MASTER EN SCIENCES INFORMATIQUES

Apprentissage dans un système d'auto-organisation

Pardoen, Maxime

Award date:
2013

Awarding institution:
Universite de Namur

Apprentissage dans un système d'auto-organisation

Maxime PARDOEN

Promoteur : Prof. Jean-Noël Colin

Signature du promoteur attestant son approbation au dépôt du mémoire :

Mémoire présenté
en vue de l'obtention du titre de
Master 120 en Sciences informatiques

ANNEE ACADEMIQUE 2012-2013

Remerciements

Je tiens à remercier mon promoteur Jean-Noël Colin pour ses critiques lors de la rédaction du mémoire. Je remercie également les responsables de l'Institut de la Francophonie pour l'Informatique de Hanoï, les membres du laboratoire MSI et, tout particulièrement, mon maitre de stage Manh Hung Nguyen pour leur accueil et leur soutien pendant toute la durée de mon stage. Enfin, je souhaite remercier toutes les personnes qui ont participé à la relecture et la correction de ce travail.

Abstract

The goal of this work is to introduce an efficient method of learning, aiming to optimize the behavior of agents in a multi-agents system featuring an auto-organization mechanism. This mechanism allows it to adapt its behavior according its agents trust values. This learning weights on the importance of different criterias which determines an agent's behavior concerning movements and communications in such a system. To do so, this work will describe the fondamental notions over the multi-agents systems theory, will introduce the concept of learning and will study the different methods used to add this learning to different kinds of systems. The algorithm described in details here is the Q-Learning, a reinforcement learning algorithm. The case study introduced in this memoire is an application of this algorithm to an existing model, Danger Mapping, allowing the simulation of an auto-organized multi-agents system by the GAMA platform. The results of this new model will be compared with the those obtained when the behavior rules of the agents in Danger Mapping are optimized using a genetic algorithm.

Résumé

L'objectif de ce travail est de présenter une méthode efficace d'apprentissage visant à optimiser le comportement d'agents dans un système multi-agents présentant un mécanisme d'auto-organisation. Celui-ci lui permet de modifier son comportement pour réagir à l'apparition d'agents non fiables. Cet apprentissage porte sur l'importance des différents critères qui influent sur les déplacements et les communications des agents présents dans un tel système. Pour ce faire, ce travail décrira les notions fondamentales entourant la théorie sur les systèmes multi-agents, présentera le concept d'apprentissage et étudiera les différentes méthodes qui permettent d'appliquer celui-ci dans différents systèmes. L'algorithme d'apprentissage qui sera détaillé est le Q-Learning, un algorithme d'apprentissage par renforcement. L'étude de cas présentée dans le présent mémoire est l'intégration de cet algorithme d'apprentissage dans un modèle existant, Danger Mapping, qui permet de simuler un système multi-agents auto-organisé grâce à la plateforme GAMA. Au terme de celle-ci, les résultats seront comparés avec ceux, théoriques, obtenus en appliquant un algorithme génétique pour optimiser l'importance accordée aux règles de comportement.

Table des matières

Table des figures

Chapitre 1

Introduction

Depuis les premiers travaux suggérant la possibilité de créer des intelligences artificielles, ce domaine a nourri de nombreux espoirs. La formalisation de problèmes complexes de la vie courante présente néanmoins plus de difficultés que les pionniers du domaine l'avaient supposé. Pour remédier à ce problème, une solution courante consiste à subdiviser ce problème en une série de sous problèmes qu'un ensemble d'agents essaient de résoudre de concert. Ces systèmes multi-agents permettent de résoudre des problèmes de grande envergure, notamment lorsque ceux-ci coopèrent efficacement.

Ces agents se heurtent néanmoins à d'importants problèmes lorsque l'environnement est inconnu ou qu'il présente un trop grand nombre de combinaisons entre les états possibles de celui-ci et les actions envisageables, c'est-à-dire lorsqu'il n'est donc pas possible de trouver une suite d'actions optimales. Une solution couramment acceptée à ce problème est l'introduction d'une forme d'apprentissage pour ces agents. En effet, en permettant à ceux-ci d'apprendre par expérience à prendre les meilleures décisions, la performance de ces agents croit jusqu'à atteindre une valeur proche de l'optimal. Bien que l'apprentissage a souvent pour objet le choix des actions à entre-

prendre, celui-ci peut également porter sur d'autres composants de l'agent impliqués dans l'évaluation de la récompense perçue après une action, l'exploration de nouveaux états ou la modification de l'utilité escomptée pour chaque action.

La recherche d'une coopération efficace entre les différents agents est aussi importante dans les systèmes multi-agents que les performances individuelles de ceux-ci. Il est donc essentiel que ceux-ci présentent une bonne coordination. Ainsi, les systèmes auto-organisés voient leur agents apprendre progressivement une organisation qui leur permet de maximiser leur efficacité. Cette recherche conduit souvent les agents à se diviser en rôles distincts en fonction de leurs caractéristiques, rôles qui influencent le comportement de ces agents en leur permettant d'adapter celui-ci aux propriétés de l'environnement, des autres agents et de lui-même.

Si cette auto-organisation permet une amélioration des interactions entre les agents d'un même système, l'importance de celle-ci est accordée au détriment de celle associée aux performances individuelles des agents. Il convient donc de la calibrer en fonction de l'environnement pour atteindre des performances optimales. Divers travaux [8] ont permis de trouver un calibrage optimal pour un environnement précis. Ces résultats sont néanmoins strictement théoriques et demandent une connaissance totale du système, ce qui n'est pas le cas des agents en pratique. Le problème qui se pose est : "Comment calibrer efficacement l'importance associée aux performances individuelles des agents et celle associée aux interactions entre ceux-ci dans un environnement inconnu ?".

Le but du présent document est de présenter l'intérêt d'intégrer un apprentissage supplémentaire pour chaque agent d'un système multi-agents auto-organisé. Celui-ci permettrait à l'importance qu'un agent accorde à l'organisation du système quant à son comportement d'évoluer. A cette fin, ce travail introduit les différents concepts liés aux systèmes multi-agents, à l'ap-

prentissage et à l'organisation de tels systèmes. Il présente ensuite une solution au problème précédemment cité et applique cette dernière à un cas d'étude concret, le modèle de système multi-agents auto-organisé Danger Mapping. Ce document analyse et discute des résultats de cette application, tire des conclusions de ceux-ci et présente des pistes pour de futurs travaux.

Chapitre 2

Intelligence artificielle

L'intelligence artificielle est un domaine qui a connu, au cours des dernières décennies, une série de phases de succès et d'optimisme démesuré, mais aussi de phases de pessimismes quant aux limitations sur le développement de celui-ci. L'usage de méthodes scientifiques dans la comparaison des différentes approches et dans l'expérimentation a considérablement accéléré les progrès dans le domaine au cours de la dernière décennie. Certaines notions, parmi lesquelles les agents, les environnements, les systèmes multi-agents, l'apprentissage et l'auto-organisation, sont désormais couramment acceptées dans la littérature et sont centrales à bon nombre d'approches. Ce chapitre définit différents concepts utilisés dans les publications contemporaines sur l'intelligence artificielle et décrit les méthodes d'apprentissage qui seront nécessaires pour comprendre les enjeux et techniques utilisés dans l'étude de cas présentée au chapitre 3.

2.1 Agent

La notion d'agent, commune à beaucoup de publications scientifiques liées à l'intelligence artificielle, possède un grand nombre de définitions. Parmi celles-ci, voici celles que nous jugerons les plus pertinentes :

- "On appelle agent toute entité qui peut être considérée comme percevant son environnement grâce à des capteurs et qui agit sur cet environnement via des effecteurs." [14]
- "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." [19]
- "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]

On dit d'un agent qu'il est rationnel si il correspond à la définition suivante :

> *Pour chaque séquence de percepts possible, un agent rationnel doit sélectionner une action susceptible de maximiser sa mesure de performance, compte tenu des observations fournies par la séquence de percepts et de la connaissance dont il dispose.* [14] [1]

La mesure de performance d'une séquence d'états de l'environnement permet de quantifier le fait que les actions de l'agent provoquent des changements dans l'environnement désirables pour la réalisation de ses objectifs.

Un agent est dit *réactif* si ses actions sont uniquement effectuées en réponse aux changements perçus dans son environnement, ne tenant compte

1. Il est à noter qu'un agent rationnel n'est pas nécessairement omniscient. Un agent omniscient connait le résultat *réel* de ses actions alors que l'agent rationnel, lui, ne considère que le résultat *attendu* de celles-ci.

que du percept courant. Un tel agent confronte celui-ci à un série de règle *condition-action* pour décider quelle action effectuer. Les agents réflexes sont remarquablement simples mais leur intelligence est limitée et ils ne sont efficaces que s'il est possible de prendre la bonne décision sur la base du seul percept courant, c'est-à-dire seulement si l'environnement est entièrement observable. À l'inverse, un agent *cognitif* est plus complexe et peut posséder une forme d'*état interne* lui permettant de modéliser l'état d'un environnement qui n'est pas observable dans sa totalité. Il peut également avoir un ou plusieurs buts. L'exploration et la planification sont des sous-domaines de l'IA qui se consacrent à la recherche de la séquence d'actions qui permettent à un agent d'atteindre ses buts. Pour ce faire, il est souvent nécessaire d'introduire une fonction d'utilité pour permettre une internalisation de la mesure de performance.

Les différents composants d'un agent interagissent avec la *représentation interne* que cet agent se fait de son environnement. L'expressivité de cette représentation, et donc sa complexité, est choisie parmi les trois niveaux suivants :

Atomique. Une représentation des états et des transitions entre états est atomique si chaque état ne possède aucune structure interne.

Factorisée. Une représentation des états et des transitions entre états est factorisée si chaque état est constitué d'un vecteur contenant la valeur des attributs liés à cet état.

Structurée. Une représentation des états et des transitions entre états est structurée si chaque état est constitué d'une collections d'objets possédant des attributs et des relations entre eux.

Un exemple d'agent serait un robot qui joue aux échecs. Les senseurs de celui-ci l'avertissent à la fin du tour du joueur adverse qu'il doit jouer, et l'informent également de l'état de l'échiquier. Après avoir converti cette entrée en représentation interne, l'agent envisage les actions possibles, en

prédisant l'état engendré par chacune d'elles et en évaluant l'utilité de celui-ci. Le robot choisit l'action qui maximise cette utilité attendue et l'exécute via ses effecteurs. Un tel agent est cognitif s'il peut prévoir l'utilité de ses actions au-delà de leur utilité immédiate, sinon il est réactif. La représentation interne la plus intuitive est structurée, chaque pièce sur l'échiquier possédant des caractéristiques qui lui sont propres. Le but recherché par le robot est la victoire de la partie.

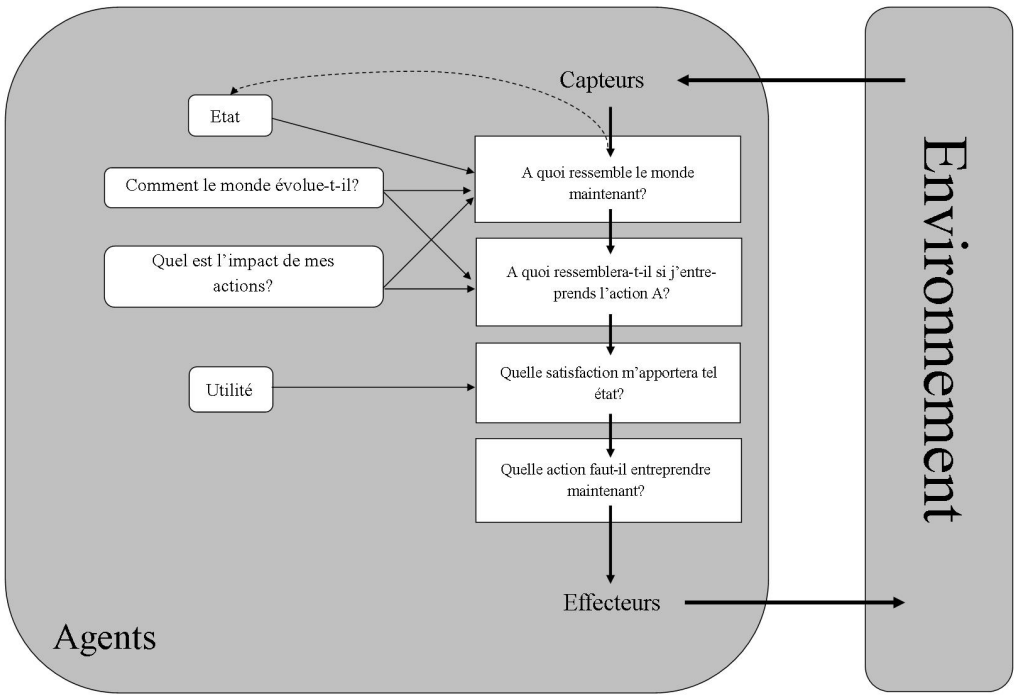

FIGURE 1 – Structure d'un agent sans apprentissage [14]

2.2 Environnement

L'environnement est le système avec lequel interagit l'agent et dont il fait partie. C'est celui-ci qui produit les informations nécessaires aux capteurs de l'agent et c'est sur celui-ci que sont effectuées les actions de ce dernier.

Un environnement peut être :

Entièrement/Partiellement observable :

L'environnement est dit entièrement observable si les capteurs d'un agent lui donnent accès à la totalité des informations pertinentes sur l'état de l'environnement à tout moment. La pertinence de ces informations dépend de la mesure de performance pour cet agent.

