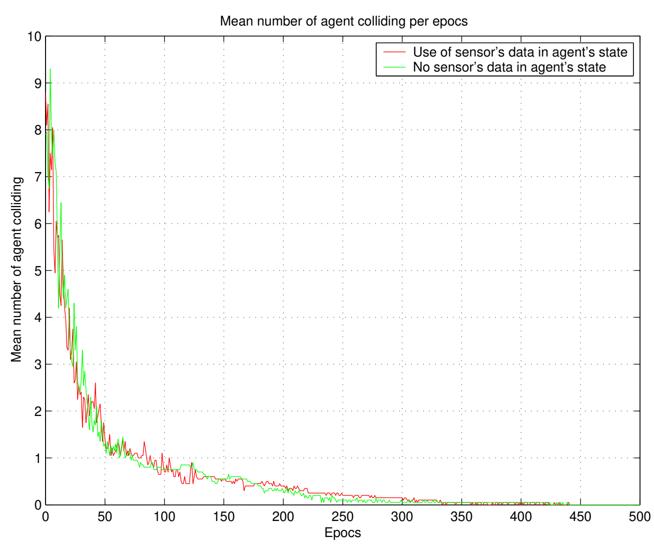


Figure 2.48: Scénario $2a$. Nombre moyen de collisions occasionnées par deux agents utilisant les données des capteurs de proximité dans un environnement avec obstacles. On observe également une convergence vers l'optimum zéro.

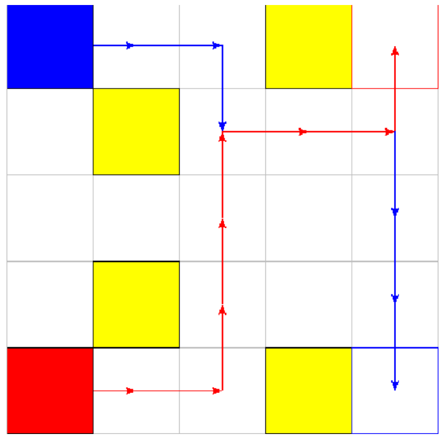


Figure 2.49: Scénario 2a. Exemple de chemins optimaux trouvés par les agents utilisant les données des capteurs de proximité.

L'environnement est modifié pour qu'il soit conforme à celui de la figure 2.46, et l'on y fait se déplacer les agents. La figure 2.50 montre le chemin emprunté par ces derniers. On remarque que les agents se perdent car l'apprentissage s'est effectué avec des obstacles fixes durant tous les épisodes, les agents n'ont donc pas appris à éviter les nouveaux obstacles. On peut noter que si les agents n'utilisent pas les données de leurs capteurs, ils suivent alors toujours le même chemin sans tenir compte des obstacles à proximité.

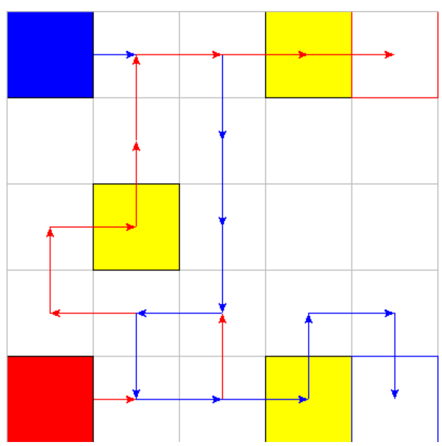


Figure 2.50: Scénario 2a modifié. Utilisation des données des capteurs de proximité. Modification de l'environnement, exemple de chemins trouvés par les agents. Les agents se perdent suite aux états inconnus rencontrés.

Pour remédier à cela, il faut rendre l'apprentissage moins dépendant de la position des obstacles. Les obstacles sont dès lors distribués aléatoirement dans l'environnement durant l'apprentissage et la politique apprise est testée sur une configuration fixe.

Q learning déterministe

L'algorithme Q learning classique sert ici à effectuer un apprentissage avec des obstacles aléatoires. La politique apprise est testée dans un environnement fixe illustré par la figure 2.46. Il faut cependant que la densité probabilité de la présence d'un obstacle soit la même dans l'environnement d'apprentissage et dans celui de test.

Les figures 2.51 et 2.52 montrent les résultats obtenus en utilisant des obstacles non fixés pour l'apprentissage ainsi qu'une politique d'exploration aléatoire. On note que l'apprentissage est loin d'avoir convergé en 5000 itérations. Ceci est la conséquence du fait que les obstacles varient lors de chaque épisode. Le nombre d'états possibles augmente par conséquent considérablement, un état comprenant la description de l'environnement de chaque agent. En outre, les obstacles dynamiques entrainent un non-déterminisme car pour un état et une action donnée, l'état résultant peut différer d'un épisode à l'autre.

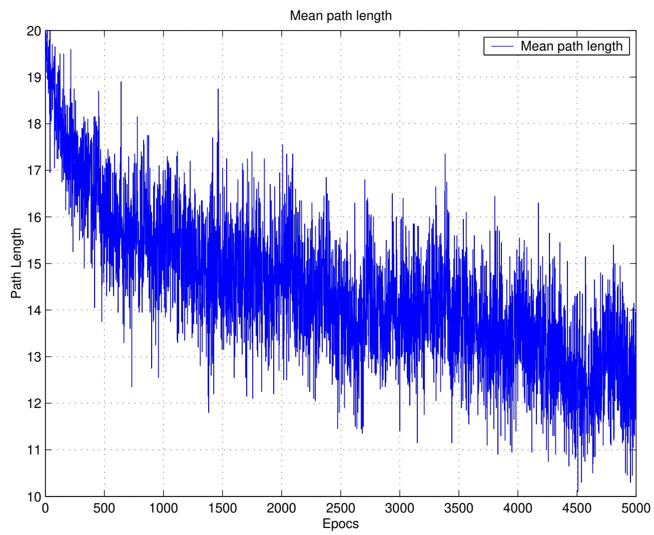


Figure 2.51: Moyennes, sur 20 apprentissages avec des obstacles dynamiques, des longueurs des chemins.

Q learning non-déterministe

L'algorithme Q learning non-déterministe décrit à la section 1.3.8 sert à effectuer un apprentissage comprenant des obstacles aléatoires. Cet algorithme doit résoudre le problème de convergence dû au non-déterminisme engendré par les obstacles dynamiques. Les graphiques sont obtenus en testant la politique apprise dans un environnement fixe illustré par la figure 2.46.

Les figures 2.53 et 2.54 montrent les résultats obtenus en utilisant des obstacles dynamiques ainsi qu'une politique d'exploration aléatoire lors de l'apprentissage. La figure 2.55 illustre des exemples de parcours obtenus en exécutant la politique apprise avec des obstacles aléatoires dans un environnement légèrement modifié.

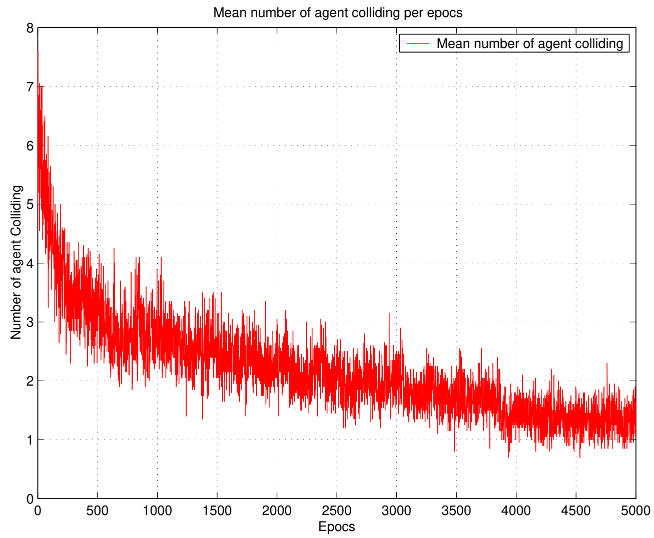


Figure 2.52: Nombre moyen de collisions, sur 20 apprentissages avec des obstacles dynamiques.

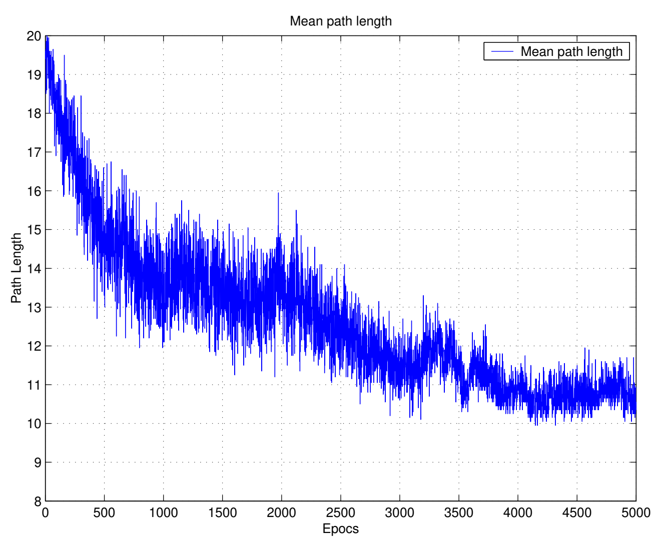


Figure 2.53: Moyenne des courbes des meilleurs chemins trouvés pendant une série de 20 apprentissages avec des obstacles aléatoires et l'utilisation du Q learning non-déterministe. Les mesures sont effectuées dans l'environnement illustré en figure 2.46.

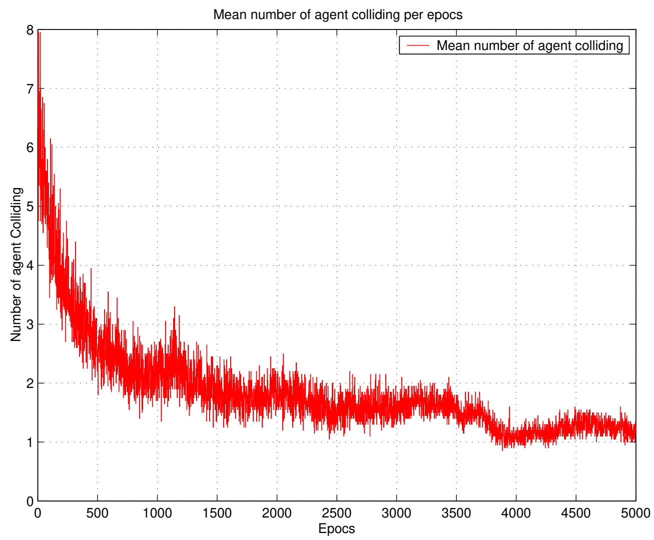


Figure 2.54: Moyenne des courbes du nombre d'agents entrant en collision pendant une série de 20 apprentissages avec des obstacles aléatoires et l'utilisation du Q learning non-déterministe. Les mesures sont effectuées dans l'environnement illustré en figure 2.46.

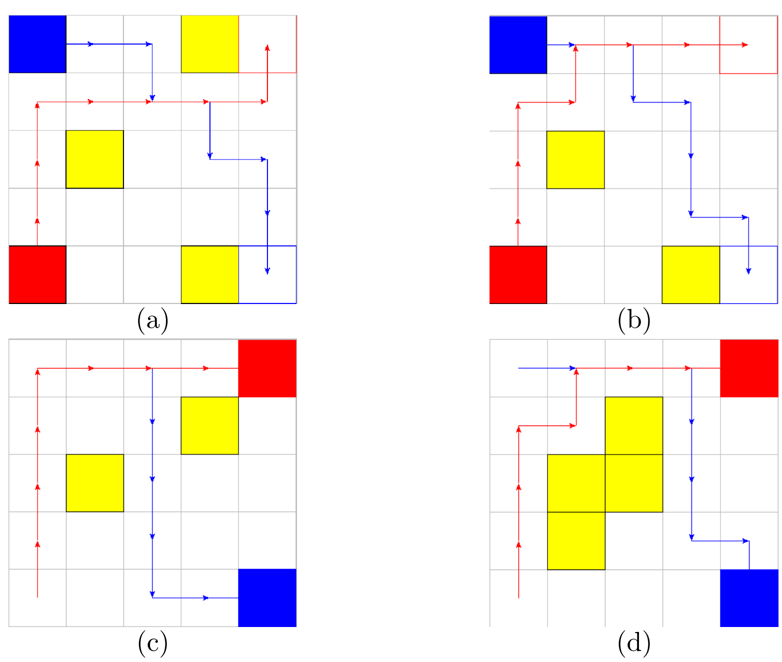


Figure 2.55: Exemple de chemins optimaux dans une configuration modifiée. L'algorithme s'adapte généralement bien à de petites modifications topologiques.

Conclusion

La politique de déplacement obtenue par l'algorithme Q learning est utilisable dans le cas de petites perturbations de l'environnement. Cette adaptativité n'est cependant pas sans limite.

En effet, si de fortes modifications entrainent l'apparition de situations complexes, l'algorithme est incapable de trouver une solution optimale puisque l'environnement est totalement inconnu. Lorsque l'agent se trouve en début de parcours, les actions jointes apprises pour son état courant tiennent compte de l'environnement d'apprentissage global. Or si le nouvel environnement diffère trop de celui de l'apprentissage, la meilleure action jointe pour un état initial risque de causer des problèmes dans le futur car l'agent va anticiper des obstacles différents de ceux présents dans l'environnement de test. Il arrive donc que la politique de décision mette les agents dans des configurations dans lesquelles ils n'ont pas appris à réagir. La figure 2.56 illustre ce cas.

Outre ce problème, une autre limitation inhérente aux Processus de Décision Markoviens apparaît : les agents ne tiennent pas compte dans leurs décisions de leurs états précédents. Si le nouvel environnement pousse les agents à revenir dans un état visité antérieurement, ils continueront à effectuer les mêmes actions et entreront de ce fait dans un cycle sans fin.

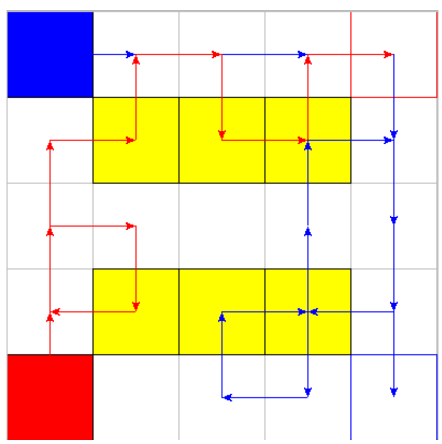


Figure 2.56: Exemple de chemins non optimaux dans une configuration modifiée. L'algorithme n'arrive pas à s'adapter à de grandes modifications topologiques.

2.3.5 Impact des paramètres

Les résultats présentés ci-dessus ont été obtenus en combinant des paramètres sélectionnés en utilisant le *bon sens*. Le but de cette section est de montrer que le choix de ces paramètres influence la qualité de l'apprentissage des agents. Deux paramètres méritent l'attention :

- La fonction d'attribution des récompenses ;
- La valeur de la température de Boltzmann dans le cas de l'exploration basée sur les connaissances acquises.

La fonction d'attribution des récompenses

La fonction d'attribution utilisée pour les tests renvoie une récompense de $+20.0$ aux agents arrivant simultanément en dernier, -20.0 en cas de collision entre agents et enfin -10.0 s'il y a collision entre un agent et un obstacle. Dans le cas d'agents collaboratifs, ces récompenses sont partagées.

La structure de cette fonction privilégie les arrivées groupées par rapport à celles décalées dans le temps. Si un agent atteint son goal avant les autres, il ne recevra pas de récompense immédiate. La valeur des actions globales futures dépendra par contre du nombre d'agents arrivant simultanément en dernier lieu. Il est dès lors plus rentable pour un agent d'allonger son chemin afin d'arriver en même temps que tous les autres, de façon à obtenir une plus grande récompense. On minimise ainsi la longueur des chemins et l'écart entre les longueurs des différents parcours trouvés par les agents. La figure 2.57 montre un scénario mettant en oeuvre trois agents collaboratifs dont les goals sont à des distances différentes. En attribuant les récompenses de goal à chaque arrivée des agents on remarque que ceux-ci minimisent leur chemin individuel. La figure 2.58 illustre un scénario mettant en oeuvre ces mêmes agents avec des goals identiques, mais les récompenses sont attribuées uniquement aux agents atteignant leur goal en dernier. On constate donc que l'agent vert, dont le goal est le plus proche, allonge son chemin de façon à ce qu'il ait la même longueur que celui des autres agents. La récompense globale est plus grande que s'ils étaient arrivés à des temps différents.

Le but d'une telle distribution des récompenses est également de donner moins de poids aux collisions agent-obstacle qu'à celles entre agents. Une collison entre deux agents entraine en effet la perte de deux unités alors que la collision d'un agent avec un obstacle n'en endommage qu'une seule. Les goals prennent la même importance que les collisions entre agents. Ainsi, si deux agents entrent en collision ils reçoivent une pénalité dont l'influence est plus grande que la récompense du goal reçue plus tard.

La fonction d'attribution des récompenses va maintenant subir une modification substantielle, afin d'observer son impact sur l'apprentissage. Dans un premier temps, deux agents évoluent dans un environnement de taille 5×5 dépourvu d'obstacle. On se base donc sur les scénarios $1a$ et $2a$ de la section 2.3.3. La fonction d'attribution des récompenses est définie de la manière suivante :

Fonction d'assignation des récompenses : si deux agents entrent en collision, ils reçoivent tous les deux une récompense négative de -10.0. En outre, chaque agent reçoit une récompense positive de $+50.0$ s'il atteint son goal en dernier. Enfin, il reçoit une punition de -10.0 lorsqu'il entre en collision avec un obstacle.

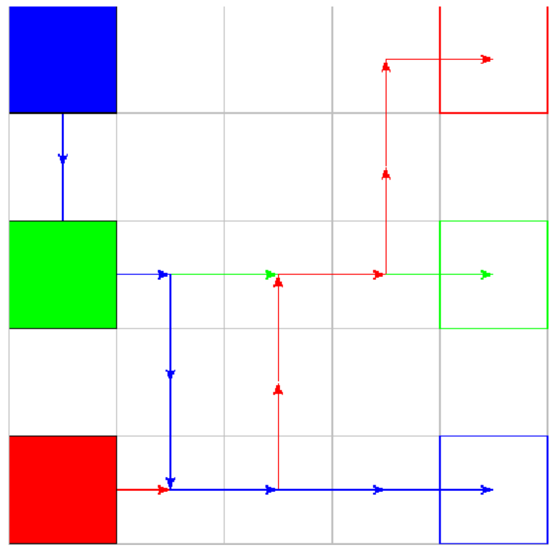


Figure 2.57: Trois agents changeant de position. La fonction de récompense favorise les chemins individuels de longueur minimale.

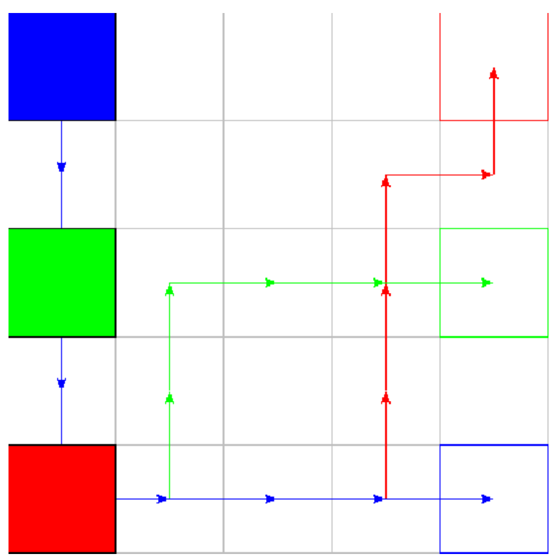
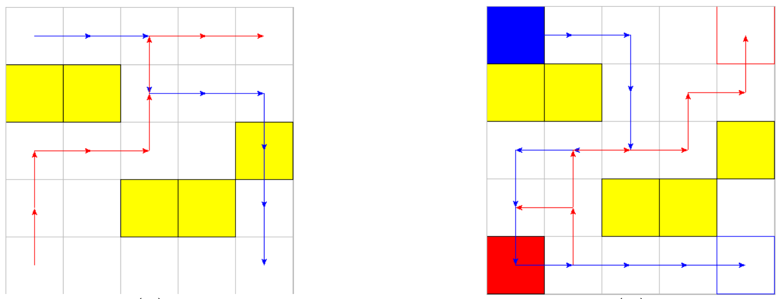


Figure 2.58: Trois agents changeant de position. La fonction de récompense favorise les chemins de même longueur.

Cette fonction a été testée dans le cas de deux agents collaboratifs. La figure 2.59 compare les parcours obtenus avec cette fonction à ceux correspondant à l'attribution d'une récompense de +20.0 lors de l'arrivée au goal et de −30.0 lors d'une collision avec un obstacle. On constate que la première fonction pousse les agents à aller le plus rapidement possible vers leur goal même s'ils doivent passer par un obstacle. Dans le second cas, la pénalité de collision étant plus importante que la récompense de l'arrivée au goal, les agents évitent les obstacles. En outre, on constate que l'agent rouge effectue un cycle en début de parcours de façon à arriver en même temps que l'agent bleu, maximisant ainsi les récompenses totales.



(a) (b)

Figure 2.59: Comparaison des fonctions d'assignation des récompenses. Dans le parcours de la figure (a), les agents préfèrent arriver rapidement au goal plutôt qu'éviter les obstacles. Celui de la figure (b) est obtenu en accordant davantage d'importance aux obstacles qu'aux goals. Les agents arrivent simultanément.

La valeur de la température

Comme expliqué dans la section 1.3.7, la valeur de la température a une grande influence sur la qualité de l'apprentissage. L'utilisation d'une petite valeur produit en effet une solution sous-optimale, aucune exploration n'étant effectuée et l'algorithme utilisant toujours les mêmes Q-valeurs. Le figure 2.60 illustre un apprentissage effectué avec deux agents collaboratifs et une température constante de 1.0. La convergence est très lente du fait que, lors de chaque déplacement, les agents ont une probabilité très faible de visiter de nouveaux états. La moyenne des longueurs des chemins atteinte en 500 itérations est de 11.5, tandis que dans le cas aléatoire les agents atteignent la longueur minimale en 200 itérations. Inversément, une trop grande valeur de la température fournit une qualité d'apprentissage semblable au cas de l'exploration aléatoire.

2.3.6 Limitations

Nombre d'agents

Le nombre d'agents présents dans l'environnement représente une limitation. En effet un grand nombre d'agents implique un espace d'états important à parcourir ainsi qu'un grand nombre de paires état-action à stocker au sein de la table $\hat{Q}$. Ces opérations requièrent donc beaucoup de mémoire. Le tableau 2.1 établit une comparaison de l'espace mémoire occupé par le programme lorsque le nombre d'agents croît, en utilisant un environnement de taille 5×5.

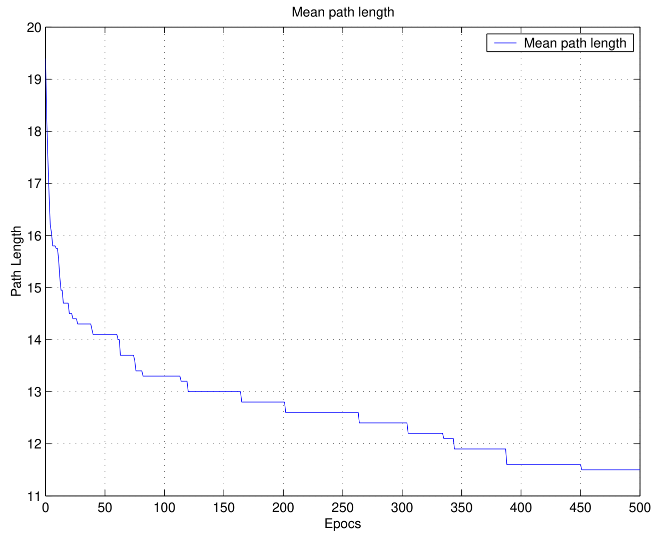


Figure 2.60: Deux agents collaboratifs dans le cas d'une température de Boltzmann constante valant 1.0. La convergence est beaucoup plus lente que dans le cas de mouvements aléatoires.

Dimension de la grille	Nombre d'agents	Nombre d'épisodes	Mémoire utilisée	Temps moyen d'apprentissage	Convergence atteinte
5×5	2	500	<1 MB	35 s	oui
5×5	4	5000	250 MB	16 min	oui
10×10	2	1000	20 MB	3 min	oui
10×10	4	10000	>1 GB	na	na
20×20	2	4000	200 MB	80 min	oui
20×20	4	20000	>1 GB	na	na

Table 2.1: Performances et limites de l'implémentation. La valeur *na* indique que la mémoire requise est trop élevée pour compléter la simulation.

Taille de l'environnement

Pour effectuer les tests ci-dessus, les environnements utilisés étaient de très petite taille. L'évaluation des performances ainsi que des limites de l'implémentation, en fonction des moyens techniques mis à notre disposition, a été répertoriée dans la table 2.1. Lorsque beaucoup d'obstacles sont présents, on peut légèrement réduire la taille de l'environnement en le représentant sous forme de graphe et en regroupant des ensembles d'obstacles en un seul noeud.

2.3.7 Conclusion

Ce chapitre a présenté et discuté les résultats obtenus après implémentation des modélisations décrites dans la section 2.2. Les limitations de chacune des configurations ont également été soulignées, elles peuvent être synthétisées de la façon suivante :

Agents indépendants Les limitations de l'algorithme d'apprentissage d'agents indépendants résident principalement dans l'impossibilité d'obtenir une coordination globale des agents. L'utilisation de l'algorithme Q learning non-déterministe améliore les résulats, mais ne fournit aucune garantie sur la convergence.

Agents collaboratifs Cette méthode offre de bons résultats de coordination mais souffre du problème de l'explosion du nombre d'états à visiter. Cet inconvénient est mis en exergue dans la section 2.3.3 lors de l'introduction d'obstacles dans un environnement de taille réduite où se meuvent quatre agents. Les résultats obtenus ne sont pas satisfaisants.

Le chapitre suivant présente une méthodologie mise en oeuvre afin de contourner cette limitation.

Chapitre 3

Généralisation d'une fonction Q de deux agents

Après la mise en évidence des inconvénients majeurs des méthodes décrites dans le chapitre précédent, nous proposons une nouvelle approche basée sur la généralisation d'une table de deux agents appelée JAG (*Joint Action Generalization*).

3.1 Description de la méthode JAG

La section 2.2 présentait les modèles basés sur une fonction d'utilité propre à chaque agent, ainsi que sur une fonction d'utilité jointe pour tous les agents. Leurs limitations principales étaient :

- Dans le cas d'agents indépendants, la coordination apprise n'était pas explicite. Cela signifie que les agents ne s'évitaient que lorsqu'ils détectaient la proximité d'un autre agent ;
- En ce qui concerne les agents partageant une table Q jointe, le temps de convergence était long suite au grand nombre d'états et d'actions joints à visiter.

Afin de contourner ces problèmes, le présent travail propose une approche intermédiaire par rapport aux deux méthodes évoquées. Il s'agit de la généralisation d'une table jointe $\hat{Q}((s_1, s_2), (a_1, a_2))$ apprise par deux agents, afin d'assurer la coordination d'un nombre quelconque d'agents. Les couples $(s_1, s_2) \in S^2$ et $(a_1, a_2) \in A^2$ sont respectivement les états et les actions des deux agents. En supposant que N_{agents} agents doivent apprendre à se coordonner, cette méthode peut être décrite de la manière suivante :

1. Les entrées de la table $\hat{Q}$ jointe de deux agents sont initialisées à zéro ;
2. Tant que le nombre d'itérations souhaité n'est pas atteint, les étapes suivantes sont répétées :
    - Les N_{agents} agents se déplacent de leur point de départ vers leur goal ;
    - Les $N_{couples}$ couples d'agents possibles sont formés à partir de N_{agents} agents différents. On a donc :

    $$N_{couples} = C^2_{N_{agents}} = \frac{N_{agents}!}{2(N_{agents} - 2)!} \tag{3.1}$$

    où $C^2_{N_{agents}}$ est le nombre de combinaisons de 2 parmi N_{agents} ;

- Les chemins des agents correspondant aux couples formés sont considérés comme une succession d'actions jointes de deux agents. Pour chaque couple, l'algorithme Q learning met à jour les entrées de la table $\hat{Q}$ jointe correspondant aux paires état-action visitées par les deux agents. On a donc $N_{couples}$ parcours de deux agents à apprendre.

3. La table $\hat{Q}$ obtenue est utilisée avec les N_{agents} agents. L'estimation de la fonction Q jointe des agents est la somme des $N_{couples}$ valeurs de la table $\hat{Q}$ pour chacun des couples d'agents. La table 3.1 décrit plus précisément l'algorithme de décision d'un agent.

Cette approche repose donc sur la division de la coordination globale en une coordination de chaque couple d'agents, en utilisant une même estimation de leur fonction d'utilité Q. Une fois la politique optimale pour deux agents connue, l'algorithme de choix de l'action d'un agent parmi N_{agents} est décrit par la table 3.1 et illustré par la figure 3.1.

Algorithme JAG de décision d'un agent i parmi N_{agents} :

1. Détermination de toutes les actions jointes possibles pour l'état courant des N_{agents} agents ;
2. Pour chacune de ces actions jointes, calcul de la somme Q_{totale} des $N_{couples}$ entrées de la table $\hat{Q}$ correspondant à chaque état et action joints des paires d'agents possibles. Ces paires sont non ordonnées ;
3. Identification de l'action jointe de valeur Q_{totale} maximale. Dans le cas de plusieurs valeurs identiques, les agents doivent s'accorder afin de sélectionner la même action ;
4. Exécution de l'action correspondant à l'agent i dans cette action jointe.