Monoagent/Multi-agents :

Un environnement est multi-agents s'il présente plusieurs agents. Les autres entités présentes dans l'environnement peuvent être considérées comme simples composants de cet environnement ou comme agents à part entière, notamment lorsque la mesure de performance de l'agent considéré est directement liée au comportement de ces entités. [2] Les environnements multi-agents introduisent souvent la communication comme comportement rationnel.

Déterministe/Stochastique :

S'il est possible de déterminer avec certitude quel sera l'état suivant de l'environnement sur base de l'état courant et de l'action choisie, alors cet environnement est déterministe. Sinon, il est stochastique. Un environnement non entièrement observable peut paraitre stochastique, même lorsque son comportement est déterministe. Si les probabilités attachées à ces changements d'état ne sont pas déterminées, l'environnement est dit *non déterministe*.

2. Ainsi, les échecs sont un environnement multi-agents concurrentiel alors que des fourmis œuvrant ensemble à accumuler de la nourriture sont un environnement multiagent coopératif.

Épisodique/Séquentiel :

Un environnement est épisodique si l'expérience de l'agent est divisée en épisodes atomiques pendant lesquels l'agent reçoit un percept et choisit une action à exécuter. À l'inverse, dans les environnements séquentiels, toute décision est susceptible d'affecter les décisions futures.

Statique/Dynamique :

Un environnement est dynamique s'il peut changer sans action de l'agent, à mesure que le temps s'écoule. Un tel environnement peut changer alors que l'agent est en train de décider quelle action effectuer, obligeant celui-ci à évaluer sans cesse l'état du monde. [3] À l'inverse, un environnement est statique si l'environnement ne change qu'en conséquence d'une action de l'agent.

Discret/Continu :

L'espace des états de l'environnement, du temps, des percepts et des actions peut être discret ou continu.

Connu/Inconnu : L'agent peut connaitre les règles qui régissent son environnement. Si tel est le cas, l'environnement est considéré comme connu.

Un environnement partiellement observable, multi-agents, non déterministe, séquentiel, dynamique, continu et inconnu est un cas extrêmement difficile. Beaucoup d'activités de la vie courante répondent pourtant à ces critères, ce qui rend l'exécution de tâches habituelles pour l'homme par un agent artificiel particulièrement complexe ou inefficace. Un exemple d'un tel environnement serait celui dans lequel évolue un robot piéton qui doit se déplacer efficacement dans les rues bondées de New York. La seule caractéristique de cet environnement qui soulage un peu sa complexité est son aspect partiellement connu. Il est en effet prévisible que les autres piétons n'apprécient pas que

3. Un environnement peut également être *semi-dynamique* si l'environnement ne change pas au cours du temps mais si la mesure de performance de l'agent, elle, est affectée par l'écoulement du temps. Ainsi, lorsqu'on utilise un chronomètre, le jeu d'échecs est semi-dynamique.

l'agent s'arrête au milieu de la rue, que ne pas respecter le code de la route est pénalisant et que les agents qui interfèrent avec les actions de l'agent piéton sont moins nombreux un jour de pluie.

Le tableau suivant donne plusieurs exemples d'environnements, ainsi que leurs caractéristiques :

Environnement de tâche	Observable	Agents	Déterministe	Episodique	Statique	Discret
Problème de mots croisés	Entièrement	Mono	Déterministe	Séquentiel	Statique	Discret
Partie d'échecs chronométrée	Entièrement	Multi	Déterministe	Séquentiel	Semi	Discret
Poker	Partiellement	Multi	Stochastique	Séquentiel	Statique	Discret
Backgammon	Entièrement	Multi	Stochastique	Séquentiel	Statique	Discret
Conduite de taxi	Partiellement	Multi	Stochastique	Séquentiel	Dynamique	Continu
Diagnostic médical	Partiellement	Mono	Stochastique	Séquentiel	Dynamique	Continu
Analyse d'images	Entièrement	Mono	Déterministe	Episodique	Semi	Continu
Robot détecteur de défauts	Partiellement	Mono	Stochastique	Episodique	Dynamique	Continu
Contrôleur de raffinerie	Partiellement	Mono	Stochastique	Séquentiel	Dynamique	Continu
Répétiteur d'anglais interactif	Partiellement	Multi	Stochastique	Séquentiel	Dynamique	Discret

FIGURE 2 – Exemples d'environnements [14]

2.3 Apprentissage

Dans son article *Computing machinery and Intelligence*, publié en 1950, Alan Turing fait une estimation de la quantité de travail que prendrait la programmation à la main de ses machines intelligentes avant d'en venir à la conclusion qu' "une méthode plus rapide serait préférable". En effet, la création d'agents intelligents est extrêmement laborieuse lorsque le système est un tant soit peu complexe. [4] La solution proposée par Alan Turing consiste à construire des machines capables d'apprentissage puis à les instruire. Cette solution, désormais courante dans de nombreux domaines liés à l'intelligence artificielle, présente l'avantage que l'agent doté d'apprentissage peut opérer dans des environnements inconnus au départ et améliorer ses compétences au fur et à mesure qu'il interagit avec celui-ci. *"Plutôt que d'essayer de produire un programme qui simule l'esprit d'un adulte, pourquoi ne pas plutôt tenter de simuler celui d'un enfant"*. [16]

2.3.1 Les composants de l'agent apprenant

Un agent doué d'apprentissage peut être divisé en 4 composants distincts :

Le critique transmet au composant d'apprentissage un feed-back quant à la dernière action effectuée. Cette information est calculée en utilisant les percepts reçus par les capteurs et les critères de performance fixés au préalable.

Le composant de performances est chargé de sélectionner les actions à prendre. Il reçoit les percepts des capteurs et décide quelle action effectuer en fonction de l'utilité attendue de cette action.

4. Si la création d'agents réactifs dans un environnement simple ne pose pas de grande difficulté, on observe une explosion combinatoire du nombre de règles de comportement nécessaires pour des agents ou un environnement plus complexes.

Le composant d'apprentissage utilise le feed-back du critique au sujet des actions passées et détermine comment il convient de modifier le composant de performances pour améliorer celui-ci et faire mieux à l'avenir.

Le générateur de problèmes suggère des actions exploratoires qui aboutiront à des expériences nouvelles et instructives. En explorant des possibilités sous-optimales à court terme, il est possible de découvrir des solutions beaucoup plus efficaces à long terme.

Il est à noter que si les modifications proposées par le composant d'apprentissage concernent généralement le composant de performances, il n'est pas impossible que celles-ci puissent porter sur d'autres composants. Ainsi, si le générateur de problèmes suggère une action exploratoire qui permet de découvrir une solution plus efficace, ce dernier peut en déduire que l'exploration est un comportement souhaité.

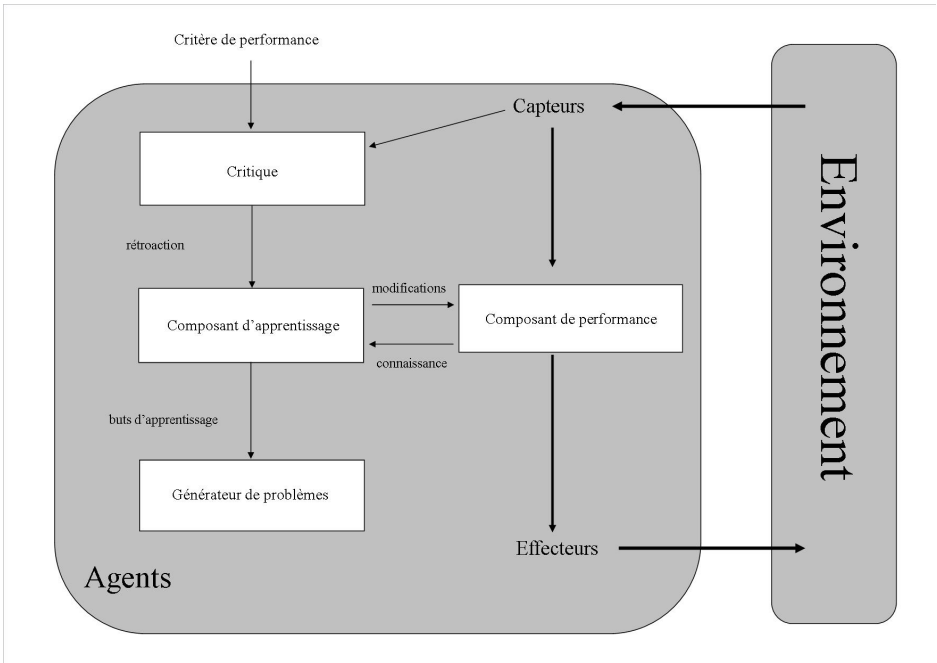

FIGURE 3 – Structure d'un agent avec apprentissage [14]

2.3.2 Les différents types d'apprentissage

Il existe de nombreuses formes d'apprentissage selon la nature de l'agent, du composant à améliorer et du retour d'expérience disponible par le biais des percepteurs. On dit d'un apprentissage qu'il est *inductif* s'il permet d'apprendre des règles à partir d'exemples observés de couples action-résultat. À l'inverse, il est également possible d'effectuer un apprentissage *déductif* ou *analytique*, c'est-à-dire partir d'une règle connue pour en obtenir une nouvelle, plus utile et contenue logiquement dans la première.

L'apprentissage dépend, en grande partie, du retour d'expérience procuré par les percepteurs et, le cas échéant, du critique. Il existe trois types de feed-back qui déterminent ces types d'apprentissage :

1. *L' apprentissage non supervisé* est celui qui consiste à apprendre des percepts lorsque ceux-ci ne représentent pas un feed-back explicite sur ses actions. Il s'agit donc d'une analyse de l'environnement quand l'agent n'intervient pas. Ce type d'apprentissage est donc plus efficace dans un environnement dynamique.
2. Dans *l'apprentissage par renforcement*, l'agent apprend en confrontant les actions entreprises précédemment avec les récompenses ou les punitions obtenues. C'est l'agent qui décide ou déduit quelles actions sont responsables du résultat et qui adapte leur utilité.
3. *L'apprentissage supervisé* consiste à apprendre une fonction de l'entrée (percepts) vers la sortie (actions) après avoir observé plusieurs couples entrée-sortie.

Par exemple, considérons un agent "élève". Cet agent peut effectuer, par induction, un apprentissage non supervisé qui consiste à prédire les questions du prochain test en se basant sur les tests précédents. Ce même élève, au vu du résultat plus ou moins fructueux de ses méthodes d'étude, va adapter son comportement d'étude en fonction des résultats qu'il en retire. Enfin, les

conseils avisés de son professeur lui permettront d'établir des correspondances précises entre une entrée (question) et une sortie efficace (réponse). [5]

2.3.3 L'apprentissage par renforcement

L'apprentissage par renforcement est une classe d'algorithmes dont le fonctionnement consiste, pour un agent, à apprendre du succès et de l'échec de ses actions comment interagir avec son environnement et à choisir, de manière autonome et rationnelle, les actions optimales pour atteindre son but. Il est donc applicable aux agents qui poursuivent un but ou qui essaient de maximiser leur mesure de performance. L'apprentissage par renforcement utilise les récompenses observées pour apprendre une politique optimale pour un environnement défini. Par exemple, un agent "joueur" qui apprend à jouer au tennis recevra une récompense à chaque fois que ses actions le mènent à gagner un échange. Inversement, les actions qui le mènent à perdre l'échange entraineront une punition. Ainsi, un agent qui tente de gagner va apprendre à utiliser les actions maximisant la récompense totale jusqu'à atteindre son but.

Le principal avantage de l'apprentissage par renforcement est que l'agent peut ne connaitre ni son but, ni son environnement. Au fur et à mesure de ses expériences, l'agent construira son propre modèle de l'environnement et fixera un ensemble de règles définissant un comportement efficace dans celui-ci. Il est important que l'agent, par le biais de son générateur de problèmes, explore son environnement pour déterminer les renforcements efficaces qui lui permettront d'apprendre un comportement adapté à celui-ci. Il existe deux manières d'évaluer la valeur de la récompense pour une action donnée qui constituent deux schémas d'apprentissage différents. On peut soit apprendre la fonction qui détermine l'utilité d'être dans un état particulier,

5. Il est à noter que ces différentes catégories ne sont pas toujours clairement distinctes. Ainsi, il existe aussi *l'apprentissage semi-supervisé*.

soit la fonction action-valeur, ou fonction-Q ,qui associe une utilité à une action particulière depuis un état initial donné. Le premier schéma nécessite un modèle de l'environnement, sans lequel l'utilité de l'état résultat d'une action puisque ce résultat est inconnu. Le second, lui, ne nécessite pas d'avoir un modèle de l'environnement mais, faute de savoir où ses actions vont conduire l'agent, il ne peut pas anticiper, ce qui limite sa capacité à apprendre.

2.3.4 Q-Learning

L'algorithme utilisé dans l'étude de cas est le *Q-Learning*, un algorithme d'apprentissage par renforcement. Un agent utilisant le Q-Learning apprend une "fonction action-valeur" ou fonction-Q, qui donne l'utilité espérée de la réalisation d'une action donnée dans un état donné. Cette fonction-Q fournit une estimation de cette utilité, appelée Q-valeur. Celles-ci sont évaluées à partir des récompenses provenant de l'environnement sous forme de percepts. Le Q-Learning est un algorithme itératif qui évalue progressivement les Q-valeurs de chaque couple état-action lors de son exploration de l'environnement. Au fur et à mesure de cette exploration, un modèle est construit en même temps que la table Q qui définit les Q-valeurs de chacun des couples état-action.