Table 3.1: Algorithme JAG de décision d'un agent parmi N_{agents} utilisant une table $\hat{Q}$ pour deux agents. Il est exécuté par chaque agent lors de chacun de ses déplacements.

3.1.1 Avantages

La méthode JAG présente certains avantages par rapport aux approches mettant en oeuvre des agents indépendants ou collaboratifs.

Le nombre de paires état-action à visiter avant d'atteindre la convergence de l'apprentissage d'une table jointe croît exponentiellement avec le nombre d'agents. C'est ce que montre la formule 2.5. Dans le cas d'une grille de dimension $D \times D$ dans laquelle $N_{agents} > 2$ agents peuvent effectuer $N_{actions}$ actions, le nombre de paires état-action possibles est

$$N_{état-action} = (N_{actions} \cdot D^2)^{N_{agents}}$$

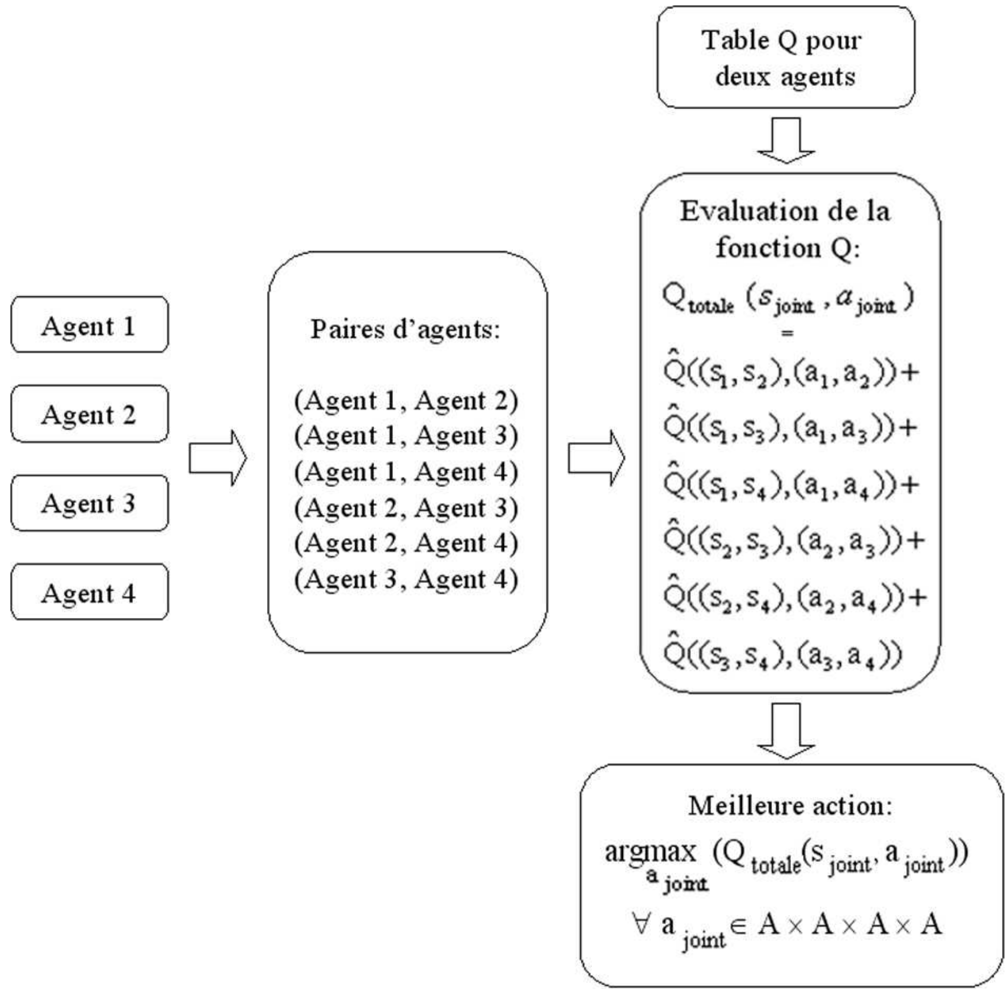


Figure 3.1: Généralisation d'une politique apprise par deux agents à quatre agents, A étant l'ensemble des actions possibles pour un agent.

L'espace des paires état-action possibles pour N_{agents} est donc $(N_{actions} \cdot D^{\ 2})^{N_{agents}-2}$ fois plus grand que dans le cas de deux agents. Le temps de convergence pour une même qualité d'apprentissage sera donc écourté de ce même facteur. La table 3.2 et la figure 3.2 comparent le nombre total de paires état-action à visiter pour une table $\hat{Q}$ jointe classique avec le cas de la généralisation d'une table de deux agents, et ce en fonction de la taille de l'environnement. La méthode JAG engendre un nombre de paires état-action nettement inférieur.

Etant donné que la taille maximale de la table $\hat{Q}$ est proportionelle au nombre total de paires état-action, celle-ci est $(N_{actions} \cdot D^{\ 2})^{N_{agents}-2}$ fois plus réduite dans la méthode JAG que dans le cas d'une table jointe de N_{agents} agents. Stocker cette table nécessitera donc sensiblement moins de mémoire.

Contrairement au cas d'agents indépendants, la coordination est ici globale. L'utilité des actions jointes entre chaque paire d'agents est en effet évaluée, et l'action jointe choisie pour

Configuration	Paires état-action pour une table $\hat{Q}$ de deux agents	Paires état-action pour une table $\hat{Q}$ de quatre agents
$D = 5$	10^4	10^8
$D = 10$	$1,6 \cdot 10^5$	$2,56 \cdot 10^{10}$
$D = 50$	10^8	10^{16}

Table 3.2: Comparaison de taille de l'espace état-action dans le cas de quatre agents entre une table $\hat{Q}$ jointe classique et une table jointe $\hat{Q}$ apprise par deux agents. L'environnement est une grille de taille $D \times D$ et les agents ont chacun quatre actions possibles. L'état d'un agent comporte uniquement sa position.

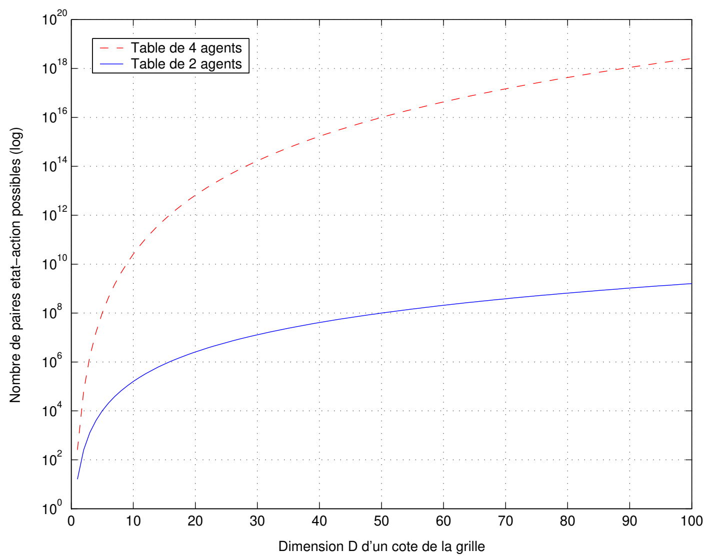


Figure 3.2: Comparaison des tailles de l'espace état-action dans le cas de quatre agents entre une table $\hat{Q}$ jointe classique et une table jointe $\hat{Q}$ apprise par deux agents. L'environnement est une grille de taille $D \times D$ et les agents ont chacun quatre actions possibles. L'état d'un agent comporte uniquement sa position.

N_{agents} agents correspond au maximum des fonctions d'utilités des $N_{couples}$ paires d'agents possibles. La coordination de chacun des couples d'agents assure la coordination globale, pour autant que la politique obtenue pour deux agents soit optimale.

3.1.2 Limitations

Cette approche semble donc intéressante. Il existe toutefois des limitations inhérentes à cette méthode :

Indépendance par rapport aux goals de l'apprentissage

Afin de pouvoir utiliser la fonction d'utilité apprise $\hat{Q}((s_1, s_2), (a_1, a_2))$ avec tous les couples d'agents, la politique optimale obtenue doit être indépendante du goal des agents utilisés lors de l'apprentissage. Si ce n'est pas le cas, la fonction d'utilité ne sera pas applicable aux couples d'agents ayant des goals différents. Une première manière de résoudre ce problème consiste à trouver une représentation d'état indépendante de la position du goal d'un agent. Une telle représentation peut cependant ne pas être assez complète pour assurer un apprentissage correct.

Un autre échapatoire consiste à inclure la position du goal des agents dans leur état, ils vont alors apprendre à se coordonner pour tous les goals utilisés lors de l'apprentissage. La table $\hat{Q}$ sera mise à jour successivement par des couples d'agents possédant des goals différents, elle va donc stocker une politique adéquate pour chacun de ces couples de goals. En comparaison à une représentation d'état sans position du goal, le nombre d'états à visiter pour une qualité d'apprentissage fixée sera donc multiplié par $N_{couples}$.

Indépendance par rapport à l'ordre des entrées

Afin d'utiliser efficacement la fonction d'utilité apprise $\hat{Q}((s_1, s_2), (a_1, a_2))$, il est également nécessaire que l'estimation de la fonction d'utilité $\hat{Q}$ soit indépendante de l'ordre des paires état-action des entrées. Si $\hat{Q}$ a convergé correctement vers la fonction Q, on a

$$\forall\ (s_1, s_2) \in S^2, \forall\ (a_1, a_2) \in A^2,$$
$$\hat{Q}((s_1, s_2), (a_1, a_2)) = \hat{Q}((s_2, s_1), (a_2, a_1))$$

Cette propriété permet de diminuer le nombre d'entrées de la table $\hat{Q}$ à visiter par deux, une même entrée correspondant à deux états joints inversés. Cette symétrie peut être implémentée en triant les états de chaque agent par rapport à leur position dans un même état joint. Si cette condition n'est pas vérifiée, l'utilité jointe calculée dépend alors de l'ordre des éléments de chaque paire d'agents évaluée. Il est en outre nécessaire que chaque agent évalue les paires (s_1, a_1) et (s_2, a_2) dans un ordre identique à celui des autres afin qu'une même action jointe optimale soit choisie par tous.

Temps calcul supplémentaire

Des calculs additionnels sont nécessaires lors de l'utilisation de la politique apprise. Ceci est dû au fait qu'au lieu de déterminer directement quelle est l'action jointe ayant la plus grande valeur dans la table $\hat{Q}$ jointe, il faut désormais évaluer la somme des $N_{couples}$ valeurs. Il ne s'agit toutefois pas d'un accroissement significatif du temps de calcul pour un nombre d'agents raisonnable. En se basant sur l'équation 3.1 on peut calculer que, pour 10 agents et 4 actions, un agent doit évaluer

$$N_{couples} = C^2_{N_{agents}} = \frac{N_{agents}!}{2(N_{agents} - 2)!} = \frac{10!}{2 \cdot 8!} = 45$$

couples d'agents pour un maximum de

$$N_{actions\ jointes} = (N_{actions})^{N_{agents}} = 4^{10} = 1048576$$

actions jointes possibles, et ce lors de chaque mouvement.

3.2 Résultats expérimentaux

Les résultats obtenus en utilisant la méthode JAG peuvent être comparés aux résultats de simulation des scénarios mettant en oeuvre des agents collaboratifs maximisant une fonction d'utilité globale (section 2.3.3). Les graphiques représentent toujours les résultats moyens sur 20 apprentissages. Dans un premier temps, l'attention sera portée sur l'analyse des résultats basés sur le scénario $2a$ de la section 2.3.3 comportant deux agents. Par la suite, il sera montré que dans le cas de plus de deux agents, la méthode JAG présentée dans ce chapitre fournit de meilleurs résultats.

Scénario $2a$

Dans ce scénario, deux agents se trouvent dans un environnement de taille 5×5 comportant des obstacles et doivent atteindre leur goal par le chemin le plus court sans entrer en collision. Ils explorent leur environnement de manière aléatoire. La figure 2.23 illustre ce scénario.

Les figures 3.3 et 3.4 mettent en perspective les performances des différents algorithmes dans l'environnement contenant des obstacles. Comme prévu, dans le cas de deux agents les résulats ne diffèrent pas. Il s'agit d'une situation normale puisque pour deux agents aucune diminution n'est réalisée au niveau de l'espace d'états.

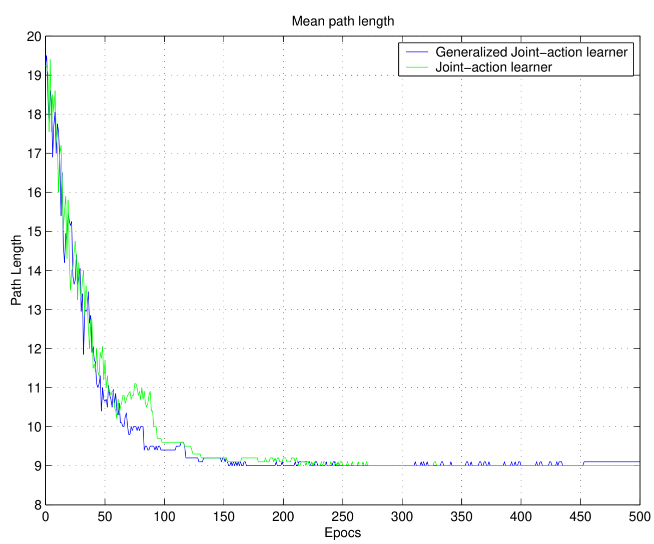


Figure 3.3: Scénario $2a$. Longueur des chemins dans le cas de deux agents dans un environnement exploré aléatoirement et comportant des obstacles. Comparaison avec la courbe de longueur des chemins trouvés par les agents collaboratifs de la section 2.3.3. On observe la même qualité d'apprentissage.

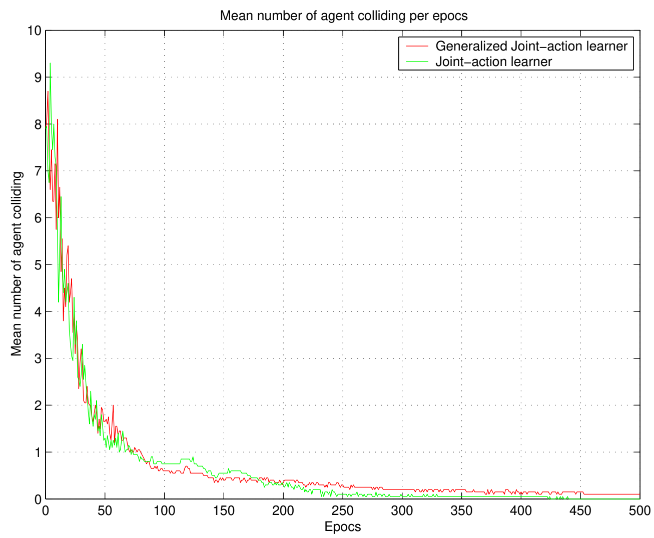


Figure 3.4: Scénario 2a. Nombre de collisions dans le cas de deux agents dans un environnement exploré aléatoirement avec des obstacles. Comparaison avec la courbe des collisions occasionées par les agents collaboratifs de la section 2.3.3. On observe la même qualité d'apprentissage.

Scénario $1c$

Le scénario $1c$ décrit au sein du chapitre précédent par la figure 2.34 comporte quatre agents mais aucun obstacle. Les résultats obtenus avec la généralisation d'une table de deux agents sont illustrés par les figures 3.5 et 3.6. On remarque que la méthode JAG converge plus rapidement vers la longueur de chemin optimale que dans le cas d'une table jointe classique. Le nombre de collisions diminue légèrement plus rapidement dans le cas d'une table jointe de quatre agents. La longueur du chemin ne converge cependant pas en 5000 itérations. Par contre, l'algorithme proposé remplit simultanément les deux critères d'optimalité, et ce en moins de 3000 itérations.

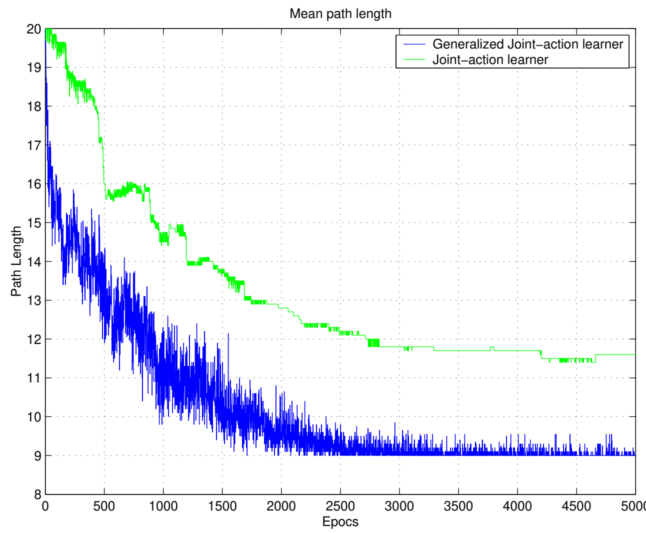


Figure 3.5: Longueur des chemins dans le cas de quatre agents dans un environnement exploré aléatoirement sans obstacle. Comparaison avec la courbe de longueur des chemins trouvés par les agents collaboratifs de la section 2.3.3.

Scénario $2c$

Le scénario $1c$ décrit dans la section précédente par la figure 2.43 comporte quatre agents et des obstacles. Les résultats obtenus avec la généralisation d'une table de deux agents sont illustrés par les figures 3.7 et 3.8. Une nouvelle fois, l'algorithme proposé converge plus rapidement vers la longueur de chemin optimale que dans le cas d'une table jointe classique. De plus, le nombre de collisions converge également plus rapidement vers une valeur proche de zéro, bien qu'il n'atteigne pas cette valeur sur la moyenne des 20 apprentissages. Il suffit en effet qu'un des apprentissages sur les 20 effectués se soit terminé avec une collision pour que la moyenne en soit affectée. L'algorithme classique lui n'a pas convergé après 5000 itérations. En fin d'apprentissage, il existe un écart de 7 déplacements entre les moyennes des longueurs des chemins trouvées avec les deux méthodes, et une différence de 15,5 entre les nombres moyens de collisions. Pour rappel, le nombre de collisions comptabilise la somme des collisions de chaque agent.

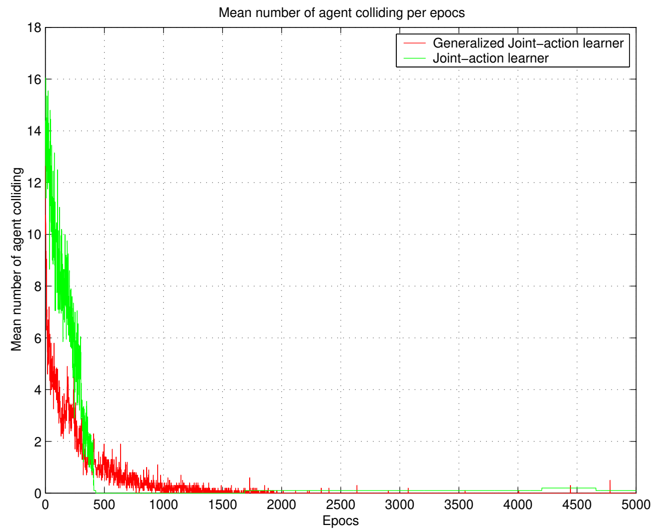


Figure 3.6: Nombre de collisions dans le cas de quatre agents dans un environnement exploré aléatoirement sans obstacle. Comparaison avec la courbe des collisions occasionnées par les agents collaboratifs de la section 2.3.3.

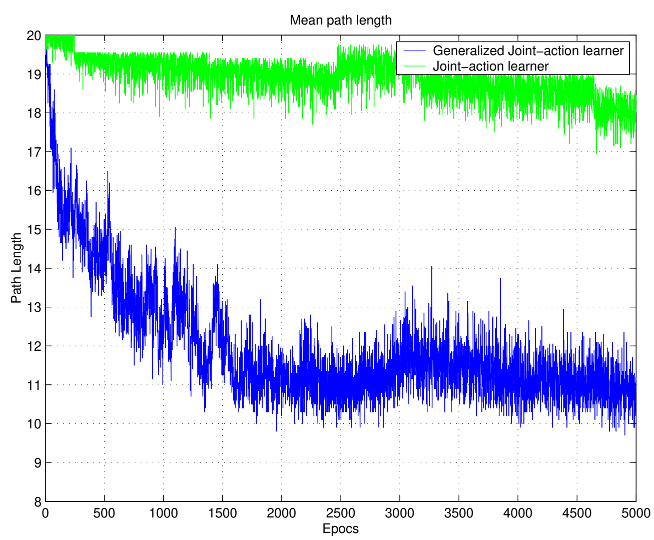


Figure 3.7: Longueur du chemin dans le cas de quatre agents dans un environnement exploré aléatoirement comportant des obstacles. Comparaison avec la courbe de longueur des chemins trouvés par les agents collaboratifs de la section 2.3.3.

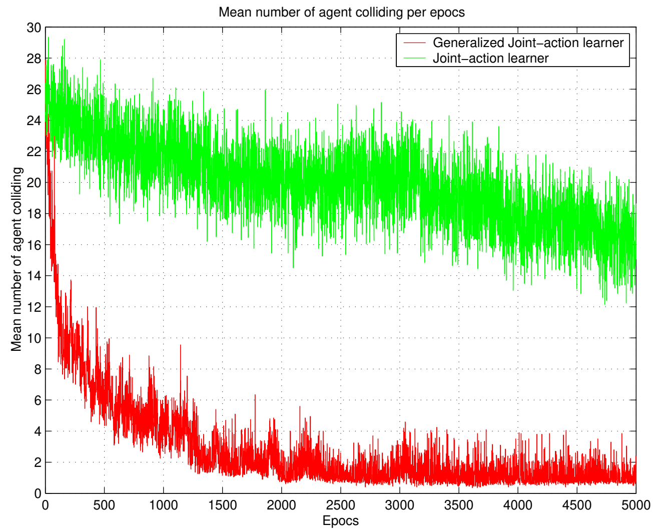


Figure 3.8: Nombre de collisions dans le cas de quatre agents dans un environnement exploré aléatoirement comportant des obstacles. Comparaison avec la courbe des collisions occasionnées par les agents collaboratifs de la section 2.3.3.

3.3 Conclusion

Le présent chapitre a détaillé la méthode JAG basée sur la généralisation d'une table de deux agents. Cette approche apporte une forte diminution de l'espace état-action à parcourir, permettant une convergence plus rapide vers une coordination optimale que dans le cas d'agents collaboratifs. Les résultats expérimentaux effectués avec quatre agents confirment cette hypothèse. Cette approche est donc prometteuse, bien qu'un grand nombre d'améliorations restent possibles. Ces dernières sont l'objet de chapitre suivant.

Chapitre 4

Travail ultérieur

Les chapitres précédents ont introduit le Q learning, présenté une modélisation du problème de coordination et enfin proposé de nouvelles solutions. Il subsiste cependant plusieurs aspects qui n'ont pas été traités par manque de temps.

4.1 Approximation de la fonction Q

L'un des principaux problèmes rencontrés consiste à visiter toutes les paires état-action possibles afin de déterminer l'utilité de tous ces états. Lorsque l'apprentissage est incomplet, il est possible d'interpoler la valeur de la fonction Q à partir des estimations connues $\hat{Q}$.

Ceci peut être réalisé en remplaçant la table $\hat{Q}$ par un Perceptron Multi-Couche (Multi-Layer Perceptron ou MLP), comme décrit en [4]. Il s'agit d'un réseau de neurones artificiels utilisé pour approximer la fonction Q. L'apprentissage est alors supervisé, c'est-à-dire que la généralisation est effectuée à partir de plusieurs couples d'entrées/sorties corrects. Une configuration possible est de créer un MLP pour chaque action. Celui-ci prendrait un état en entrée et l'estimation de la fonction Q en sortie. Pour le réseau de l'action a, les couples $(s, \hat{Q}(s,a))$ proviennent des estimations successives réalisées sur les états visités s au cours d'un épisode, à l'aide de la formule 1.8.

Un avantage est que cette approximation permet d'obtenir une estimation continue de la fonction $\hat{Q}$, ce qui était impossible pour une table $\hat{Q}$.

4.1.1 Exemple d'application

Cette méthode peut s'avérer utile notamment dans le cas où l'apprentissage de la table jointe pour deux agents est effectué pour un certain nombre de goals prédéfinis. Si l'on souhaite ultérieurement que l'agent aille vers un autre objectif, un MLP peut effectuer une interpolation entre les entrées correspondant aux goals appris pour approximer la valeur de la fonction Q du nouveau goal. Par exemple, supposons que l'on s'intéresse aux valeurs d'une action $a =$ Sud. Soit les états s_1 et s_2 comprenant chacun la position $(2,2)$ et les goals respectifs $(5,5)$ et $(1,5)$. Les valeurs $\hat{Q}(s_1,a)$ et $\hat{Q}(s_2,a)$ ont été estimées par le Q learning, il reste à les faire apprendre au MLP. Les entrées fournies au MLP pour ces deux états visités peuvent être les nombres suivants convertis en binaire :

$$input_{(2,2),(5,5)} = 2 + 2 \cdot 5 + 5 \cdot 5^2 + 5 \cdot 5^3 = 762$$
$$input_{(2,2),(1,5)} = 2 + 2 \cdot 5 + 1 \cdot 5^2 + 5 \cdot 5^3 = 662$$

Les sorties déterminées par la règle de mise à jour de la table $\hat{Q}$ lorsque l'action Sud est effectuée dans les états s_1 et s_2 sont

$$output_{Sud,(2,2),(5,5)} = \hat{Q}(s_1, a) = 3.36$$
$$output_{Sud,(2,2),(1,5)} = \hat{Q}(s_2, a) = 8.78$$

Si lors de l'utilisation de la politique apprise, l'état s_3 possède la même position que s_1 mais un goal en $(2, 5)$, on peut utiliser le réseau neuronal pour approximer la valeur de $\hat{Q}(s_3, a)$. L'entrée relative à s_3 est

$$input_{(2,2),(2,5)} = 2 + 2 \cdot 5 + 2 \cdot 5^2 + 5 \cdot 5^3 = 687$$

et la valeur $\hat{Q}$ retournée par le MLP est $6,07$, comme l'illustre la figure 4.1. L'utilité de l'action $a =$ Sud en s_3 est en effet intermédiaire entre celle de s_2 et celle de s_1.

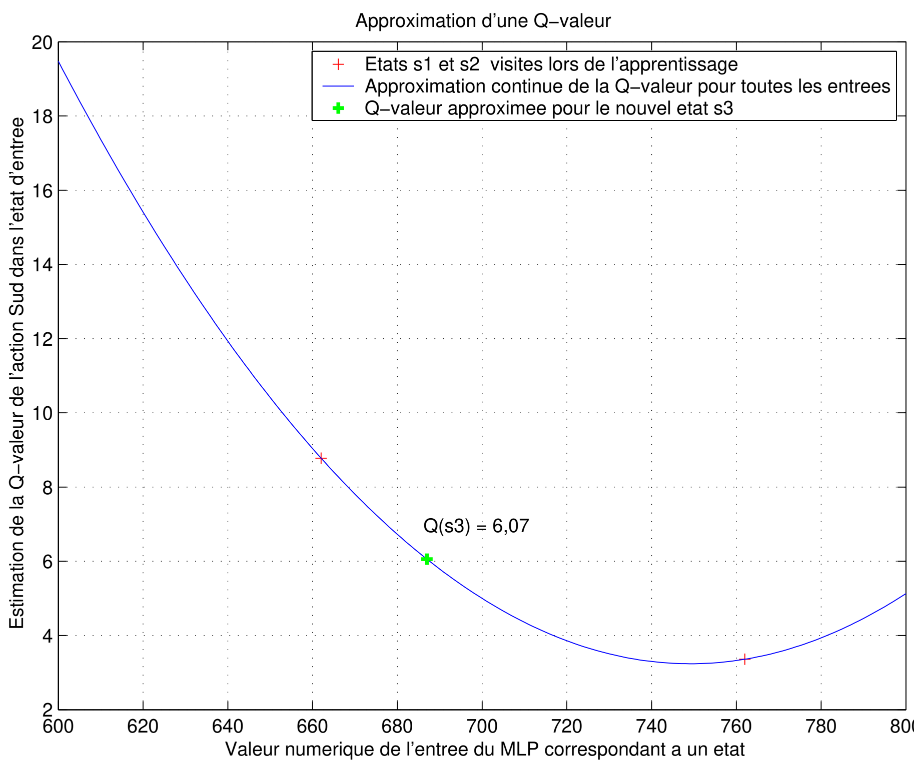


Figure 4.1: Exemple d'approximation d'une fonction Q pour une action $a =$ Sud à partir des valeurs $\hat{Q}(s_1, a)$ et $\hat{Q}(s_2, a)$. On obtient $\hat{Q}(s_3, a) = 6,07$.

4.1.2 Limitations

Le premier inconvénient de cette méthode est le temps de calcul nécessaire à la rétro-propagation permettant de modifier les poids du réseau de façon à réduire l'erreur d'approximation. Ceci est réalisé pour chaque état visité et peut ralentir considérablement l'apprentissage.