L'utilisation du Q-learning suppose plusieurs hypothèses :

- Il existe un ensemble S qui contient tous les états possibles de l'agent, un ensemble A qui contient toutes les actions que l'agent peut effectuer. À un moment t, l'agent est dans un état s_t appartenant à S et choisit une action a_t appartenant à A.
- Après chaque action, l'environnement renvoie à l'agent une récompense $r(s_t, a_t)$ sous forme d'un nombre et l'état résultat $s_{t+1} = \delta(s_t, a_t)$.
- L'agent peut ne pas connaitre les fonction r et δ. En effet, l'agent ne possède pas de modèle de l'environnement et ne sait donc pas à l'avance

le résultat de ses actions.

– Le problème doit pour être modélisé sous la forme d'un Processus de Décision Markovien (MDP). Les fonction $\delta(s_t, a_t)$ et $r(s_t, a_t)$ ne doivent donc dépendre que de l'état et de l'action effectuée au temps t, et non des états et des actions antérieurs.

Le raisonnement mathématique derrière le Q-Learning pour un MDP déterministe est le suivant : pour chaque paire (s, a) où l'action a est exécutée depuis l'état s, on associe une grandeur Q qui est l'espérance de gain lorsqu'on effectue l'action a depuis cet état s. L'espérance de gain est la somme de l'espérance de gain immédiat de l'action choisie et des espérances de gain de tous les états, pondérés par la probabilité d'atteindre chacun d'eux depuis l'état s en menant l'action a. Dans le cas d'une politique idéale et en supposant que r_t est la récompense pour l'action (s_t, a_t) effectuée à l'instant t, on écrit :

$$Q^*(s, a) = E_{r \times \pi(s,a)}[r(s, a) = \gamma V^*(s')] \tag{2.1}$$

V^* peut être écrite grâce à Q^* selon l'équation de Bellman :

$$V^*(s) = \max_{a \in A} Q^*(s, a) \tag{2.2}$$

Ce qui permet d'obtenir la définition récurrente de Q^* :

$$Q^*(s, a) = E_{r \times \pi(s,a)}[r(s, a) = \gamma \max_{a \in A} Q^*(s', a)] \tag{2.3}$$

L'algorithme du Q-Learnning cherche une approximation Q^* comme point fixe de l'équation précédente, en écrivant :

$$Q_{t+1}(s_t, a_t) = (1 - \alpha_t)Q_t(s_t, a_t) + \alpha_t[r_t + \gamma_t * \max_{a' \in A} Q_t(s_{t+1}, a')] \tag{2.4}$$

Cette équation permet de mettre à jour la valeur courante de $Q(s_t, a_t)$ grâce à sa valeur précédente et à l'entrée précédente.

On en déduit l'algorithme suivant :

Algorithm 1 Q-Learning

initialiser $Q_0(s, a)$ arbitrairement
choisir un point de départ s_0
while la politique n'est pas assez bonne **do**
  choisir a_t en fonction de $Q_t(s_t)$
  l'action produit s_{t+1} et r_t
  $Q_{t+1}(s_t, a_t) \leftarrow (1 - \alpha_t)Q_t(s_t, a_t) + \alpha_t(r_t + \gamma * \max_{a' \in A} Q_t(s_{t+1}, a'))$
  $t \leftarrow t + 1$
end while

Le paramètre α doit être réglé pour fixer la politique apprise au fur et à mesure, au détriment de l'exploration. Le paramètre γ permet de fixer quelle importance ont les récompenses à court terme par rapport à celles à plus long terme. Dans la pratique, α doit être compris entre 0 et 1 et décroitre vers 0 au fur et à mesure que les Q-Value des différents couples (a, s) sont proches de leur valeur d'utilité réelle. Quant à γ, on doit choisir un γ constant compris entre 0 et 1. Le critère "tant que la politique n'est pas assez bonne" n'est pas défini spécifiquement et dépendra des critères d'apprentissage désirés. On peut limiter l'apprentissage à un certain nombre d'itérations comme pour la définition des courbes d'apprentissage ou laisser un apprentissage continu qui ne s'arrêtera que lorsque le but, s'il existe, est atteint.

Différentes politiques peuvent être envisagées pour le choix des actions :

Le choix aléatoire est la plus simple de celles-ci. Elle correspond à une distribution uniforme des probabilités de choix pour chacune des actions envisagées. Bien que très facile à mettre en oeuvre, cette solution est bien souvent inefficace et peut nécessiter un temps d'exploration très long.

L'approche probabiliste associe à chaque action une probabilité d'autant plus haute que la récompense escomptée de cette dernière est élevée.

L'inconvénient de cette politique de choix est que si l'agent choisit à chaque fois l'action maximisant l'estimation $\hat{Q}$, il n'explorera pas toutes les paires état-action possibles et ce choix pourrait être sous-optimal. Pour régler ce problème, on utilise souvent une *distribution de Boltzmann*,

$$w(a) = \frac{e^{\frac{Q(s,a)}{T}}}{\sum_{b \in A} e^{\frac{Q(s,b)}{T}}} \tag{2.5}$$

où la température [6] T est une valeur élevée au début et qui décroit vers 0 à la fin de l'apprentissage. Cette valeur permet d'éviter les maxima locaux en créant une forme d'instabilité dans le choix de l'action à effectuer, ce qui dans le cas présent est associé à l'exploration de solutions.

L'algorithme du Q-Learning converge vers la fonction Q^* lorsque les conditions suivantes sont réunies :

- Le problème est modélisable par un processus décisionnel de Markov (MDP) déterministe.
- Les valeurs des récompenses immédiates sont bornées, c'est-à-dire $\forall s, a \exists e$: $|r(s,a)| < e$
- L'agent sélectionne ses actions de façon à visiter chaque paire état-action possible une infinité de fois.

La démonstration du théorème de convergence ne sera pas détaillée ici. Nous pouvons néanmoins décrire l'idée principale derrière celui-ci. L'entrée de la table $\hat{Q}(s,a)$ comportant la plus grande erreur réduit celle-ci d'un facteur γ à chaque itération qui la met à jour, la nouvelle valeur étant la somme de la récompense immédiate sans erreur et de l'estimation erronée $\hat{Q}$ réduite par un facteur γ

Un exemple d'application de l'algorithme Q-learning consisterait à ap-

6. Le terme température est une analogie aux techniques de fabrication d'alliages. Il ne s'agit en aucun cas d'une véritable température.

prendre à un robot mobile à trouver le chemin qui le mènera à destination. Au début de cet algorithme, les valeur-Q sont initialisées à 0, pour tous les couples état-action possibles. L'environnement ne procure de récompense (100) que lorsque le robot atteint son but, la case annotée d'une croix dans le schéma ci-dessous. Le robot se dirige d'abord de manière complètement aléatoire. Lorsqu'il arrive la première fois dans l'état final, il associe à la dernière action sur l'état précédent une valeur-Q égale à la récompense, c'est-à-dire 100 (Fig.4). En supposant que $\gamma = 0.9$, les valeurs-Q des autres couples action-état sont successivement mises à jour en fonction des valeurs-Q des états résultants dont la valeur-Q est maximale(Fig.5). Pour finir, l'ensemble des valeurs-Q associées à chaque couple état-action est défini et la politique de déplacement optimale est celle qui maximise ces valeurs à chaque action (Fig.6).

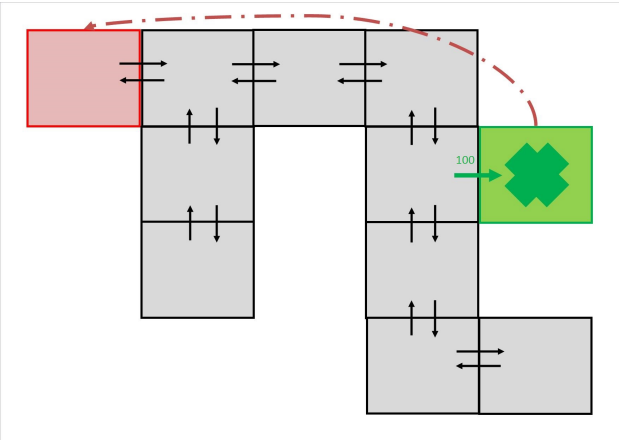

FIGURE 4 – État des valeurs-Q pour un robot parcourant un labyrinthe après le premier parcours

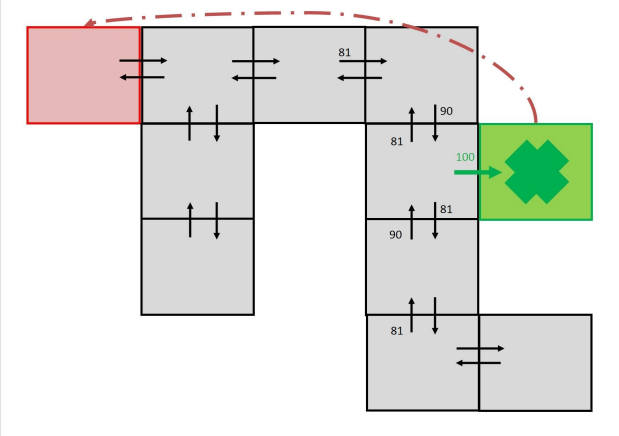

FIGURE 5 – Mise à jour de l'État des valeurs-Q pour un robot parcourant un labyrinthe

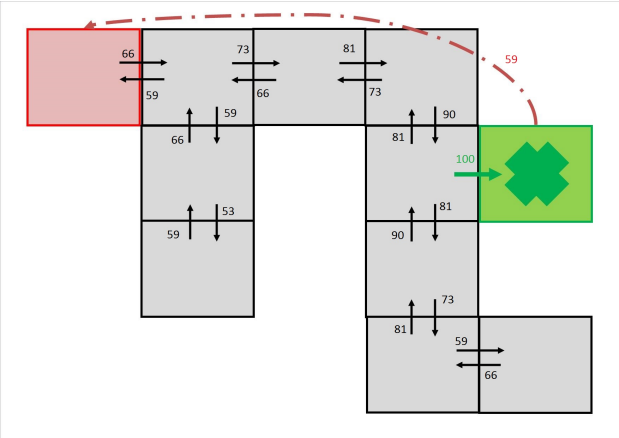

FIGURE 6 – État des valeurs-Q pour un robot parcourant un labyrinthe après l'apprentissage

Il existe différentes techniques permettant d'améliorer les performances de base de l'algorithme Q-Learning dans certains cas :

L'apprentissage de primitives de haut niveau est une technique permettant d'améliorer l'efficacité du Q-Learning en diminuant le nombre d'états possibles. En effet, la complexité de cet algorithme augmente exponentiellement avec le nombre d'états qui caractérisent l'agent. En regrou-

pant les actions sous forme de comportements, il est possible de diminuer grandement le temps de calcul nécessaire à l'algorithme, même en consacrant une partie de celui-ci à définir ces comportements de manière efficace.

L'estimateur de progrès est une fonction continue dans le temps qui mesure le chemin parcouru pour atteindre un objectif. Ainsi, l'agent peut attribuer une récompense sans avoir atteint le but final, ce qui permet plus de robustesse. Bien que les estimateurs de progrès demandent une certaine connaissance du domaine, ils sont moins dépendants du bruit des capteurs, ils peuvent encourager l'exploration en récompensant la découverte et ils permettent de relativiser les récompenses dues à des coïncidences. De manière générale, les résultats expérimentaux tendent à montrer que l'apprentissage combinant les estimateurs de progrès et les récompenses une fois le but atteint sont plus efficaces que le Q-Learning traditionnel.

Organiser l'espace état-action en arbre ou en réseau de neurones, plutôt qu'en table permet un gain de temps de calcul non négligeable lors du parcours de ce dernier, particulièrement dans le cas où cet espace état-action comporte un très grand nombre d'entrées.

L'algorithme du Q-Learning, tel que nous l'avons décrit jusqu'à présent, considère toujours que les récompenses et que les états résultant d'un couple état-action sont fixes. Dans le cas contraire, si un couple état-action pouvait, à deux moments différents, donner lieu à des récompenses et des états résultats différents, le problème devrait être modélisé par un processus décisionnel de Markov non déterministe (MDP-ND). L'agent ne devrait plus apprendre la récompense cumulée V^{π} pour une politique π donnée, telle que définie dans l'équation (2.2), mais la moyenne des récompenses cumulées reçues en suivant

cette politique après un nombre infini d'itérations.

$$V^{\pi}(s_t) = E[\sum_{i=0}^{\infty} \gamma_i r_{t+i}] \tag{2.6}$$

La fonction $Q(s, a)$ devenant :

$$Q(s, a) = E[r(s, a)] + \gamma \sum_{s'} P(s'|s, a) V^*(s') \tag{2.7}$$

où $P(s'|s, a)$ est la probabilité que l'action a mène à s' depuis l'état s. Il est toujours possible de définir la valeur de Q récursivement en faisant la moyenne des différentes valeurs obtenues dans le cas déterministe.

$$Q(s, a) = E[r(s, a)] + \gamma \sum_{s'} P(s'|s, a) \max_{a'} Q(s', a') \tag{2.8}$$

La règle de mise à jour de la table $\hat{\mathrm{Q}}(s, a)$ doit également être adaptée, la précédente ne convergeant plus. Il faut que la nouvelle valeur soit une pondération de l'estimation $\hat{\mathrm{Q}}$ courante et de la valeur du cas déterministe, de telle manière que l'impact du cas déterministe sur la mise à jour de $\hat{\mathrm{Q}}$ diminue au fur et à mesure que le nombre de visite du couple état-action (s,a) augmente. Ainsi,

$$\hat{\mathrm{Q}}_n(s, a) \leftarrow (1 - \alpha_n)\hat{\mathrm{Q}}_{n-1}(s, a) + \alpha_n[r(s, a) + \gamma \max_{a'} \hat{\mathrm{Q}}_{n-1}(s', a')] \tag{2.9}$$

avec

$$\alpha_n = \frac{1}{1 + \mathrm{visits}_n(s, a)} \tag{2.10}$$

où $\mathrm{visits}_n(s, a)$ est le nombre de fois que la paire état-action (s, a) a été visité par l'agent lors de l'itération n. Le facteur α_n décroit bien au fur et à mesure de itérations, l'estimation déterministe prenant donc de moins en moins d'importance. Ainsi, on s'assure que l'algorithme converge vers la

fonction Q.