Le second inconvénient est qu'il est nécessaire que les entrées/sorties utilisées soient de bonne qualité et en quantité suffisante avant que le MLP puisse converger vers une approximation correcte de la fonction $\hat{Q}$. Cela peut notamment poser problème si l'approche probabiliste décrite à la section 1.3.7 est utilisée, les valeurs initiales de l'estimation $\hat{Q}$ risquant d'être erronées en début d'apprentissage, et ce même pour les états déjà parcourus. On peut pallier à ce problème en interrogeant préférentiellement une table $\hat{Q}$ qui sera correcte pour les pairesétat-action visitées.

Enfin, pour utiliser un MLP il est nécessaire de ramener les composantes des états d'entrée à des nombres réels. Il est souhaitable que l'ordre total entre les nombres ait une signification concrète. Par exemple, si un agent est caractérisé par un mode *fuite*, *attaque* ou *patrouille*, ces états peuvent respectivement être ramenés aux nombres 1, 2 et 3. Or le fait que le mode représenté par le nombre 1 est plus éloigné de 3 que de 2 n'a pas de signification réelle et peut donc nuire à l'apprentissage.

4.2 Configuration automatique des paramètres

L'algorithme du Q learning possède plusieurs paramètres qui doivent être ajustés correctement pour obtenir un bon apprentissage. On peut citer les valeurs des récompenses obtenues lorsqu'un agent arrive à son goal ou quand il entre en collision, le facteur γ de la règle de mise à jour 1.8 ou encore la température de Boltzmann T. Il serait donc utile de mettre au point une méthode permettant la configuration automatique de ces paramètres afin d'optimiser l'apprentissage.

4.2.1 Learning Momentum

La méthode du Learning Momentum (LM) décrite dans l'article [2] pourrait remplir cette tâche, si du moins elle était appliquée aux paramètres de l'algorithme Q learning et non pas à des agents. Cette approche nécessite le stockage d'une table contenant les modifications à imposer aux paramètres en fonction des performances actuelles. Dans le cadre du présent problème, les critères de performance peuvent être la longueur du meilleur parcours trouvé et le nombre de collisions.

Concrètement, les paramètres de l'algorithme sont modifiés si un épisode de l'apprentissage génère une politique moins bonne que lors des épisodes antérieurs. Si, par exemple, un épisode produit davantage de collisions que précédemment, l'algorithme du LM se chargerait d'augmenter la pénalité obtenue lors des collisions. Si, par contre, les collisions sont inexistantes mais que la longueur du chemin obtenue ne diminue pas, l'influence du goal serait accrue en augmentant la récompense qui lui est associée.

4.2.2 Algorithme génétique

Une autre alternative est l'utilisation d'un algorithme génétique pour trouver la meilleure combinaison de paramètres. Un tel algorithme suit le schéma suivant, illustré par la figure 4.2 :

1. Formation d'une population de combinaisons de paramètres possibles, sous forme de chaînes de caractères. Si la récompense liée au goal vaut 20.0, celle relative aux collisions vaut -10.0, $\gamma = 0.9$ et $T = 1.5$, un exemple de chaîne serait le nombre 20100915 encodé sous forme binaire ;

2. Sélection des parents parmi la population. Il s'agit des chaînes de la population correspondant aux paramètres les plus performants ;

3. Recombinaison des parents deux par deux choisis aléatoirement, de façon à former la génération future. Un exemple de recombinaison consiste à prendre la moitié gauche de la chaîne du premier parent concaténée avec la moitié droite de celle du second parent ;

4. Mutation aléatoire d'un faible pourcentage de la nouvelle génération, afin de maintenir une certaine diversité au sein de la population. Un exemple illustrant cette mutation serait le remplacement d'un chiffre de la chaîne par un nombre pris au hasard entre 0 et 9 ;

5. Sélection des survivants parmi l'ancienne population et la nouvelle génération de manière à former une nouvelle population de taille identique à la précédente. Les chaînes correspondant aux paramètres produisant les algorithmes les plus performants sont sélectionnées pour constituer la nouvelle population. Si la qualité du meilleur algorithme trouvé n'est pas satisfaisante, on recommence à partir de l'étape 2.

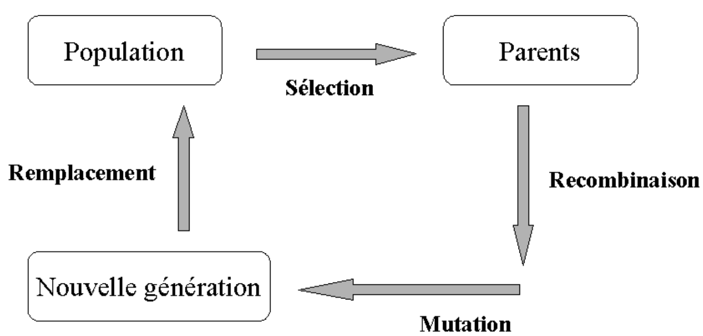


Figure 4.2: Différentes étapes d'un algorithme génétique.

Le principal inconvénient posé par cette méthode est le temps nécessaire à l'évaluation de la qualité de chaque apprentissage. Lors de chaque itération, il faut en effet effectuer un apprentissage pour tous les membres de la population afin d'en sélectionner les plus efficaces.

4.3 Apprentissage de comportements primitifs

Un autre aspect qui n'a pas encore été traité est l'apprentissage de sous-comportements (*behaviors*) de base, dans le but de former des comportements plus complexes [7, 6].

Dans le cadre du problème initial présenté dans la section 2.1.1, on peut par exemple définir comme comportement primitif le fait que plusieurs avions passent de leur position initiale à leur goal en se coordonnant. L'apprentissage de ce comportement est le problème traité dans ce mémoire. Une fois cette tâche apprise, on peut définir des formations d'avions (avions alignés, en V, de front, etc.) et apprendre quelles formations permettent d'assurer un trajet optimal du point de départ de l'escadrille à son point d'arrivée. Les changements successifs sont donc gérés par la politique apprise qui assure la coordination entre les formations. Les états seraient représentés par la position globale de l'escadrille sur une carte ainsi que la formation actuelle, et les actions seraient les mouvements globaux de l'escadrille définis par les directions cardinales. Un autre algorithme de coordination qui gèrerait le mouvement global de l'escadrille pourrait être envisagé.

La généralisation d'une table de deux agents proposée au chapitre 3 peut également être vue comme une division en sous-comportements. La coordination entre deux agents constitue dès lors un comportement primitif. Cette capacité est ensuite utilisée pour la coordination globale de plusieurs agents. Le Q learning n'intervient alors que pour l'apprentissage du comportement de base et non pour gérer la juxtaposition de ces comportements pour plus de deux agents.

4.4 Raffinement de graphe

La section 2.1.3 abordait le concept de graphe dans le but de remplacer la grille bidimensionelle initialement proposée. Ces graphes permettent de modéliser des environnements plus complexes, tout en limitant implicitement le choix des actions possibles pour chaque état. L'ajout de règles définissant les actions permises en fonction de l'état n'est pas nécessaire, car ces actions sont représentées par les arêtes du graphe.

Une amélioration possible liée à la modélisation de l'environnement sous forme de graphe est le raffinement successif en fonction du degré de précision souhaité. A la place du graphe complet décrivant l'environnement, l'agent ne possède au départ qu'une représentation simplifiée du monde qui l'entoure. Il s'agit d'un graphe dans lequel chaque noeud correspond à un ensemble de noeuds du graphe complet. Les figures 4.3 et 4.4 illustrent un exemple de simplification.

Lors de l'apprentissage, l'agent va estimer la fonction Q pour les états du graphe simplifié, réduisant ainsi considérablement le nombre de paires état-action à visiter. Lorsqu'un agent rencontre un obstacle, il va raffiner son graphe en remplaçant le noeud auquel il se trouve par un sous-graphe plus détaillé de l'environnement. Ce nouveau graphe pourra être raffiné davantage ultérieurement, lorsque l'agent aura une connaissance plus précise des emplacements des obstacles. La précision du raffinement ne peut cependant excéder le niveau de détail du graphe complet de l'environnement.

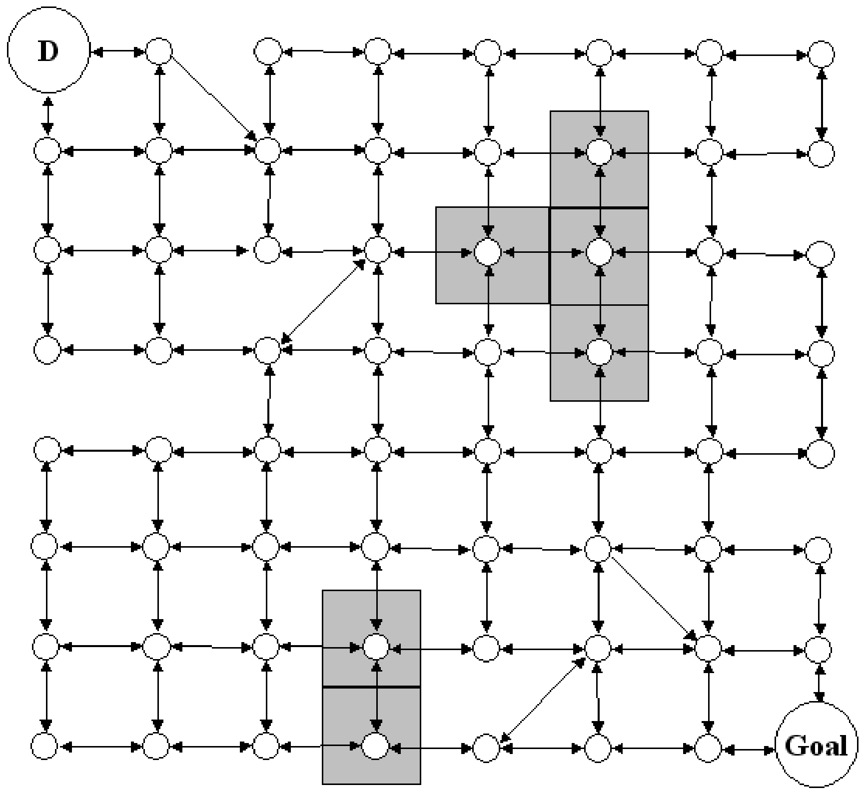


Figure 4.3: Graphe complet d'un environnement, le noeud D est la position initiale de l'agent.

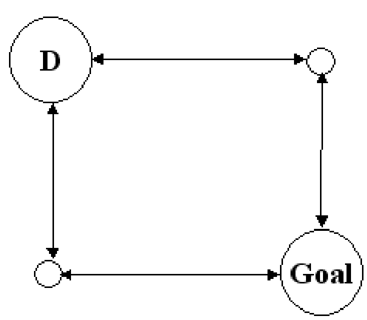


Figure 4.4: Graphe simplifié initial comprenant quatre noeuds.

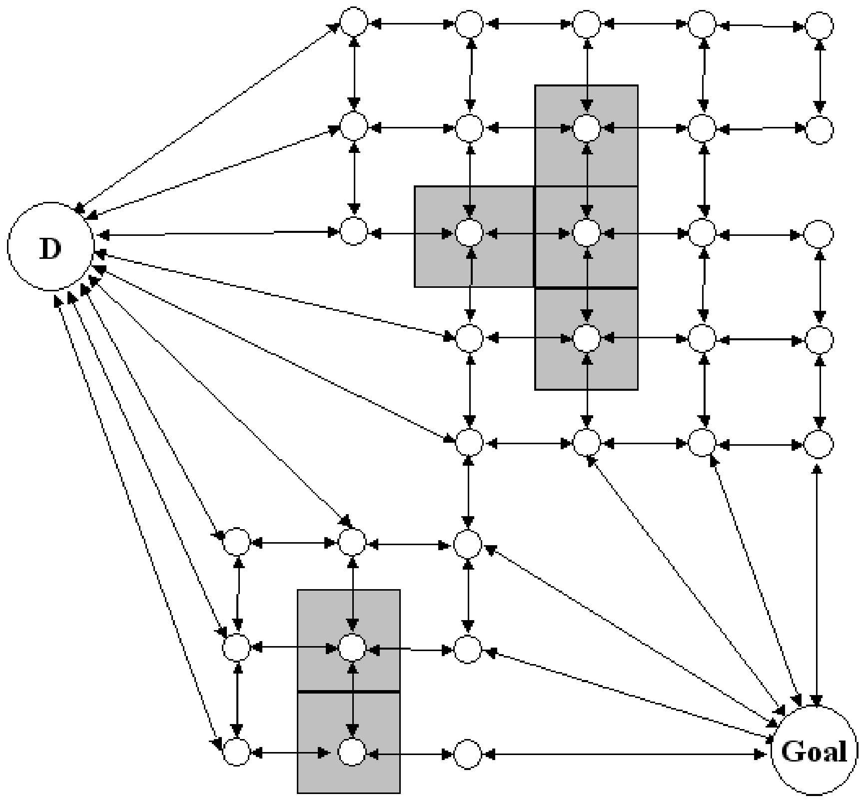


Figure 4.5: Graphe final de l'agent après apprentissage, dans lequel l'environnement proche des obstacles est raffiné de façon à ne stocker que les états importants dans la table $\hat{Q}$.

Ainsi, l'agent va obtenir une estimation de la fonction Q pour des états précis aux endroits où cela est nécessaire et des états plus généraux lorsqu'il n'y a pas de variation dans l'environnement. Le nombre d'états à visiter sera par conséquent réduit. La figure 4.5 montre un exemple de raffinement final en comparaison du graphe de la figure 4.3.

Le principal inconvénient de cette méthode est qu'elle n'est pas applicable dans le cas d'un environnement dynamique. L'intérêt du raffinement est en effet qu'il n'est effectué que près des obstacles. Si ceux-ci changent d'emplacement, le raffinement devient inutile. En outre, l'apprentissage d'un agent avec des obstacles aléatoires produirait un graphe final complètement raffiné.

Un autre problème réside dans la prise en compte des collisions entre agents. Celles-ci peuvent en effet se produirent au sein de n'importe quel noeud du graphe, rendant ainsi inutile le raffinement des noeuds où elles ont lieu. Cette solution est donc uniquement intéressante dans le cas d'un agent seul évoluant dans un environnement statique.

4.5 Apprentissage de sous-goals

Si la politique à apprendre est indépendante du goal, elle peut être utilisée par des agents possédant des goals différents de ceux de l'apprentissage. De ce fait, une amélioration possible pour un chemin de longueur totale n est de considérer successivement chaque état emprunté s_i $(1 \leq i \leq n)$ comme un goal. La table $\hat{Q}$ doit alors être mise à jour pour chacun des sous-épisodes partant de la position initiale jusqu'à l'état intermédiaire s_i, tout en donnant une récompense appropriée dans ce dernier état. La figure 4.6 illustre un exemple d'apprentissage basé sur des sous-goals. Pour un chemin de longueur totale n on a donc n chemins de longueurs croissantes, ce qui correspond à

$$\sum_{j=1}^{n} j = \frac{n(n+1)}{2}$$

déplacements pouvant modifier la table $\hat{Q}$.

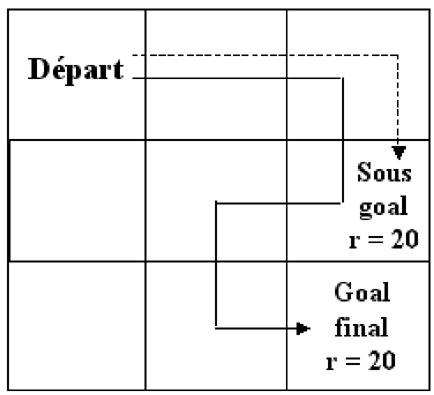


Figure 4.6: Exemple de sous-épisode allant jusqu'à la position $(3,2)$ (en pointillés), l'épisode complet allant de $(1,1)$ à $(3,3)$ est représenté en trait plein. Une même récompense de 20 est donnée pour tous les sous-goals.

Cette méthode présente l'avantage qu'un seul épisode engendre nettement plus d'ajouts dans la table $\hat{Q}$ que l'algorithme traditionnel. Ceci est très utile lorsque le coût de réalisation d'un épisode est élevé comparé à celui de l'exécution de l'algorithme Q learning. C'est le cas, par exemple, de robots apprenant à se mouvoir en temps réel.

Cette méthode ne s'avère cependant intéressante que si l'on ne connaît pas les goals avec lesquels la politique apprise sera utilisée. Dans le cas contraire, il est préférable de limiter l'apprentissage aux goals prédéfinis. La représentation d'état est importante dans le cadre de cette amélioration. En effet, si elle inclut la position du goal, le nombre d'itérations nécessaires restera identique mais le temps d'apprentissage sera multiplié par le nombre moyen de sous-épisodes par rapport à l'apprentissage d'un goal fixé.

4.6 Initialisation basée sur un algorithme du plus court chemin

Au cours de ce travail, les valeurs de la table $\hat{Q}$ ont toujours été initialisées à zéro. Cela revient à commencer l'apprentissage avec des agents dépourvus de connaissance. Une amélioration substancielle serait d'utiliser un algorithme tel que A^* trouvant le plus court chemin. Cet algorithme requiert cependant la connaissance de la distance au goal pour chaque position, ce qui constitue une hypothèse très restrictive en comparaison de celles proposées dans ce travail. Une fois le parcours le plus court déterminé, il faut alors incrémenter les valeurs de la table $\hat{Q}$ correspondant aux paires état-action de ce chemin.

Bien que la coordination ne soit pas garantie, les agents commenceront avec une bonne connaissance de l'environnement. Ils pourront ainsi concentrer leur apprentissage sur la diminution du nombre de collisions et non sur la recherche du chemin le plus court. L'utilisation de la politique d'exploration probabiliste est souhaitable afin de profiter pleinement des informations mises à la disposition des agents.

4.7 Propagation des modifications de la table $\hat{Q}$

L'algorithme déterministe du Q learning décrit dans la table 1.1 met à jour les valeurs de la table $\hat{Q}$ selon la formule

$$\hat{Q}(s, a) \leftarrow r(s, a) + \gamma \max_{a'} \hat{Q}(s', a')$$

où $r(s, a)$ est la récompense immédiate reçue en effectuant l'action a dans l'état s, et le second terme représente la valeur de la meilleure action possible dans l'état résultant s' pondérée par un facteur γ.

Une amélioration à l'algorithme du Q learning serait d'appliquer cette règle de mise à jour non pas aux seuls couples état-action de l'épisode en cours, mais à toutes les paires (s, a) possibles. La mise à jour d'un épisode entier peut en effet également modifier les paires état-action dont l'état résultant a été parcouru par un agent. Si la nouvelle valeur d'un couple (s, a) de l'épisode est très grande, les paires état-action (s_{prec}, a) $\forall$ $a \in A$ dont l'état résultant est s seront fortement influencées par cette modification. De même, la valeur des paires $(s_{prec'}, a)$ dont l'état résultant est s_{prec} peut également être altérée. Cette propagation peut continuer tant que la valeur de la meilleure action à effectuer dans l'état résultant est différente de celle de l'épisode précédent. La figure 4.7 illustre un exemple de propagation des valeurs de $\hat{Q}$. On peut noter que dans le cas de l'algorithme non-déterministe dont la mise à jour est définie par la formule 1.13, les valeurs de toutes les paires état-action possibles seraient modifiées lors de chaque épisode.

L'avantage présenté par cette méthode est qu'elle permet d'accélérer la convergence de l'algorithme, l'influence de chaque épisode étant propagée à tous les couples état-action possibles. Leur valeur sera donc plus proche de leur utilité réelle que dans l'algorithme de base.

Le principal inconvénient de cette approche est qu'elle nécessite une analyse de toutes les paires état-action possibles lors de chaque épisode. Ce nombre peut être très grand, comme le montre la formule 2.5, puisque dans le cas d'une table jointe il augmente exponentiellement avec le nombre d'agents. Il faut alors un temps considérable pour mettre à jour tous les couples possibles lors de chaque épisode d'apprentissage. Cette technique nécessite en outre une structure de données permettant de déterminer quels sont les états pouvant mener à un état résultant s de l'épisode.

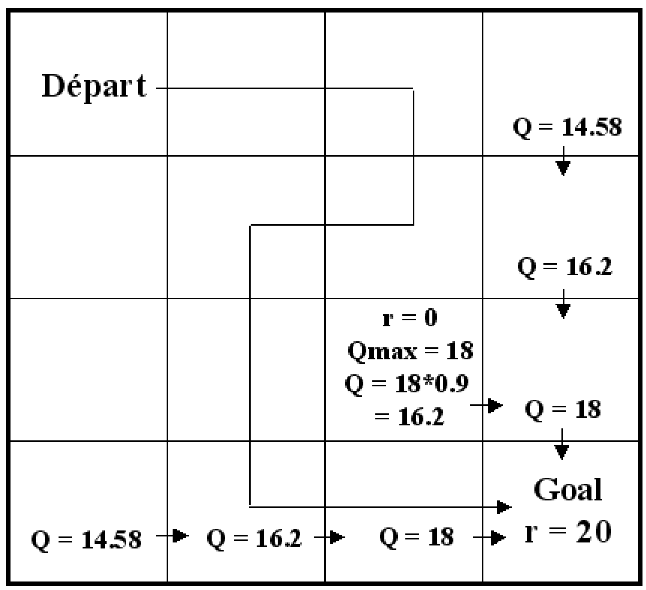


Figure 4.7: Exemple de propagation partielle des nouvelles valeurs de $\hat{Q}$ d'un épisode d'apprentissage.

Conclusion

L'objectif visé par ce travail est d'analyser différentes méthodes d'apprentissage de la coordination dans un système multi-agents. Se basant sur des concepts existants, une analyse des performances d'agents indépendants et d'agents collaboratifs partageant une même table $\hat{Q}$ a été menée. L'influence de la politique d'exploration sur l'apprentissage a également été étudiée, comparant l'exploration aléatoire à l'approche probabiliste. A cet effet, un programme permettant de tester les différents algorithmes d'apprentissage a été implémenté.

Les agents indépendants apprennent rapidement car ils ont chacun un nombre restreint d'états à visiter par rapport aux agents collaboratifs. Il leur est cependant nécessaire d'inclure leur environnement proche dans leur état afin de pouvoir éviter les collisions directes. En outre, l'algorithme Q learning déterministe ne parvient pas à gérer les collisions d'agents séparés par une case. Le Q learning non-déterministe améliore l'apprentissage tout en ne permettant pas d'assurer un risque de collision nul, puisque la probabilité d'obtenir une collision est identique pour beaucoup de paires état-action. L'algorithme ne parvient donc pas à différencier la valeur de ces actions. On ne peut par conséquent pas garantir la coordination optimale d'agents indépendants.

En ce qui concerne les agents collaboratifs élaborés dans [3], l'état global comprend la position de tous les agents. Ils parviennent donc à apprendre à se coordonner correctement en maximisant la valeur de leurs actions jointes. Le problème majeur rencontré en utilisant cette méthode est que la dimension de l'espace d'état-action à visiter croît exponentiellement avec le nombre d'agents. Ceci rend cette technique difficilement applicable à des problèmes réalistes.

Dans le but de remédier à cet inconvénient, une nouvelle méthode nommée JAG (Joint Action Generalization) a été proposée. Elle consiste à généraliser une table jointe de deux agents à un nombre quelconque d'agents. Ceci réduit grandement le nombre d'états à visiter pour une qualité d'apprentissage identique à celle d'agents collaboratifs. La méthode JAG converge ainsi plus rapidement vers une coordination optimale. Les résultats expérimentaux avec 4 agents confirment ces espérances. La figure 4.8 compare la moyenne sur 20 apprentissages des longueurs des chemins obtenues avec les trois techniques analysées. La figure 4.9 illustre le nombre moyen de collisions. On constate que la méthode JAG est la seule à converger vers une coordination optimale en moins de 3000 épisodes.

L'un des objectifs visés par ce mémoire est d'obtenir un comportement capable de faire face à des situations imprévues. Pour ce faire, l'apprentissage des agents collaboratifs a été réalisé dans un environnement hautement dynamique, tout en utilisant la politique obtenue dans un environnement modifié. Les résultats sont concluants dans le cas de de petites pertubations de l'environnement. Le Q learning ne permet toutefois pas aux agents

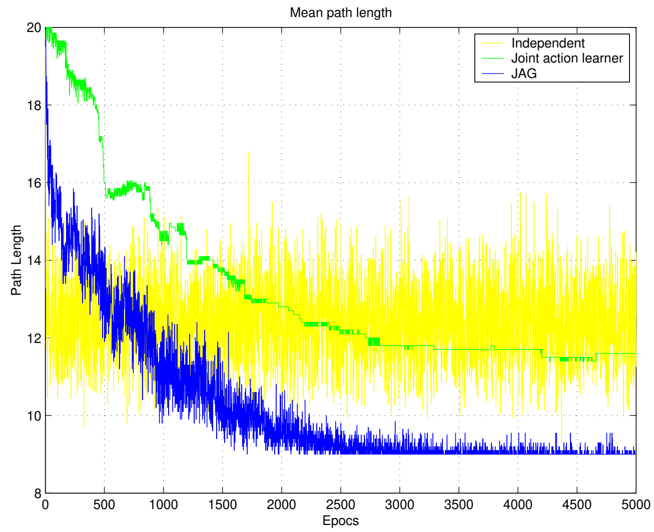


Figure 4.8: Comparaison des longueurs des chemins obtenues avec des agents indépendants, collaboratifs et la méthode JAG. La méthode JAG converge vers le chemin le plus court bien plus rapidement que les agents collaboratifs, en moins de 3000 épisodes.

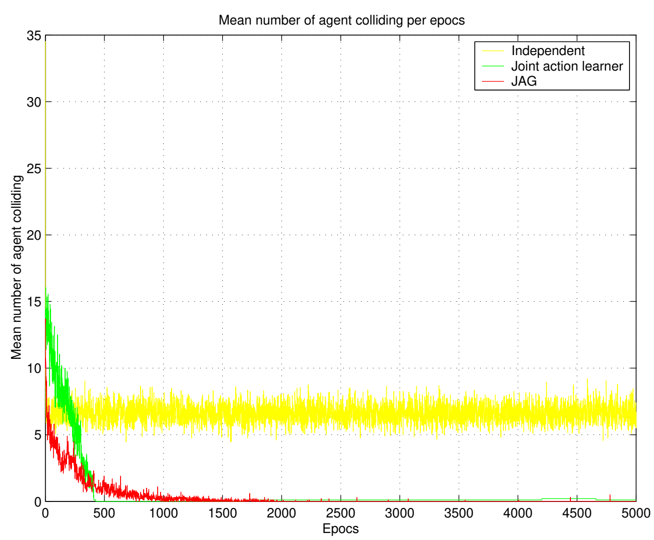


Figure 4.9: Comparaison du nombre moyen de collisions obtenu avec des agents indépendants, collaboratifs et la méthode JAG. Les agents utilisant la méthode JAG évitent plus rapidement toutes les collisions que les agents collaboratifs, en moins de 500 épisodes.

de réagir correctement lorsqu'ils sont placés dans un environnement totalement différent. Ceci est la conséquence d'une restriction inhérente aux Processus de Décision Markoviens : les agents ne prennent pas en compte leurs états précédents lors de leurs décisions. Ils peuvent néanmoins adapter leur coordination dans un environnement évoluant en cours d'apprentissage.

Dans le but d'améliorer la méthode JAG, un grand nombre d'idées de travail ultérieur ont été proposées, prouvant qu'il reste beaucoup à faire dans le domaine de l'apprentissage de la coordination d'agents. Ces différentes idées se veulent une porte ouverte sur une étude plus approfondie qui utiliserait ce mémoire comme base de travail.

Références

[1] Ronald C. Arkin, Yoichiro Endo, J. Brian Lee, Douglas C. MacKenzie, and Eric Martinson. Multistrategy learning methods for multirobot systems. In *Proceedings of the 2nd International Workshop on Multi-Robot Systems*, pages 137–150, Washington, D.C., USA, 2003.

[2] Ronald C. Arkin and J. Brian Lee. Adaptive multi-robot behavior via learning momentum. In *Proceedings of the 2003 IEEE International Conference on Robots and Systems (IROS'03)*, Las Vegas, NV, USA, 2003.

[3] Craig Boutilier. Planning, learning and coordination in multiagent decision processes. In *Theoretical Aspects of Rationality and Knowledge*, pages 195–201, 1996.

[4] Mark Humphrys. *Action Selection methods using Reinforcement Learning*. PhD thesis, Trinity Hall, University of Cambridge, 1997.

[5] Spiros Kapetanakis, Daniel Kudenko, and Malcolm J.A. Strens. Reinforcement learning approaches to coordination in cooperative multi-agent systems. In *Adaptive Agents and Multi-Agent Systems*, volume 2636 of *Lecture Notes in Computer Science*, pages 18–32. Springer, 2003.

[6] Eric Martinson and Ronald C. Arkin. Learning to role-switch in multi-robot systems. In *Proceedings 2003 of the IEEE International Conference on Robotics and Automation*, Taipei, Taiwan, 2003.

[7] Eric Martinson, Alexander Stoytchev, and Ronald C. Arkin. Robot behavioral selection using Q-learning. In *Proceedings of the 2002 IEEE International Conference on Robots and Systems (IROS'02)*, Lausanne, Switzerland, 2002.

[8] Tom Mitchell. *Machine Learning*. McGraw Hill, 1997.

[9] Enric Plaza and Santiago Ontañón. Cooperative multiagent learning. In *Adaptive Agents and Multi-Agent Systems*, volume 2636 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2002.

[10] Peter Van Roy and Seif Haridi. *Concepts, Techniques, and Models of Computer Programming*. MIT Press, 2004.

[11] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2002.