2.3.5 Les performances de l'apprentissage

Les performances de différents algorithmes d'apprentissage ne sont pas égales. On utilise *la courbe d'apprentissage* de ces algorithmes pour comparer leurs performances sur un ensemble de tests. Cette courbe compare la proportion de choix corrects sur l'ensemble de test par rapport à la taille de cet ensemble. En effet, si la plupart des algorithmes finissent par arriver à un très bon taux de bonnes réponses, la taille de l'échantillon nécessaire avant d'atteindre un tel résultat varie.

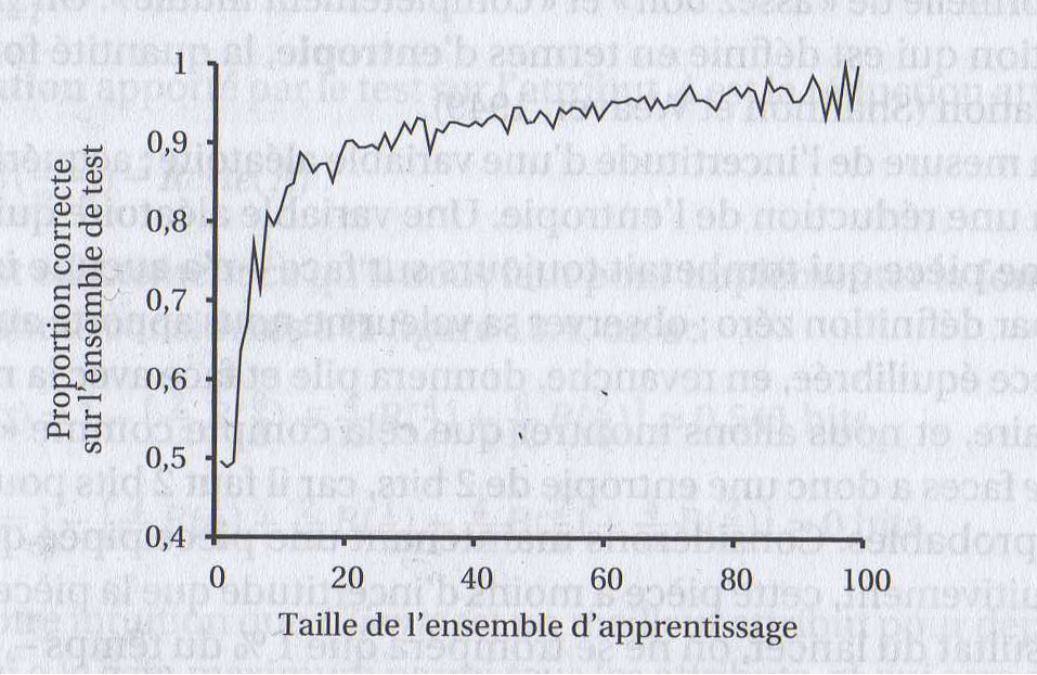

FIGURE 7 – Exemple de courbe d'apprentissage [14]

2.3.6 Le choix des attributs

Lors du choix d'une action à effectuer par un agent, celui-ci évalue une série d'attributs liés à l'environnement ou à lui-même et estime, en fonction de ceux-ci, quelle action est la plus susceptible d'amener un résultat optimal. Pour que la formulation d'hypothèses soit la plus efficace possible, il convient de choisir l'ordre dans lequel les différents attributs perçus sont testés judicieusement. La recherche gloutonne, dont on se sert en apprentissage par arbre de décision, permet de minimiser la profondeur de l'arbre final. Pour ce faire, l'algorithme essaie de maximiser la réduction de l'*entropie*[7] à chaque test. Il s'agit donc de se focaliser sur les attributs les plus discriminants à chaque test. Par exemple, un agent banquier qui s'interroge sur la solvabilité d'un client va préférer effectuer un test sur sa situation financière plutôt que sur la couleur de ses cheveux.

Pour définir des hypothèses correctes, il est important de réduire les attributs en entrée à ceux qui sont pertinents. Supposons qu'un apprenant essaie, par induction, de trouver quelles règles sous-tendent le résultat d'un lancer de dé. S'il accepte en entrée des information, qu'intuitivement on jugerait non pertinentes comme l'heure du lancer, le fait d'avoir croisé les doigts ou la couleur de la chemise du lanceur, il pourrait en conclure des hypothèses incorrectes. Ce phénomène s'appelle le *surapprentissage*. Le surapprentissage devient plus probable lorsque l'espace d'hypothèses et le nombre d'attributs en entrée est grand, mais sa probabilité diminue si l'ensemble d'apprentissage s'agrandit. On juge qu'un critère d'entrée est non pertinent si le *gain d'information* est proche de zéro. Pour déterminer si le gain d'information d'un attribut en entrée est suffisant pour être pertinent, on peut procéder à un test de *signifiance statistique*.

7. L'entropie, telle que définie par Shannon et Weaver en 1949, est la mesure de l'incertitude d'une variable aléatoire

2.4 Systèmes multi-agents

Les systèmes multi-agents sont un outil puissant de résolution de problèmes. En divisant les buts en sous-buts distincts et en affectant des agents de plus faible complexité à ceux-ci, on voit émerger des comportements intelligents de la composition de ces comportements simples. "Nous avons rendu nos agents de plus en plus idiots jusqu'à ce que, finalement, ils rapportent de l'argent" a dit Oren Etzioni au sujet de l'expérience commerciale de NETBOT, Inc.

Les systèmes multi-agents présentent les caractéristiques suivantes :
- Chaque agent possède une information incomplète sur son environnement ;
- Le contrôle des interactions est décentralisé ;
- Les informations sont décentralisées ;
- Les processus sont asynchrones

Les agents composant un système multi-agents peuvent arborer des comportements très différents. Les relations entre ceux-ci peuvent aller de la *coopération* à la *concurrence*. On parle de *concurrence* si, lorsqu' un agent tente de maximiser sa mesure de performance, il minimise celle d'autres agents. Le jeu d'échecs est un bon exemple d'une telle relation entre les agents "Joueurs". Inversement, la relation est une *coopération* si lorsque un agent tente de maximiser sa mesure de performance, il contribue à la maximisation de celle d'autres agents. Plusieurs agents "Robots" qui tentent d'éviter la collision ont donc une relation de coopération. Il est également possible que ces relations soient plus ambigües. Par exemple, dans le *Dilemme du Prisonnier*, les actions de deux agents peuvent soit entrer en concurrence (témoignage) soit, au contraire, être coopératives (silence).

Les systèmes multi-agents coopératifs présentent souvent une forme de communication entre leurs agents pour coordonner leurs actions. La communication entre deux agents est un échange intentionnel d'informations par le

biais de symboles, produits par un agent, perçus par un autre agent et issus d'un système partagé de signes conventionnels. De telles communications permettent d'échanger des informations perçues directement ou inférées, ce qui est un atout considérable lorsque l'environnement n'est que partiellement observable. Une communication peut également être implicite, auquel cas ce sont les modifications de l'environnement opérées par un agent qui permettent à d'autres agents de récupérer de l'information.

Dans un système multi-agents, l'apprentissage est *centralisé* s'il agit comme s'il était seul et considère que les autres agents font partie intégrante de l'environnement. Un tel agent ne requiert aucune interaction explicite avec les autres agents pour son apprentissage. Inversement, un apprentissage *décentralisé* est réparti parmi plusieurs agents, ce qui leur permet, si ces agents communiquent, de pouvoir partager leur apprentissage. Cette notion de centralisation de l'apprentissage est indépendante de l'homogénéité [8], ou l'hétérogénéité de ces différents agents.

Il est possible d'utiliser le Q-Learning de façon hiérarchique en divisant un problème en une multitude de sous-problèmes et autant d'agents chargés de les résoudre. En ajoutant un agent coordinateur chargé d'apprendre la fonction Qs, i où i est l'agent qui doit agir dans l'état s, on réduit considérablement l'espace des actions et donc la complexité de la fonction $Q(s, a)$ traditionnelle. Les résultats obtenus en employant cette méthode sont meilleurs que ceux obtenus avec le Q-Learning traditionnel, notamment lorsque l'espace des actions est grand.

8. Un système multi-agents est *homogène* lorsque les agents qui le composent sont identiques. Sinon, il est hétérogène.

2.5 Systèmes auto-organisés

Plusieurs études [9] montrent que l'activité d'apprentissage collectif permet au système d'être plus performant qu'un système dont les agents apprennent de manière individuelle. L'étude des méthodes d'apprentissage collectif met en évidence deux niveaux d'apprentissage. L'enrichissement du domaine de compétences des agents et l'amélioration des interactions entre agents. L'interaction d'un agent avec d'autres agents permet lui d'apprendre plus et plus vite sur le domaine que s'il est isolé. Les performances du système sont aussi améliorées lorsque les agents se coordonnent et interagissent mieux. Ces différentes études permettent d'identifier trois principales contraintes auxquelles doit se soumettre l'activité d'apprentissage :

- Le processus d'apprentissage du système multi-agents doit être implémenté de manière *distribuée*, c'est-à-dire qu'il soit localisé chez les agents et qu'il n'opère que sur les informations locales à l'agent qu'il améliore. Ainsi, les mécanismes d'apprentissage peuvent différer entre les agents, notamment à cause d'une différence éventuelle de la nature ou de l'implémentation de ces agents. Bien que ceux-ci n'ont pas une vue globale du système, les performances dans un tel contexte n'en sont pas moins excellentes du fait, justement, de l'incomplétude des connaissance.
- Le processus d'apprentissage doit être *asynchrone*. Il est important que l'apprentissage respecte l'autonomie des agents, cette autonomie leur permettant d'avoir un fonctionnement asynchrone de leurs activités de raisonnement, d'action sur l'environnement et de communication entre agents.

9. La liste, non exhaustive, de ces études comprend les travaux de Sian sur son *Multi-Agent Learning Environment* (MALE) en 1991, les modèles de Sekaran et Sen en 1994 sur les performances du *Q-Learning* dans les systèmes multi-agents coopératif ou conflictuels, sans communication explicite, le *système de Mataric* en 1994, la *méthode de Shaw* en 1989 et,enfin, celle de Weiß en 1993.

– L'apprentissage dans un système multi-agents doit être orienté dans une optique de *coopération* entre les agents. Celle-ci est utilisée pour déterminer quelles connaissance relative au domaine apprendre, mais aussi quelles caractéristiques les agents doivent posséder pour atteindre leur but au mieux.

L'apprentissage des connaissances sur le domaine d'application peut être réalisé avec les méthodes classiques ou par le biais d'observations sur les autres agents. L'apprentissage d'une meilleure coordination entre agents, par contre, consiste en la recherche d'une meilleure organisation des agents, c'est-à-dire que chaque agent soit "au bon endroit, au bon moment, dans le chemin du raisonnement" [17]. Pour ce faire, il est nécessaire de supprimer au maximum la propagation inutile d'informations du point de vue du récepteur. Diminuer le nombre d'agents intermédiaires et l'envoi de résultats à un agent qui n'en fera rien permet de diminuer le nombre de communications entre les agents, de diminuer les actions non rentables et donc d'approcher l'organisation optimale en utilisant au mieux les ressources du système.

Si les agents sont dotés de la capacité de détecter une organisation non optimale et de participer à une modification de l'organisation à laquelle il appartient dans le but d'atteindre une organisation optimale, *on assimile l'apprentissage dans un système multi-agents à l'auto-organisation de ce système.* C'est-à-dire la recherche, par le biais de l'apprentissage des agents, de l'organisation optimale de ceux-ci. Cette définition suppose que chaque agent possède un corpus de compétences fixé initialement et invariable, mais aussi qu'il existe une organisation optimale qui peut fournir les comportements souhaités du système.

L'importance que chaque agent attribue à ses performances individuelles et celle qu'il associe à un bonne coordination avec les autres agents sont déterminantes quant à l'efficacité du comportement adopté par ces agents. Le chapitre suivant propose une solution au problème du calibrage des poids

associés à ces importances relatives lorsque les agents ne connaissent pas l'état du système.