[12] Sandip Sen and Mahendra Sekaran. Individual learning of coordination knowledge. *Journal of Experimental & Theoretical Artificial Intelligence*, 10:333–356, 1998.

[13] Yoav Shoham. Agent oriented programming. *Artificial Intelligence*, 60:51–92, 1993.

[14] Richard S. Sutton. Integrated architectures for learning, planning and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, Austin, Texas, USA, 1990.

[15] Ming Tan. Multi-agent reinforcement learning: independent vs cooperative agents. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 330–337, Amherst, MA, USA, 1993.

[16] Christopher J.C.H. Watkins. *Learning from delayed rewards*. PhD thesis, King's College, University of Cambridge, 1989.

[17] Christopher J.C.H. Watkins and Peter Dayan. Technical note: Q-learning. *Machine Learning*, 8:279–292, 1992.

[18] Gerhard Weiss. Adaptation and learning: Some remarks and a bibliograhy. *Lecture notes in artificial intelligence*, 1042, 1996.

[19] Gerhard Weiss, editor. *Multiagent systems : a modern approach to distributed artificial intelligence*. MIT Press, Cambridge, Mass., 1999.

[20] Michael Wooldridge. *An Introduction to Multi-Agent Systems*. John Wiley & Sons, Inc. New York, NY, USA, 2001.

Annexe A

Concepts de programmation utilisés

L'une des caractéristiques de notre travail est que le programme permettant d'évaluer les performances des différents algorithmes a été implémenté en *Oz* à l'aide de la plate-forme *Mozart 1.3.0*. Oz est un langage multi-paradigmes permettant d'utiliser à la fois la programmation déclarative, orientée-objet, concurrente ainsi que la programmation par contraintes.

Pour notre programme, nous avons principalement employé le modèle concurrent avec échange de messages. Nous nous sommes cependant limités au modèle déclaratif dans les fonctions ne nécessitant pas de modèle plus expressif. Nous présentons brièvement les divers concepts tirés de [10] que nous avons utilisés.

Active object et Dataflow behavior

Un port est un tunnel de communication entre deux entités permettant de transmettre des messages de manière asynchrone. La lecture du flux de données contenue dans le port se fait de manière séquentielle.

Un *Active Object* est une combinaison de un à plusieurs ports et d'un flux de données. Cette association étend les capacités du flux de données en permettant notamment une communication *many-to-one*.

Les agents et le simulateur sont modélisés par des *Active Object*. L'interface d'un agent est donc un port sur lequel des messages représentant les ordres sont envoyés. L'unique interface du simulateur est également un port sur lequel des requêtes sont envoyées. Ces ports sont donc utilisés pour la communication entre les agents et le simulateur. Les agents envoient leurs requêtes sur l'interface du simulateur, ces messages sont reçus et partionnés, selon leur délai d'arrivée, en des ensembles de requêtes *pseudo-simultanées*. Chaque ensemble de requêtes est ensuite traité par le simulateur et les réponses sont renvoyées via les mêmes messages reçus. On utilise donc le *dataflow behavior* pour synchroniser les requêtes des agents et les réponses correspondantes du simulateur.

High order programming

Il s'agit d'un ensemble de techniques de programmation applicables lorsque l'on utilise des valeurs procédurales et qui permettent notamment de réaliser les opérations suivantes:

- Abstraction procédurale
- Généricité
- Instanciation
- Encapsulation

Nous avons utilisé le high order programming afin de rendre les agents et le simulateur génériques. Par exemple, les différents types d'exploration de l'environnement par les agents sont implémentés par des fonctions interchangeables. Ainsi, pour explorer l'environnement aléatoirement ou de manière probabiliste, seul le paramètre représentant la fonction d'exploration doit être adapté. De même, l'attribution des récompenses est représentée par une fonction, ainsi que toutes les opérations susceptibles d'être modifiées par la suite.

Thread

Une *thread* est une séquence d'instructions indépendantes du reste du programme. Nous utilisons les threads pour simuler l'exécution simultanée des étapes de l'algorithme Q learning par les différents agents. Les actions de chaque agent sont donc exécutées dans des thread séparés. Associées à un port, elle permettent à plusieurs objets indépendants de communiquer de manière asynchrone.

Annexe B

Manuel de l'utilisateur

Ce manuel décrit le fonctionnement de notre programme de simulation. Celui-ci permet à l'utilisateur d'évaluer les performances de l'algorithme du Q learning dans plusieurs scénarios dont le but est d'obtenir une coordination efficace entre agents homogènes. Ces agents se trouvent dans un même environnement et apprennent à se coordonner en utilisant différentes techniques basées sur le Q learning. Ces techniques ont été largement présentées et discutées dans ce mémoire et peuvent être testées avec différentes combinaisons de paramètres d'apprentissage en utilisant le programme se trouvant sur le CD-ROM joint à notre travail.

Le programme constitue donc un outil permettant de faire apprendre à des agents à se coordonner dans un environnement en 2D comportant des obstacles. Cet environnement peut être une grille, les agents et les obstacles sont alors représentés par des cases de cette grille. L'environnement peut également être un graphe, les agents et les obstacles sont alors des noeuds de ce graphe. Le programme permet d'effectuer un apprentissage avec les paramètres choisis dans le panneau de configuration, pour ensuite analyser la politique obtenue à l'aide du panneau de simulation. En cours d'apprentissage, un panneau de statistiques informe en temps réel l'utilisateur de l'avancement de l'algorithme et de la qualité de la politique trouvée.

B.1 Installation

Le programme requiert l'installation au préalable de *Mozart 1.3.0* qui peut être téléchargé sur le site http://www.mozart-oz.org. L'installation du programme sous Linux à partir du CD-ROM peut s'effectuer en suivant les étapes suivantes:

- Copiez l'entièreté du répertoire *program* sur le disque dur.
- Dans le repertoire *multi-agent-learning*, exécutez la commande **make all** pour compiler tous les fichiers. Entrez ensuite **make exec** afin de compiler le fichier exécutable.
- Dans un environnement *X-Windows*, entrez la commande

  testor --env=<EnvType> --atype=<AgentType> pour lancer le programme. Les options des paramètres sont les suivantes:

  – <**AgentType**> peut prendre les valeurs **ind**, **joint** ou **jag**. Ces valeurs correspondent respectivement à des agents indépendants, à des agents collaborants et à des agents collaborants utilisant la nouvelle méthodologie présentée en section 3 pour la coordination. La valeur par défaut est **joint**

– <**EnvType**> peut prendre les valeurs **grid** ou **graph** qui correspondent respectivement à un environnement modélisé par une grille ou un graphe. La valeur par défaut est **grid**

Exemple: *testor –env=grid –atype=joint* lance le programme avec un environnement en forme de grille 2D et avec des agents collaboratifs.

B.2 Utilisation de l'interface

L'interface graphique de notre programme comporte trois composants principaux: un panneau de configuration, un panneau de simulation permettant de visualiser les résultats obtenus et enfin un panneau de statistiques décrivant le déroulement de l'apprentissage.

B.2.1 Panneau de configuration

Lorsque le programme est lancé, une fenêtre semblable à celle de la figure B.1 apparait. Il s'agit du panneau de configuration principal des paramètres de l'apprentissage.

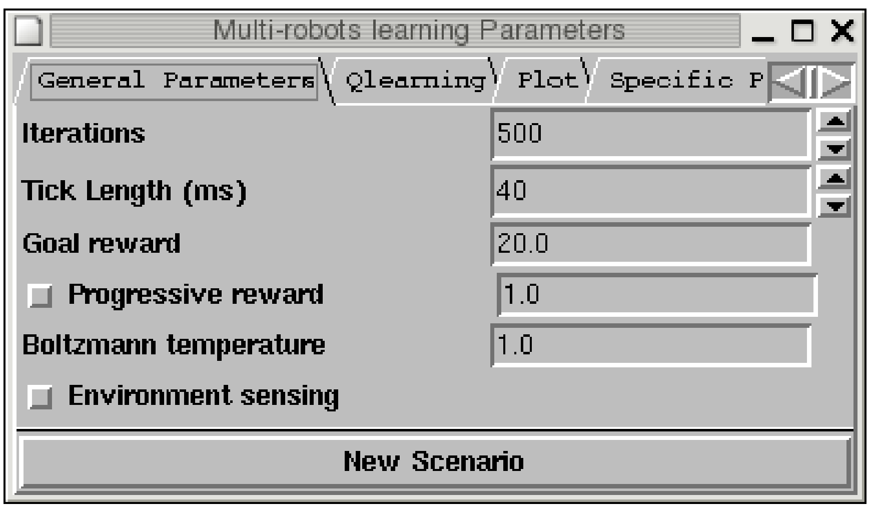


Figure B.1: Panneau de configuration permettant l'ajustement des paramètres généraux de l'apprentissage.

Le premier onglet *General Parameters* permet de définir les paramètres généraux suivants:

Iterations: nombre d'épisodes générés lors de l'apprentissage. Un grand nombre d'itérations permet d'assurer une politique optimale mais cela peut engendrer un long temps d'exécution.

Tick Length: nombre de millisecondes définissant l'intervalle de temps durant lequel le simulateur considère les messages reçus comme simultanés. La synchronisation des agents est décrite plus amplement à la section 2.2.2. L'intervalle de temps minimal est dépendant de Mozart, généralement il est limité à 10 ms, même si la valeur entrée est inférieure. Un grand intervalle de temps peut ralentir considérablement l'apprentissage.

Goal Reward: dans le cas d'agents indépendants, il s'agit de la récompense que chaque agent reçoit quand il atteint son goal. Dans le cas d'agents collaborants il s'agit de la récompense que reçoivent tous les agents terminant leur apprentissage en derniers.

Progressive Reward: si ce paramètre est activé chaque agent reçoit une récompense lorsqu'il se rapproche de son goal et une pénalité quand il s'en écarte. L'utilisateur doit définir la valeur de cette récompense. Il s'agit d'un estimateur de progrès dont l'intérêt est décrit dans l'article de la section 1.2.2. Ce paramètre prend uniquement effet dans le cas d'en environnement en grille.

Boltzmann Temperature: paramètre définissant la valeur initiale de la température dans le cas de l'utilisation d'une politique d'exploration probabiliste. L'influence de ce paramètre est décrite à la section 1.3.7.

Environment Sensing: paramètre activant les capteurs des agents. S'ils sont utilisés, l'état d'un agent comprend la description du contenu des positions adjacentes à la sienne. Ceci permet d'améliorer la qualité de l'apprentissage dans le cas d'agents indépendants.

New Scenario: affiche un nouveau panneau de simulation configuré avec les paramètres entrés dans le panneau de configuration.

L'onglet *Q learning* permet à l'utilisateur de définir les paramètres relatifs à l'algorithme du Q learning présenté à la section 1.3. La figure B.2 illustre cette interface. Les paramètres sont les suivants:

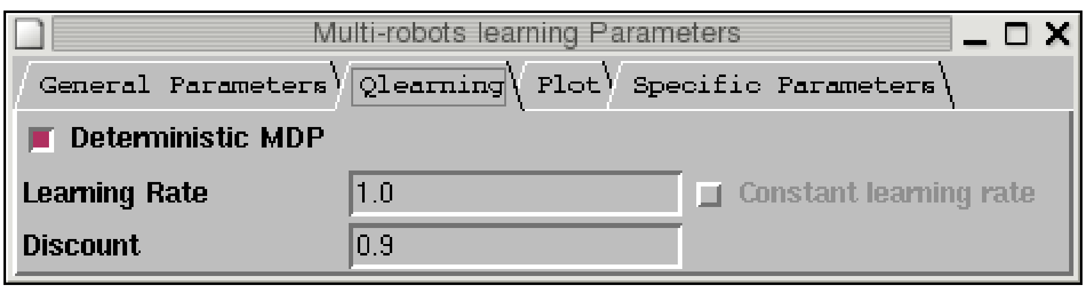


Figure B.2: Panneau de configuration permettant l'ajustement des paramètres relatifs à l'algorithme du Q learning.

Deterministic MDP: si ce paramètre est activé, l'algorithme utilisé est le Q learning déterministe décrit par la table 1.1. Sinon, il s'agit de l'algorithme non-déterministe défini à la section 1.3.8. Ce dernier permet aux agents d'apprendre dans un environnement où le résultat d'une même action dans un état particulier peut générer des états résultants différents.

Learning rate: taux d'apprentissage α de la règle de mise à jour 1.13 de l'algorithme non-déterministe. Ce paramètre est compris entre 0 et 1. Si ce paramètre est proche de 1, la nouvelle récompense sera favorisée par rapport aux anciennes valeurs de la table $\hat{Q}$.

Discount: facteur de réduction γ déterminant l'influence des récompenses futures sur la valeur d'une action dans un état particulier. Ce paramètre est compris entre 0 et 1. Une valeur proche de 1 signifie que la valeur d'une action a dans un état s sera fortement liée aux récompenses que les actions futures engendreront.

L'onglet *Plot* illustré par la figure B.3 contient les paramètres liés à l'affichage des statistiques lors de l'apprentissage:

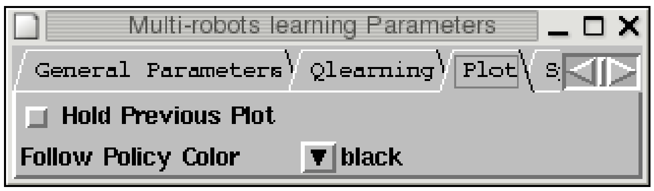


Figure B.3: Panneau de configuration permettant l'ajustement des paramètres des graphiques du panneau de statistiques.

Hold Previous Plot: si ce paramètre est activé, les nouveaux graphiques du panneau de statistiques vont venir se superposer aux anciens. S'il est désactivé les anciens graphiques seront effacés lors du prochain apprentissage.

Policy Color: définit la couleur des graphiques qui seront affichés lors du prochain apprentissage.

L'onglet *Specific Parameters* décrit par la figure B.4 permet de modifier les paramètres spécifiques à l'environnement:

Grid size: ce paramètre n'est présent que dans le module basé sur un environnement sous forme de grille. Il permet de définir le nombre de cases d'un côté de la grille.

Nodes per line: ce paramètre n'est présent que dans le module basé sur un environnement sous forme de graphe. Il permet de définir le nombre de noeuds par ligne dans le cas d'un graphe représentant une grille. Pour utiliser des graphes plus complexes, il faut entrer manuellement la matrice d'adjacence du graphe dans le fichier *Gui.oz*.

Max Path Length: longueur maximale des chemins de la politique apprise. Dans le cas où la politique apprise forme un cycle, ce paramètre permet de continuer l'apprentissage. Il définit également la hauteur maximale du graphique du panneau de statistiques représentant la longueur de chemin obtenue en fonction du nombre d'itérations.

Obstacles reward: valeur de la récompense reçue lors de la collision d'un agent avec un obstacle.