Chapitre 3

Étude de cas

Le cas étudié ici est un modèle de système distribué de collecte d'informations représenté par un environnement à explorer et un ensemble d'entités dotées de capacités de déplacement, de senseurs et de communication. Parmi ces entités, certaines produisent des informations incorrectes. Un exemple concret d'un tel cas serait le sondage méthodique d'un terrain inconnu par un ensemble de robots mobiles, dotés de capteurs leur permettant de percevoir le terrain autour d'eux et capables de communiquer entre eux s'ils sont assez proches. Un robot peut produire des informations incorrectes parce que son but diffère ou à cause de capteurs défectueux. Un autre exemple serait la collecte d'informations sur internet où les entités sont confrontées à des sources d'informations mensongères dont il faut déterminer la fiabilité. Les différents agent, qui sont ici des personnes ou des robots qui collectent des informations sur Internet, partagent celles-ci lorsqu'ils entrent en contact avec d'autres agents et doivent apprendre à collecter ces informations au mieux en écartant les sources non fiables.

Chaque entité, pour renforcer la cohérence du système d'informations collectées, maintient deux objets :

- Un réseau de confiance de type *TrustSets* [11] qui lui permet d'estimer la fiabilité des autres entités rencontrées ainsi que celle, déduite, des informations collectées.
- Un ensemble d'informations sur l'environnement comprenant des informations directes (captées par l'entité elle-même) et des informations indirectes (obtenues en communicant avec d'autres entités).

Ces données permettent à chaque entité de choisir le déplacement et les communications qu'elle juge appropriés à l'environnement tel qu'elle le perçoit, ce qui contribue à une auto-organisation du système [12].

L'objectif du cas d'étude est de doter les entités d'une capacité d'apprentissage leur permettant de modifier le poids des différentes règles de comportement de façon à trouver dynamiquement la meilleure stratégie de déplacement et de communication pour ce contexte. Une fois cet apprentissage implémenté, la comparaison des performances du système avec les stratégies optimales théoriques permet de tirer des conclusions sur l'efficacité d'intégrer un processus d'apprentissage supplémentaire dans un système d'auto-organisation.

Dans la suite de ce document, nous présenterons le système en décrivant précisément les différents mécanismes qui entrent en jeu dans la collecte d'information et l'auto-organisation de celui-ci. Ensuite, nous expliquerons quels choix ont été faits quant à la définition du processus d'apprentissage, ainsi que les raisons de ces choix. Enfin, nous présenterons les résultats obtenus par le système doué d'apprentissage, en le comparant notamment aux performances optimales trouvées au niveau théorique grâce à l'algorithme génétique [8].

3.1 Contexte

3.1.1 TrustSets

Dans un système où certains agents ne sont pas fiables et perçoivent des informations erronées, le concept de confiance permet, pour un agent, de décider s'il doit interagir avec un autre. La confiance est la croyance d'un agent en l'honnêteté, la fiabilité et la coopération pour atteindre un même objectif d'un ou plusieurs autres agents. C'est cette notion de confiance qui permet aux agent d'opérer efficacement dans un environnement où ils ne savent pas avec certitude si les autres entités sont coopératives ou même fiables.

La plupart des modèles de confiance calculée est basée sur deux facteurs : la confiance directe, c'est-à-dire celle qui est déduite d'expériences personnelles, et la confiance indirecte, celle qui est acquise par les communications avec d'autres agents. Les modèles de confiance que nous allons utiliser sont les *TrustSets* [11]. Les trustSets se présentent comme suit :

T_{ij} est la confiance d'un agent i envers un agent j. Cette valeur inclut la confiance directe et la confiance indirecte de i pour j. La confiance indirecte est pondérée par la confiance T_{ik} de i en l'agent k qui a coopéré avec j et ensuite partagé ses informations de confiance avec i. Cette confiance T_{ij} est comprise entre 0 et 1, 0 correspondant un manque total de confiance, 1 correspondant à une confiance totale de i en j.

T_{ii} est la confiance d'un agent i en lui-même. Cette notion de confiance en soi est importante dans la définition par un agent de son rôle. Ainsi, un agent qui a une faible confiance en lui même préfèrera relayer des informations obtenues auprès d'agents fiables plutôt qu'utiliser des informations issues de ses senseurs.

Un *TrustSet* est composé d'un *TrustGraph* et d'une *TrustTable*. Le TrustSet

de l'agent X est noté $TS_X = (TG_X, TT_X)$.

Un *TrustGraph* est un graphe orienté qui contient à la fois les confiances directes envers les autres agents et les confiances indirectes en ces derniers. La racine de l'arbre est le propriétaire de celui-ci. Chaque nœud correspond à un agent du système. Chaque flèche connectant deux nœuds signifie que ces deux agents ont interagi. La pondération associée à ces arcs est porteuse d'informations quant à la confiance qu'un agent a envers un autre. La valeur assignée à un arc partant du nœud racine représente donc la *confiance directe*, notée DT_{AB}, qu'a l'agent propriétaire A envers l' agent cible B. La valeur de tout arc dont le nœud d'origine C n'est pas le propriétaire du graphe représente la *confiance indirecte* IT_{CD} de C envers le nœud de destination D. Le TrustGraph de l'agent X est noté $TG_X =< \{\text{Node}_X\}, \{< \text{Arc}_X, \text{Value}_X >\} >$.

Une *TrustTable* associe à chaque nœud B du graphe, c'est-à-dire à chaque agent connu du système, une valeur T_{AB}, *la confiance intrinsèque*. Cette valeur représente la confiance de l'agent propriétaire A en B, en prenant en considération à la fois les confiances directes et indirectes. L'algorithme responsable du calcul de la confiance intrinsèque peut être spécifique à l'agent qui maintient le TrustSet. Le TrustSet de l'agent X est noté $TT_X =< \{< \text{Node}_X, \text{Value}_X >\} >$.

Le TrustGraph représente la partie *publique* du TrustSet, c'est lui qui sera communiqué aux autres agents. La TrustTable pouvant être calculée à partir d'un algorithme spécifique à l'agent, elle représente la partie *privée* du TrustSet et ne sera pas communiquée aux autres agents. Les agents doivent donc recalculer leur TrustTable à partir de la version mise à jour de leur TrustGraph à chaque nouvelle communication de celui-ci.

Lors de *l'initialisation des TrustSets*, chaque agent construit son propre TrustSet. Le TrustGraph est initialisé avec un noeud racine et aucun arc. La TrustTable ne contient qu'une valeur, la confiance qu'a l'agent propriétaire en

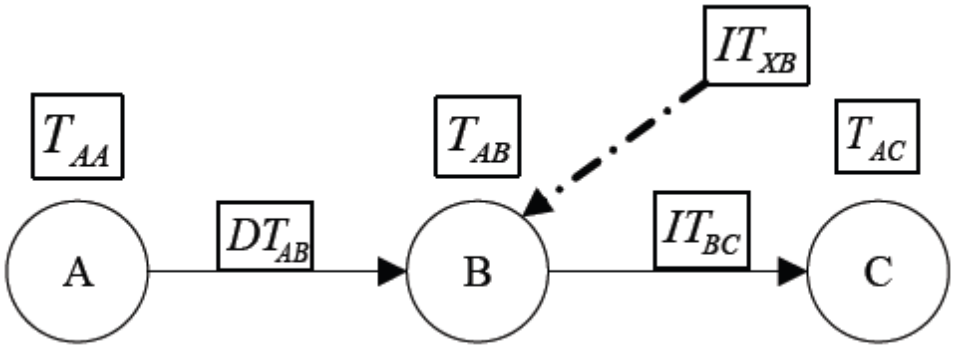

FIGURE 8 – Exemple de TrustGraph [11]

lui-même, qui est initialisée à 1. L'agent n'a, en effet, aucune raison, à priori, de douter de lui-même. Notons que lors de la découverte d'autres agents, la valeur initiale utilisée pour la confiance à accorder à un nouvel agent est fixée à T_{init}.

Lors d'une communication entre un agent A et un agent B, ce premier pourra partager avec B sont TrustGraph. Quand un agent reçoit un TrustGraph, il l'intègre au sien par un mécanisme de *fusion de TrustGraphs* et il met ensuite à jour sa TrustTable.

La *fusion de TrustGraphs* se fait en 3 étapes [1]

1. A calcule sa confiance directe DT_{AB} en B ou, si cette valeur existe déjà, met à jour celle-ci en comparant ses données avec celles reçues.
2. A connecte le TrustGraph de B au sien.
3. A corrige les incohérences présentes dans les chemins partagés par les deux TrustGraphs.

1) Pour calculer la confiance directe d'un agent A en un agent B, on compare ses données D_A avec les données D_B, transmises par B. Pour ce faire, on sélectionne les données comparables, c'est-à-dire celles qui portent sur les

1. :

mêmes objets. Leur nombre est noté β. A calcule le niveau d'incohérence Inc_level entre D_A et D_B grâce à l'équation suivante :

$$\text{Inc\_level} = \frac{\sum_{i \in \{1,\beta\}} \delta(x_i, y_i)}{\beta * \delta_{\text{Max}}\text{info}} \tag{3.1}$$

où $\delta(x, y)$ représente la distance entre les données x de D_A et les données y de D_B. $\delta(x, y) = 0$ signifie que les données x et y sont cohérentes, alors que $\delta(x, y) > 0$ indique qu'il existe un certain degré d'inconsistance entre x et y. δ_{info} est la distance maximale possible entre deux informations incohérentes et (x_i, y_i) est une paire de données comparables.

On fixe un seuil d'incohérence μ en dessous duquel l'agent A augmente sa confiance en B d'un facteur τ^+ et au dessus duquel A diminue sa confiance en B d'un facteur τ^-. Ces deux valeurs sont définies au préalable par les concepteurs du système, en fonction des propriétés qu'ils veulent donner à celui-ci. Ils fixent également deux autres seuils, upp et low, qui déterminent quand la confiance d'un agent envers un autre suffit à considérer celui-ci comme "fiable", "non fiable" ou "en observation". NI est le nombre estimé d'interactions dont a besoin chaque agent pour atteindre son objectif et ρ_{stab} est le taux de stabilisation, compris entre 0 et 1, qui représente la proportion de la simulation après laquelle la classification des agents en agents fiables ou non est considérée finie. $\rho_{\text{stab}} * NI$ représente donc le nombre d'interactions nécessaires pour passer de T_{init} à upp ou low. Il est possible d'en déduire :

$$\tau^+ = \frac{(\text{Upp} - T_{\text{init}})}{\rho_{\text{stab}} * NI} \tag{3.2}$$

et

$$\tau^- = \frac{(T_{\text{init}} - \text{Low})}{\rho_{\text{stab}} * NI} \tag{3.3}$$

2) La fusion de deux TrustGraphs est constituée par l'union de ceux-ci. Une fois celle-ci effectuée, l'arc AB est ajouté et les arcs à destination de

A sont enlevés pour éviter les cycles. Bien que ces arcs soient supprimés du graphe, la valeur de ceux-ci est à prendre en compte dans le calcul des confiances intrinsèques de la TrustTable. Nous avons donc :

$$\mathrm{Arc}_* = \{AB\} \cup \mathrm{Arc}_A \cup \mathrm{Arc}_B \backslash \{BA, \ldots, XA\} \tag{3.4}$$

3) Il y a incohérence dans les poids associés aux arcs d'un TrustGraph lorsque cet arc apparait dans les deux TrustGraph initiaux et que celui-ci a une pondération différente dans chacun d'eux. Ce cas apparait typiquement lorsque les agents A et B rencontrent l'agent C à deux moments différents. Pour éviter cette incohérence, une nouvelle valeur pour cet arc est calculée comme suit : Pour chaque arc XY commun à TG_A et TG_B, CTR_{XY_A} est la valeur de celui ci dans TG_A et CTR_{XY_A} est sa valeur dans TG_B. On construit les ensembles Path_A et Path_B composés, respectivement, de tous les plus courts chemins de TG_A et TG_B qui contiennent l'arc XY. La valeur $CTR_{XY_{AB}}$ qui résulte de la fusion des graphes est calculée par l'équation :

$$CTR_{XY_{AB}} = \frac{\sum_{X \in \mathrm{Path}_A} c_{AX} * CTR_{XY_A} + \sum_{Y \in \mathrm{Path}_B} c_{AY} * CTR_{XY_B}}{\sum_{X \in \mathrm{Path}_A} c_{AX} + \sum_{Y \in \mathrm{Path}_B} c_{AY}} \tag{3.5}$$

Une fois le TrustGraph redéfini, il est nécessaire de mettre à jour la TrustTable pour toutes les valeurs nouvelles ou influencées par les nouveaux éléments, ceux-ci incluant les éléments supprimés lors de l'étape 2. Cette mise à jour demande de calculer la confiance intrinsèque de chaque nouveau nœud transmis par B, mais également celle de tous les nœuds impactés par ceux-ci, à l'exception d'eux-mêmes. Le calcule de la confiance intrinsèque d'un agent A envers un agent X se calcule à l'aide de la formule suivante :

$$C_{AX} = \frac{C_{AA} * CT_{AX} + \sum_{Y \in \mathrm{AGENTS}} (C_{AY} * CTR_{XY})}{C_{AA} + \sum_{Y \in \mathrm{AGENTS}} C_{AY}} \tag{3.6}$$

La maintenance d'un TrustSet par un agent permet à celui-ci de modifier sa stratégie de déplacement et de communication en fonction de la confiance qu'il accorde aux autres agents. Il peut, par exemple, choisir d'ignorer totalement les informations communiquées par des agents jugés non fiables, c'est-à-dire ceux dont la valeur de confiance T est tombée en-dessous du seuil low et ainsi éviter la propagation d'informations erronées parmi les agents du système. Les TrustSets permettent non seulement de maintenir une information sur la confiance à accorder aux agents, mais aussi de déterminer la fiabilité des informations collectées en fonction de la quantité et du niveau de confiance des sources de ces informations.

3.1.2 Modèle d'auto-organisation

Le modèle d'auto-organisation [12] est basé sur deux composants distincts : celui lié à la collecte directe d'informations et celui qui supervise les communications. Les règles de comportement peuvent être divisées en 3 catégories : les règles de *déploiement*, les règles de *communication* et les règles de *rétro-action* entre le déploiement et la communication.

Déploiement

Le but des agents étant de collecter des informations sur l'environnement, ceux-ci maintiennent de l'information sur la zone explorée, c'est-à-dire toutes les positions visitées par eux-mêmes et les autres agents. Une position est considérée visitée par un agent i si elle est passée à portée r_i des senseurs de cet agent. Chaque fois que l'agent a atteint la position désirée, il sélectionne la position suivante à atteindre en considérant comme candidates toutes les positions X qui sont au bord de la zone déjà explorée par cet agent. Le choix d'une position à atteindre parmi celles-ci suit les règles de déplacement suivantes :

Règle de mouvement influencé par la distance. Cette règle indique que les agents ont tendance à choisir la position candidate la plus proche de leur position actuelle. L'agent calcule la distance minimale à parcourir $D(i, X)$ pour atteindre chaque position X. Le gain estimé, pour l'agent i, de choisir X comme prochaine position en fonction de sa distance à la position actuelle est calculé comme suit :

$$g_{\text{dist}}(i, X) = \frac{1}{\sqrt[n]{D(i, X)}} \tag{3.7}$$

où n est une constante plus grande où égale à 1. Plus n est grande, plus proche est g_{dist} d'une fonction linéaire.

Règle de mouvement influencé par la quantité d'informations. Cette règle indique que les agents ont tendance à choisir la position candidate sur laquelle il possède le moins d'informations, ce qui évite notamment la concentration d'agents aux même endroits. Chaque agent i connait la quantité d'informations $N(i, X)$ qu'il a collectée pour toutes les positions X envisagées. Le gain estimé, pour l'agent i, de choisir X comme prochaine position en fonction de la quantité d'informations déjà collectées sur celle-ci est calculé comme suit :

$$g_{\text{inf}}(i, X) = \frac{1}{\sqrt[n]{N(i, X) + 1}} \tag{3.8}$$

où n est une constante plus grande où égale à 1. Plus n est grande, plus proche est g_{dist} d'une fonction linéaire.

Règle de mouvement influencé par la qualité d'informations. Cette règle indique que les agents ont tendance à choisir la position candidate sur laquelle ils possèdent les informations les moins fiables ce qui permet aux agents de confirmer ou d'infirmer les informations peu fiables dont ils disposent. Chaque agent i connait la fiabilité $R(i, X)$ associée aux informations sur chaque position X envisagée. Cette fiabilité des infor-

mations dépend de la fiabilité des sources de celle-ci. Le gain estimé, pour l'agent i, de choisir X comme prochaine position en fonction de la fiabilité des informations déjà collectées sur celle-ci est calculé comme suit :

$$g_{\text{reli}}(i, X) = 1 - R(i, X) \tag{3.9}$$

Règle de mouvement influencé par les autres agents. Cette règle indique que les agents ont tendance à choisir la position candidate qui leur permet de se rapprocher des autres agents fiables et de s'éloigner des agents non fiables, de telle manière qu'ils aient accès à des informations fiables. Pour ce faire, l'agent i doit calculer la force d'attraction ou de répulsion entre lui-même et chaque agent j présent dans sa zone de perception $A_{\text{per}}(i)$. Cette force est calculée par :

$$\vec{F_{ij}} = \frac{T_{ij} - 0,5}{0,5} * \frac{\vec{V_{ij}}}{D(i,j)} \tag{3.10}$$

où $\vec{V_{ij}}$ est le vecteur partant de la position de i vers la position de j et $D(i, j)$ la distance entre i et j. On calcule $\vec{F_i} = \sum_{k \in A_{\text{per}}(i)} \vec{F_{ik}}$, le vecteur résultant des forces d'attraction et de répulsion de tous les agents dans $A_{\text{per}}(i)$. Le gain estimé, pour l'agent i, de choisir X comme prochaine position en fonction de la proximité d'agents fiables est calculé comme suit :

$$g_{\text{agent}}(i, X) = cos(\vec{F_i}, \vec{V_{iX}}) * |\vec{F_i}| \tag{3.11}$$

Le gain total estimé, pour l'agent i, de choisir X comme prochaine position utilise une combinaison de ces 4 mesures normalisées et associées à un

poids :

$$g(i,X) = w_1 * \frac{g_{\text{dist}}(i,X)}{\max_{Z \in P_{\text{front}}(i)} g_{\text{dist}}(i,Z)} + w_2 * \frac{g_{\text{info}}(i,X)}{\max_{Z \in P_{\text{front}}(i)} g_{\text{info}}(i,Z)} + w_3 * \frac{g_{\text{reli}}(i,X)}{\max_{Z \in P_{\text{front}}(i)} g_{\text{reli}}(i,Z)} + w_4 * \frac{g_{\text{agent}}(i,X)}{\max_{Z \in P_{\text{front}}(i)} g_{\text{agent}}(i,Z)} \quad (3.12)$$

où $P_{\text{front}}(i)$ est l'ensemble de toutes les positions candidates considérées pour l'agent i. w_1, w_2, w_3 et w_4 sont des poids, compris entre 0 et 1, représentant l'importance de chaque règle de comportement. Ceux-ci doivent toujours respecter la propriété suivante :

$$w_1 + w_2 = w_3 + w_4 = 1 \quad (3.13)$$

Le calibrage de ces poids est fait lors de la création du système et c'est lui qui détermine la politique de déplacement des agents.

La position X sélectionnée pour le déplacement suivant est celle qui maximise le gain $g(i,X)$. Si plusieurs positions répondent à cette condition et ont donc un gain estimé équivalent, la position à atteindre est choisie aléatoirement entre celles-ci.

Communication

Une autre manière, pour les agents, de collecter des informations sur l'environnement est de communiquer avec les autres agents pour échanger leurs connaissances de celui-ci. Lorsqu'au moins un autre agent est à portée de communication, les agents considèrent s'ils doivent communiquer avec celui-ci et, le cas échéant, quelles informations doivent être partagées. Voici les règles de communication qui définissent le processus de décision :

Règle de tendance générale à la communication. Cette règle sert à décider si l'agent va communiquer avec un autre agent à l'instant présent. Plus l'agent est loin de la position cible vers laquelle il se dirige, moins l'agent communiquera. Cette probabilité de communiquer se calcule comme suit :

$$p(i) = \frac{1}{\sqrt[n]{D(X_{\text{curr}}, X\text{next}}} \tag{3.14}$$

où $D(X_{\text{curr}}, X\text{next})$ est la distance entre la position actuelle de l'agent i et la position cible vers laquelle il se dirige. n est une constante plus grande où égale à 1. Plus n est grande, plus proche est $p(i)$ d'une fonction linéaire. Cette équation indique que plus l'agent est proche de sa destination, plus il a de chances de communiquer. En effet, l'agent a plus de chances de collecter des informations fiables sur la destination au plus il se rapproche de celle-ci. Si l'agent choisit de communiquer, les trois règles suivantes permettent de définir avec quel autre agent il communiquera.

Règle de communication basée sur la fiabilité du partenaire. Cette règle indique la tendance qu'à un agent i à choisir un agent j, à portée de communication, comme partenaire de la communication en fonction de la confiance qu'il accorde à celui-ci. La probabilité qu'il le fasse se calcule comme suit :

$$p_{\text{reli}}(i, j) = T_{ij} \tag{3.15}$$

où T_{ij} est la confiance de l'agent i en l'agent j. Cette équation indique que plus l'agent j est fiable, plus l'agent i sera intéressé de communiquer avec lui.

Règle de communication conditionnée par la durée de déconnexion. L'agent i tend à communiquer plus avec l'agent j au plus leur dernière communication remonte à longtemps. Cette tendance se calcule comme suit :

$$p_{\text{dur}}(i, j) = \min\left(\frac{\Delta t_{ij}}{\text{average}_{k \in A_{\text{com}}(i)}(\Delta' t_{ik})}, 1\right) \tag{3.16}$$

où Δt_{ij} est la quantité de temps écoulée depuis la dernière communication entre i et j, $\Delta' t_{ik}$) est le temps écoulé entre les deux dernières communications entre i et k. $A_{\text{com}}(i)$ est l'ensemble des agents ayant déjà communiqué avec i.

Règle de communication conditionnée par la nouveauté des informations reçues. L'agent i tend à communiquer avec les agents qui envoient de nouvelles informations. Cette tendance se traduit par l'équation suivante :

$$p_{\text{new}}(i,j) = \frac{1}{2}\left(\frac{n_j}{\max_{k \in A_{\text{com}}(i)}(n_k)} + \frac{n_j}{n'_j}\right) \tag{3.17}$$

où n_k est le nombre de nouvelles données reçues de l'agent k, n'_j est le nombre total de données reçues de j et $A_{\text{com}}(i)$ est l'ensemble des agents ayant communiqué avec i.

La combinaison de ces trois règles de communication détermine la tendance qu'a un agent i à communiquer avec l'agent j. Cette tendance est formalisée comme suit :

$$p(i,j) = w_5 * p_{\text{reli}}(i,j) + w_6 * p_{\text{dur}}(i,j) + w_7 * p_{\text{new}}(i,j) \tag{3.18}$$

où w_5, w_6 et w_7 sont des poids compris entre 0 et 1, et qui respectent la propriété suivante :

$$w_5 + w_6 + w_7 = 1 \tag{3.19}$$

Ces poids sont définis lors de la création du système et calibrent l'importance donnée à chaque règle de communication dans le processus de décision. Si l'agent i décide de communiquer avec j, les règles suivantes détermineront quelles informations sont communiquées.

Règle de communication d'informations directes. L'agent i tend à envoyer les informations directes au plus elles sont fiables. Au plus les récepteurs ont confiance en l'émetteur, au plus ce dernier envoie ses informations

directes. Cette tendance se traduit par l'équation suivante :

$$p(i, j, I_{\text{dir}}(i, X)) = T_{ii} \tag{3.20}$$

où $I_{\text{dir}}(i, X)$ est l'information directe de l'agent i sur la position X et T_{ii} est la confiance de i en lui même.

Règle de communication d'informations indirectes. L'agent i tend à envoyer les informations indirectes au plus elles sont fiables. Ainsi, l'agent contribue à la qualité du système communicant. Cette tendance se traduit par l'équation suivante :

$$p(i, j, I_{\text{ind}}(i, X)) = \text{reliability}(I_{\text{ind}}(i, X)) \tag{3.21}$$

où $I_{\text{ind}}(i, X)$ est l'information indirecte que i possède sur la position X et reliability$(I_{\text{ind}}(i, X))$ est la fiabilité des données indirectes sur la position X reçues par i. Cette fiabilité est calculée sur base du nombre d'agents qui ont communiqué la même valeur pour cette information, ainsi que la fiabilité de ceux-ci.

Contrôle

Les deux ensembles de règles sus-cités ne sont pas indépendants. Le comportement de collecte d'informations est influencé par le comportement de communication et inversement. Pour permettre une co-évolution de ces deux composants, nous définissons un composant de *contrôle* doté de deux règles de comportement supplémentaires et qui influenceront le résultat de l'équation (3.11) :

Règle de renforcement/diversification influencée par la précision des informations reçues. Au plus les informations envoyées par j sont précises, au plus le récepteur i aura tendance à se rapprocher de l'émetteur. Inversement, si les informations transmises par j sont imprécises, le

récepteur aura davantage tendance à s'éloigner de celui-ci. Le niveau de fiabilité d'un agent j en fonction de toutes les informations envoyées à i est la confiance que celui-ci a en j. En revanche, le niveau de fiabilité d'un agent j en fonction de la dernière communication effectuée vers l'agent i se calcule comme suit :

$$\text{level}_{\text{reli}}(i,j) = \frac{\sum_1^n C(I_{\text{dir}}(i,X), I_{\text{dir}}(j,X))}{n} \tag{3.22}$$

où n est le nombre de données comparables dans cette communication, $C(I_{\text{dir}}(i,X), I_{\text{dir}}(j,X))$ est le résultat de la comparaison entre les données directes de i au sujet de la position X et des données directes de j sur cette même position., calculé comme suit :

$$C(I_{\text{dir}}(i,X), I_{\text{dir}}(j,X)) = \begin{cases} 1 & \text{si} I_{\text{dir}}(i,X) = I_{\text{dir}}(j,X) \\ -1 & \text{si} I_{\text{dir}}(i,X) \neq I_{\text{dir}}(j,X) \end{cases} \tag{3.23}$$

Règle de renforcement/diversification du déplacement influencé par la nouveauté des informations échangées. Au plus les données reçues par un agent sont nouvelles ou renouvelées, au plus celui-ci a tendance à se rapprocher de l'émetteur de ces données. Inversement, si les données reçues par l'agent sont peu renouvelées, celui-ci aura tendance à s'éloigner de l'émetteur. Cette tendance est calculée comme suit :

$$\text{level}_{\text{new}}(i,j) = \frac{1}{2}\left(\frac{n_j}{\max_{k \in A^t_{\text{com}}(i)}(nk)} + \frac{n_j}{n'j}\right) - 0,5 \tag{3.24}$$

où n_k est le nombre de nouvelles données reçues depuis l'agent k, $n'j$ est le nombre total d'informations reçues depuis l'agent j et $A_{\text{com}}(i)$ est l'ensemble des agents ayant communiqué avec i.