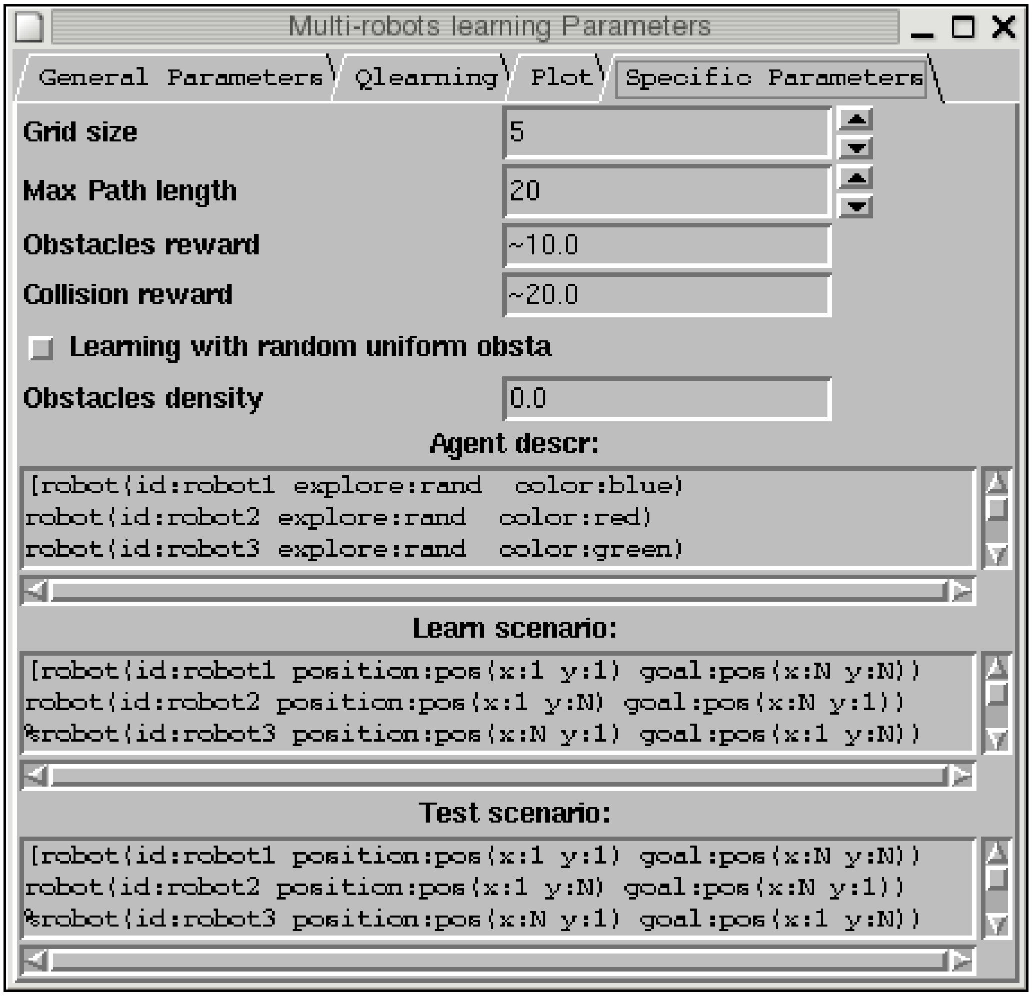


Figure B.4: Panneau de configuration permettant l'ajustement des paramètres relatifs à l'environnement d'apprentissage.

Collision reward: valeur de la récompense reçue lorsqu'un agent entre en collision avec un autre agent. Dans le cas d'agents indépendants chaque agent impliqué reçoit cette récompense. Dans celui d'une table jointe tous les agents partagent cette récompense multipliée par le nombre d'agents impliqués.

Learning with random uniform obstacles: si ce paramètre est activé des obstacles seront générés aléatoirement lors de chaque épisode de l'apprentissage. La densité de probabilité uniforme est définie par le paramètre *Obstacles density*. Les graphiques du panneau de statistiques représenteront les résultats de la politique évaluée avec les obstacles définis dans le panneau de simulation. Le paramètre *Environment Sensing* doit être activé pour que les agents puissent apprendre à éviter des obstacles dynamiques.

Obstacles density: densité de probabilité uniforme utilisée dans le cas de la génération aléatoire d'obstacles.

Agent descr: fenêtre permettant à l'utilisateur de communiquer la description des agents. La liste se trouve entre les crochets [ ] et comprend une description de chaque agent séparée par un retour à la ligne. Le caractère % placé devant la description d'un agent annule l'effet de la ligne en cours. Le format de la description d'un agent est le suivant:

robot(id:*ID* explore:*exploration* color:couleur)

où

- *ID* est un atome définissant univoquement chaque agent.
- *exploration* peut valoir **rand** pour une politique d'exploration aléatoire ou **boltz** si l'agent utilise l'approche probabiliste définie à la section 1.3.7.
- *couleur* définit la couleur de l'agent dans le panneau de simulation, ce paramètre peut prendre les valeurs **blue**, **red**, **green**, **black**, **yellow**, etc.

Learn scenario: fenêtre permettant à l'utilisateur de définir les robots utilisés lors de l'apprentissage, selon la forme:

robot(id:*ID* position:*pos* goal:*goal*)

où

- *ID* est l'un des descriptifs des agents définis dans la fenêtre *Agent descr.*
- *pos* est une chaîne de caractères de la forme **pos(x:*X* y:*Y*)** où X et Y sont les coordonnées de départ de l'agent dans la grille. La valeur de X définit le déplacement vers la droite et Y le déplacement vers le bas de la grille.
- *goal* est une chaîne de caractères **goal(x:*X* y:*Y*)** ou X et Y sont les coordonnées du goal de l'agent.

Dans le cas de l'utilisation d'une table de deux agents généralisée (paramètre –atype=**jag** du programme), il faut qu'il y ait exactement deux agents définis dans cette fenêtre.

Test scenario: fenêtre permettant à l'utilisateur de préciser les agents qui seront utilisés pour l'évaluation de la politique apprise dans le panneau de statistiques ainsi que dans la simulation des mouvements des agents. La syntaxe est identique à celle de la fenêtre *Learn scenario.*

B.2.2 Panneau de simulation

Lorsque les paramètres du panneau de configuration sont fixés, l'utilisateur doit cliquer sur le bouton *New Scenario* pour lancer le panneau de simulation illustré à la figure B.5. Ce panneau permet de lancer l'apprentissage et de visualiser la politique obtenue. L'environnement de ce panneau est initialement vide. L'utilisateur peut alors cliquer avec le bouton gauche de la souris dans celui-ci pour générer un obstacle à cet endroit. Les obstacles sont représentés en jaune. Un clic du bouton droit de la souris efface l'obstacle créé précédement et un double click permet de modifier les caractéristiques d'un obstacle.

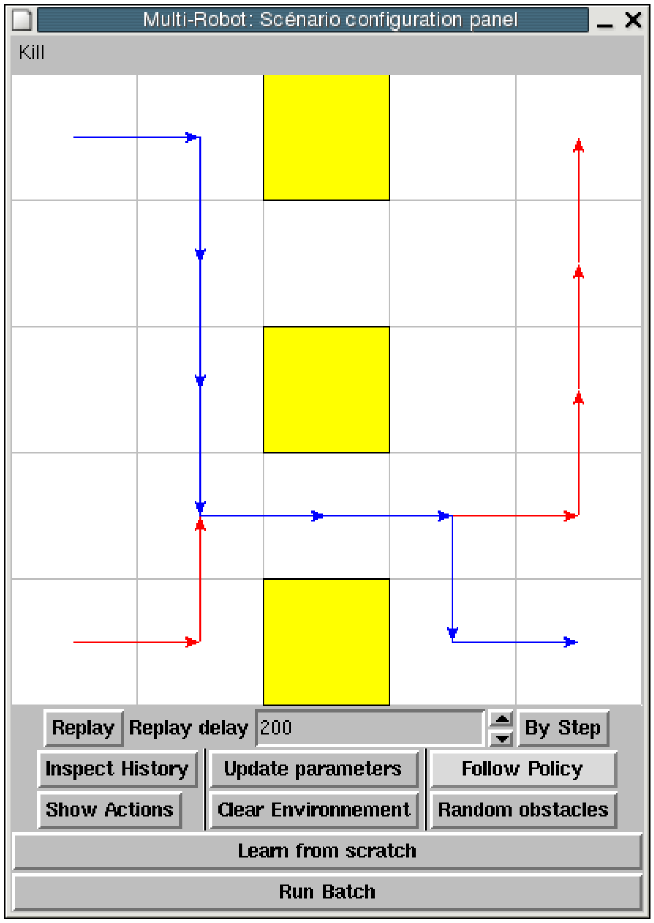


Figure B.5: Panneau de simulation permettant de visualiser la politique apprise par les agents.

A ce stade, seuls les boutons suivants peuvent être utilisés:

Update Parameters: récupère les paramètres du panneau de configuration s'ils ont été modifiés depuis que le panneau de simulation est affiché.

Random Obstacles: génère des obstacles avec une probabilité de distribution. Cette probabilité est spécifiée dans le panneau de configuration.

Learn from scratch: lance un apprentissage, dont le déroulement peut être observé à l'aide du panneau de statistiques.

Run Batch: lance une série d'apprentissages dont les statistiques seront enregistrées dans un fichier texte. La figure B.6 illustre la fenêtre correspondante, dans laquelle il est possible de spécifier le nombre d'apprentissages souhaités, le répertoire de travail ainsi que le fichier dans lequel les statistiques seront stockées.

Une fois l'apprentissage terminé, la simulation du mouvement des agents apparait dans le panneau de simulation. Les agents vont ainsi de leur point de départ à leur goal respectif. L'utilisateur peut alors utiliser les boutons suivants:

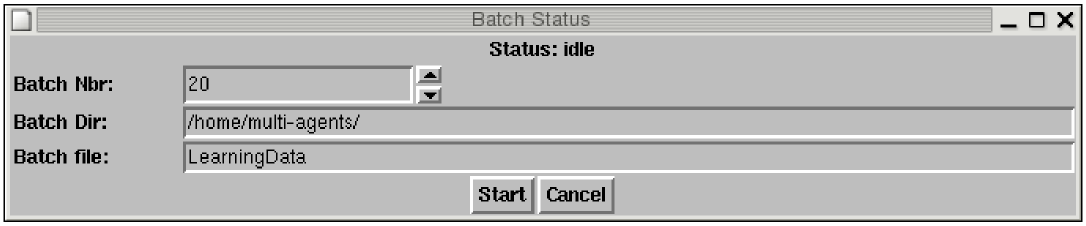


Figure B.6: Panneau de simulation permettant de lancer plusieurs apprentissages à la suite afin d'effectuer des statistiques.

Replay: relance l'affichage de la dernière simulation effectuée. Le paramètre *Replay delay* définit le temps entre chaque déplacement.

By Step: effectue un déplacement à la fois.

Inspect History: affiche un descriptif textuel des déplacements des robots.

Show Actions: affiche des flèches décrivant le parcours de chacun des agents.

Clear Environment: efface le contenu du panneau de simulation. Il faut appuyer sur *Update Parameters* pour afficher l'environnement à nouveau.

Follow Policy: relance la simulation du déplacement des agents. Les chemins empruntés peuvent différer car il est possible qu'une politique apprise donne lieu à plusieurs parcours possibles. De plus, il peut arriver que les actions des agents ne se produisent pas dans les mêmes intervalles de temps ce qui peut produire des résultats de qualité différente d'une simulation à une autre.

B.2.3 Panneau de statistiques

Lors de l'apprentissage, un panneau de statistiques tel que celui de la figure B.7 apparait. Ce panneau contient deux graphiques décrivant l'apprentissage en temps réel.

Le premier graphique illustre le nombre de collisions lors de l'évaluation de la politique apprise en fonction du nombre d'itérations d'apprentissage. Ce nombre comprend les collisions entre agents et les collisions avec les obstacles. Une collision entre n agents compte pour n collisions dans le graphique. Si l'apprentissage se déroule correctement et si l'environnement le permet, le nombre total de collisions doit décroître et atteindre une valeur nulle.

Le second graphique décrit la longueur du chemin obtenu en faisant suivre aux agents la politique apprise, en fonction du nombre d'itérations d'apprentissage. La politique est donc évaluée entre chaque épisode avec l'environnement défini dans le panneau de simulation. Si l'apprentissage se déroule bien le graphe doit décroître pour atteindre la longueur de chemin minimale. Dans le cas de plusieurs agents la longueur du chemin correspond à celle de l'agent dont le parcours est le plus long.

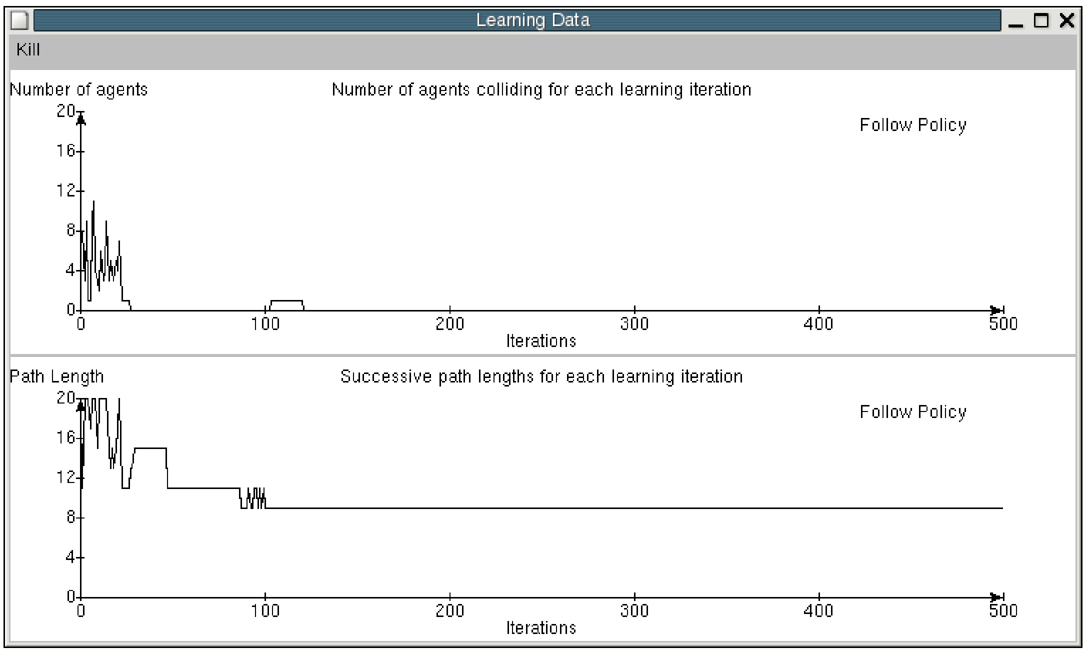


Figure B.7: Panneau de statistiques décrivant la longueur du chemin obtenu et le nombre total de collisions en fonction de l'itération d'apprentissage.