Ces différentes règles permettent de reformuler l'équation qui donne la force d'attraction d'un agent i envers un agent j (3.10) comme suit :

$$\vec{F_{ij}} = (w_8 * \frac{T_{ij} - 0,5}{0,5} + w_9 * \text{level}_{\text{reli}}(i,j) + w_{10} * \text{level}_{\text{new}}(i,j)) * \frac{\vec{V_{ij}}}{D(i,j)} \tag{3.25}$$

où w_8, w_9 et w_{10} sont des poids positif qui doivent respecter la règle suivante :

$$w_8 + w_9 + w_{10} = 1 \tag{3.26}$$

Ces poids, définis à la création du système, déterminent l'importance de chaque règle dans la définition du *comportement de contrôle* de l'agent.

En bref, ces règles représentent l'influence des communications sur le mouvement des agents.

Auto-organisation

Dans ce modèle, chaque agent divise les autres agents en trois classes différentes : les agents fiables, non fiables et en cours d'évaluation. Ces agents agissant différemment en fonction de ces valeurs de confiance, ils ont tendance à s'organiser de manière à atteindre leur but plus efficacement en suivant les différentes règles qui régissent leur comportement. En dissociant les "rôles" des agents en fonction de la confiance qui leur est accordée, ceux ci s'auto-organisent en utilisant la fiabilité comme critère discriminant. L'impact de cette organisation sur les performances du système étant fortement influencé par le calibrage du poids des règles de comportement, il est important que ceux-ci soient choisis judicieusement. Les valeurs optimales de ces poids peuvent être déterminées, pour un système donné, à l'aide d'algorithme d'optimisation tels que l'algorithme génétique. Ces valeurs sont cependant des valeurs théoriques dont la découverte nécessite d'avoir toutes les informations sur le système, ce dont ne disposent pas les agents dans la pratique.

Le chapitre suivant tente de répondre à cette question : "Comment calibrer les poids des règles de comportement sans connaitre l'état du système ?"

Chapitre 4

Problème

Les performances de ce modèle d'auto-organisation dépendent en grosse partie du calibrage des différents poids associés aux règles de comportement (w_1-w_{10}). Fixés lors de la création du système, ceux-ci doivent, pour maximiser l'efficacité du système, avoir une valeur adaptée aux autres paramètres du système, comme la proportion d'agents non fiables ou la portée de communication. Une application de l'algorithme génétique à la création de tels systèmes permet d'optimiser le calibrage de ces variables pour un système précis. Cette optimisation n'est cependant utilisable que dans des systèmes où la proportion d'agents défectueux est connue à l'avance, ce qui ne correspond pas à une situation réelle. Dans le but d'optimiser les performances du système en pratique, nous allons doter les agents d'une capacité d'apprentissage leur permettant de modifier les poids associés aux règles au fur et à mesure que leur représentation de leur environnement change et, ainsi, trouver la meilleure stratégie de déplacement et de communication pour ce contexte.

4.1 GAMA

Pour illustrer cette recherche, l'environnement de modélisation et de simulation de systèmes multi-agents *GAMA* [6], développé par l'équipe de recherche MSI de l'Institut de la Francophonie pour l'Informatique de Hanoï depuis 2007, est utilisé. Celui-ci a été développé en Java, est sous licence GPL. La version sur laquelle est développé le modèle étudié est la 1.3 [1]. Celle-ci utilise un langage de modélisation basé du XML qui lui est propre, GAML, qui est remplacé à partir de la version 1.4 de la plateforme. GAMA présente les caractéristiques suivantes :

- GAMA permet l'utilisation de données pour des Systèmes d'Information Géographiques (SIG) complexes.
- GAMA permet de lancer des simulations de systèmes multi-agents composés d'un grand nombre d'agents. Ce nombre peut atteindre plusieurs millions d'entités.
- GAMA permet d'enchainer plusieurs simulations dans lesquelles un paramètre unique change progressivement (mode "batch"). Ce mode de simulation permet d'observer l'influence du dit paramètre sur les résultats de la simulation.
- GAMA permet aux utilisateurs d'interagir les agents dans une simulation en cours.
- GAMA permet d'ajouter des architectures d'agents et des fonctions aux librairies natives.
- GAMA est multi-plateforme (disponible sous Windows, Linux et MacOS à l'heure actuelle).
- GAMA utilise une interface utilisateur inspirée de la plateforme Eclipse.

Voici quelques exemples de modèles réalisés avec GAMA :

1. La dernière version, à ce jour, est la 1.6. Celle-ci est disponible en téléchargement libre sur le site https ://code.google.com/p/gama-platform/

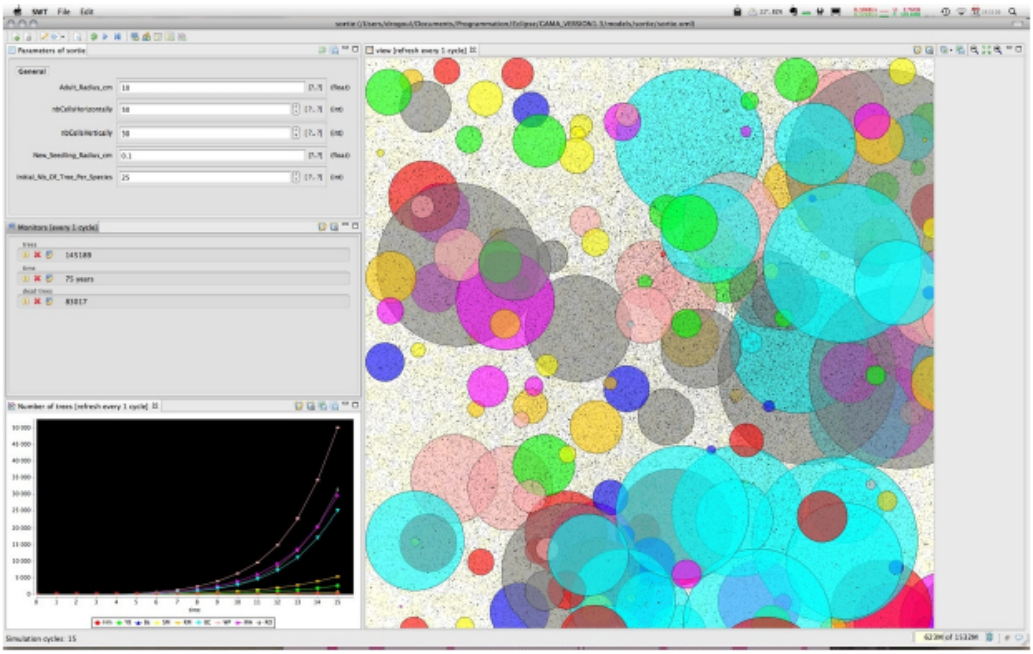

FIGURE 9 – Modèle théorique de développement des forêts, développé et maintenu par Guillaume Chérel sur GAMA

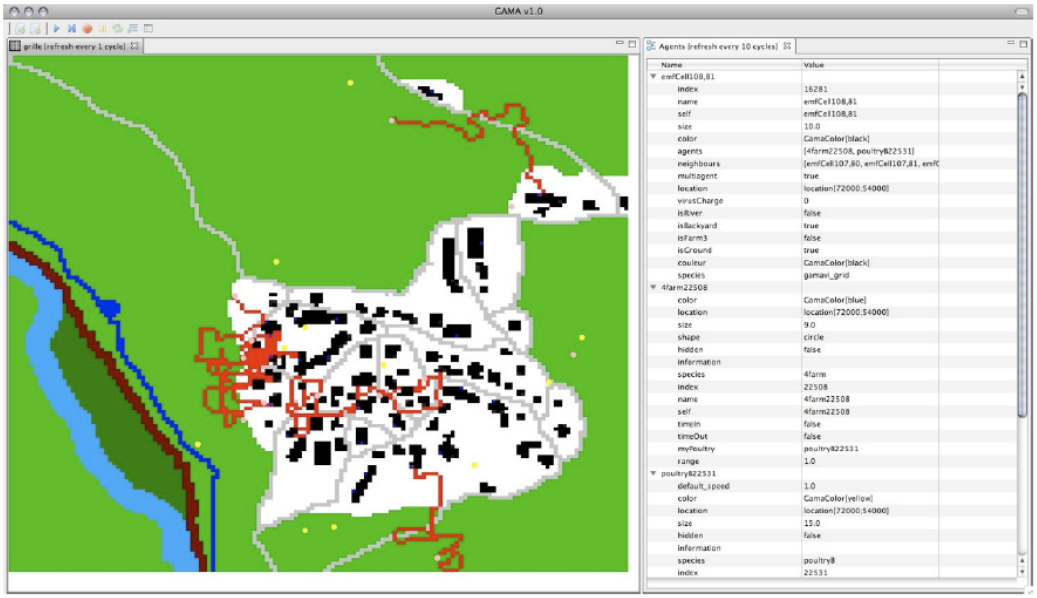

FIGURE 10 – GAMAVI : Modèle appliqué simulant la propagation du virus H5N1 parmi une population de volailles à l'échelle d'un village en fonction de l'environnement. [7]

Un modèle GAMA est composé en 4 parties :

La section *Global* permet de déclarer les variables globales, les paramètres que l'utilisateur doit entrer avant toute simulation et initialise celles-ci.

La section *Entities* décrit les différentes espèces d'entités qui interviennent dans les systèmes multi-agents modélisés. Les instances de cette espèce possèdent des attributs et des compétences propres à celle-ci. C'est également ici que sont déclarés les réflexes effectués par les agents à chaque cycle de la simulation.

La section *Environment* décrit les propriétés de l'environnement dans lequel évoluent les agents.

La section *Output* décrit les informations sortantes désirées par le créateur du modèle. Celles-ci peuvent prendre la forme de texte, de graphiques, d'animations, de fichiers et même de données SIG pouvant être utilisées

par un autre modèle.

La figure 11 donne un exemple de code pour modèle GAMA extrait de la version modifiée de Danger Mapping, écrit en GAML tel qu'il était défini jusqu'à la version 1.3. Celui-ci décrit l'un des réflexes appartenant au processus d'apprentissage des agents, la mise à jour des valeurs-Q avant chaque nouvelle action sur les poids des règles.

```
<reflex name="update_q_value" when="time &gt;=1">
    <do action="write">
        <arg name="message"
        value=" 'beginning updating q_value'" />
    </do>
    <if condition="self.nb_informations_this_step=0">
        <set name="self.nb_informations_this_step" value="1"/>
    </if>
    <create species="q_value_element" number="1" return="new_q_value_movement">
        <set var="mean_divider" value="myself.old_q_value_movement.mean_divider +1" />
        <set var="q_value" value="((myself.old_q_value_movement.q_value * myself.old_q_value_mover
    </create>
    <add item="new_q_value_movement" to="(((((movement_decision_tree) @ old_w_dis) @ old_w_inf) @

    <create species="q_value_element" number="1" return="new_q_value_communication">
        <set var="mean_divider" value="myself.old_q_value_communication.mean_divider +1" />
        <set var="q_value" value="((myself.old_q_value_communication.q_value * myself.old_q_value_
    </create>
    <add item="new_q_value_communication" to="(((((movement_decision_tree) @ old_w_dis) @ old_w_in

    <create species="q_value_element" number="1" return="new_q_value_control">
        <set var="mean_divider" value="myself.old_q_value_control.mean_divider +1" />
        <set var="q_value" value="((myself.old_q_value_control.q_value * myself.old_q_value_contro
    </create>
    <add item="new_q_value_control" to="(((((movement_decision_tree) @ old_w_dis) @ old_w_inf) @ o
                <do action="write">
        <arg name="message"
        value=" 'end updating q_value'" />
    </do>
</reflex>
```

FIGURE 11 – Extrait du code de la version modifiée de Danger Mapping pour GAMA

4.2 Danger Mapping

Le modèle utilisé ici est *Danger Mapping*, implémenté par Quang Anh Nguyen Vu pour servir d'exemple à l'utilisation du modèle d'auto-organisation décrit précédemment. [12]

Dans *Danger Mapping*, le système est composé d'une nuée de robots mobiles et localisés patrouillant un terrain inconnu. L'objectif, pour chaque robot, est de construire la carte la plus précise et la plus fiable du terrain en utilisant aussi peu de ressources que possible. Chacun de ces robots est doté de senseurs qui lui permettent de détecter directement la nature de la zone environnante mais certains de ces robots ont des capteurs défectueux. Ils peuvent également communiquer avec les autres robots pour échanger leurs connaissances sur leur environnement, c'est-à-dire la carte de celui-ci, ou sur leur confiance envers les autres agents.

Lors de la création du système, il est nécessaire d'introduire une série de paramètres qui seront fixés pour toute la durée de la simulation :

La longueur et la largeur de la carte.

L'utilisation d'un environnement aléatoire ou non. Il est en effet possible d'utiliser des cartes pré-établies en utilisant les données SIG adaptées.

La description des obstacles. La taille des obstacles à indiquer lors de la cartographie, leur nombre et le nombre de différents niveaux de danger sont autant de paramètres nécessaires à la création du système.

Le nombre de robots fiables et le nombre de robots non fiables.

Les paramètres sur les communications des robots. C'est-à-dire non seulement la distance de communication mais aussi les poids attachés aux règles de comportement sur ces communications.

Les paramètres sur les mouvements des robots, à savoir leur vitesse, la distance minimale entre deux points frontières candidats, ainsi que les poids attachés aux règles de comportement sur les déplacements.

La distance et la précision de perception des robots.

Les paramètres liés au calcul de la confiance. Le nombre minimal d'interactions nécessaires à ce calcul, le taux de stabilisation, le niveau d'incohérence accepté de l'information, ainsi que le choix d'attacher une valeur de fiabilité aux informations sont autant de paramètres nécessaires.

Les seuils de confiance envers les agents.

Les seuils de confiance envers les informations.

Les systèmes décrits par le modèle Danger Mapping donnent donc une valeur fixe pour les poids des règles de comportement. De plus, ces valeurs sont communes à tous les agents qui composent ces systèmes, ce qui a pour effet de diminuer l'indépendance relative de ces agents et de mettre l'emphase sur l'influence du TrustSet de chaque agent sur son comportement.

Voici un exemple d'utilisation du modèle *Danger Mapping* sur GAMAv1.3. Dans les impressions d'écran suivantes, les points de couleur représentent les agents, le bleu correspondant aux agents fiables et le rouge correspondant aux agents menteurs ou corrompus. Deux cercles entourent les agents. Le cercle jaune délimite la portée de communication alors que le cercle noir représente la portée de perception. Trois nombres sont attachés aux agents. Le nombre vert est l'identifiant de l'agent, le bleu est le nombre d'obstacles détectés alors que le rouge correspond au nombre d'obstacles dont l'estimation du niveau de danger est fiable. Les différents carrés de couleur représentent les dangers présents sur la carte, que les agents doivent cartographier et évaluer. Les carrés jaunes représentent les positions candidates pour le déplacement d'un agent. Enfin la zone verte est la zone explorée par les agents. La couleur de cette zone explorée devient plus claire à mesure que la fiabilité des informations sur celle-ci augmente. Lorsqu'une zone est considérée comme totalement couverte, c'est-à-dire lorsque la fiabilité des informations sur celle-ci dépasse un niveau donné, celle-ci devient blanche. Lorsque toute la carte devient blanche, la simulation s'arrête.

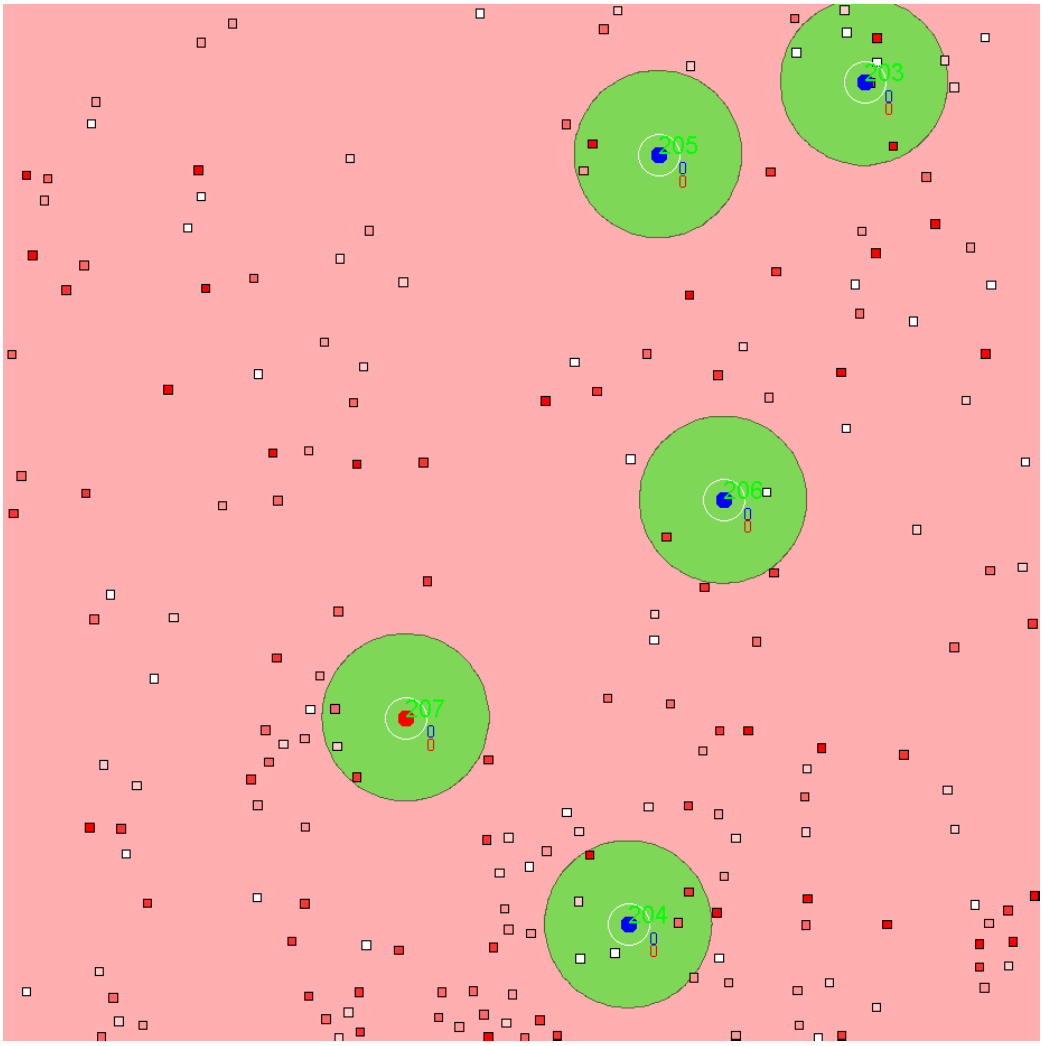

FIGURE 12 – Système multi-agents modélisé par Danger Mapping après l'initialisation, GAMA

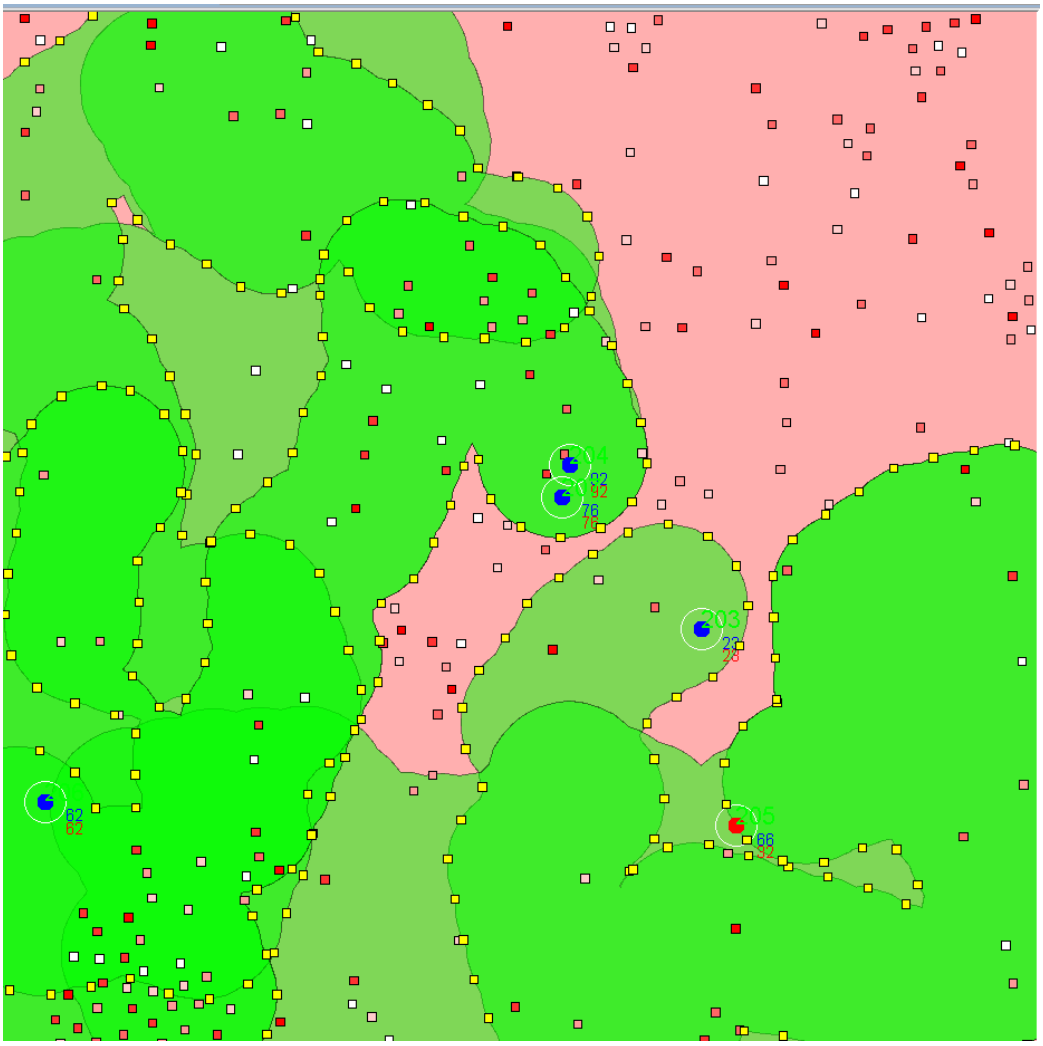

FIGURE 13 – Système multi-agents modélisé par Danger Mapping pendant la simulation, GAMA

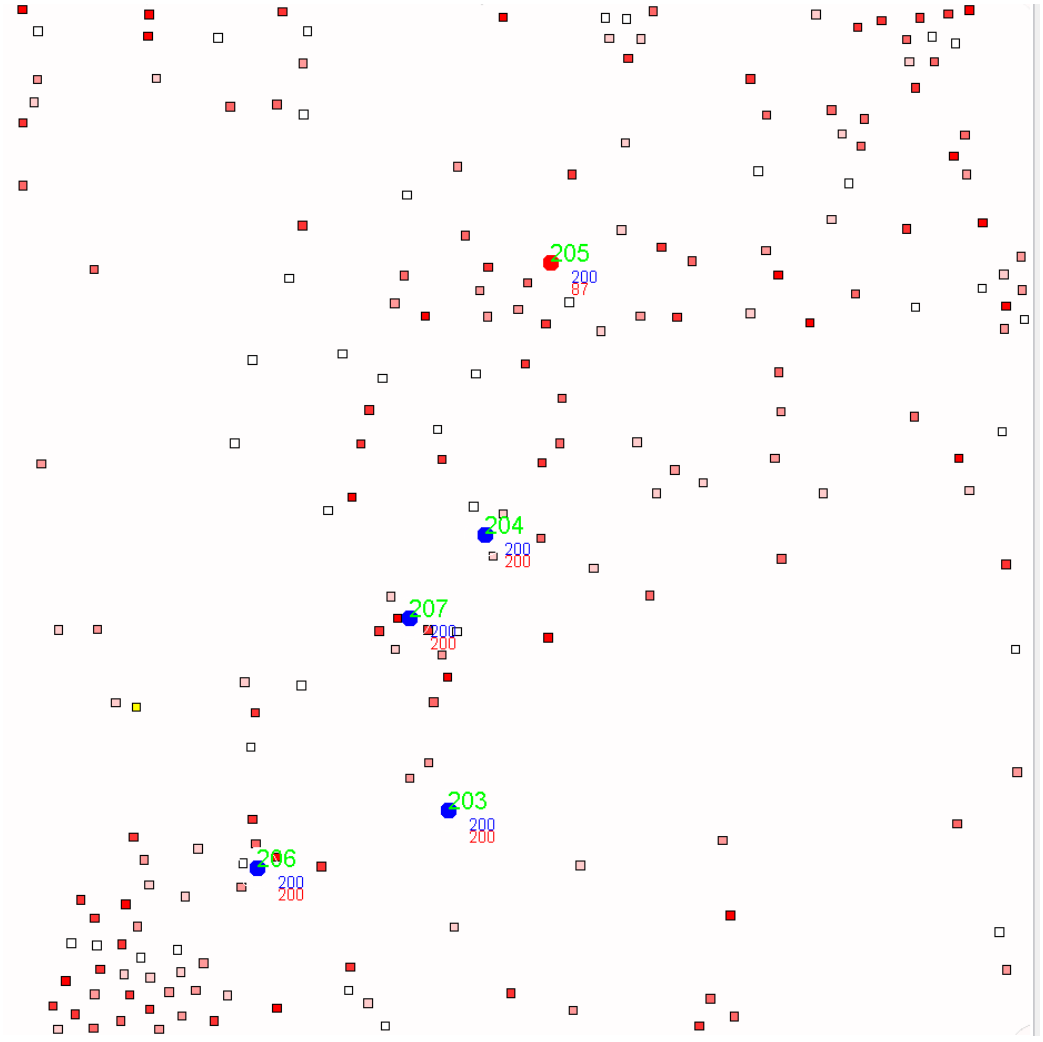

FIGURE 14 – Système multi-agents modélisé par Danger Mapping après la simulation, GAMA

Chapitre 5

Solution

Ce chapitre tente de trouver une réponse au problème du calibrage des poids des règles de comportement en intégrant à chaque agent un mécanisme d'apprentissage sur ceux-ci. Les différents choix de conception et d'implémentation de cette solution au modèle de systèmes multi-agents Danger Mapping y sont décrit et expliqués. Cette solution n'est en aucun cas optimale mais l'analyse des résultats produits permet de tirer des conclusions sur l'intérêt d'ajouter une forme d'apprentissage sur les poids associés aux comportements, et particulièrement à l'importance accordée à la bonne coordination des agents du systèmes par rapport aux performances individuelles de ces agents.

Bien que les poids optimaux associés aux règles, pour un contexte donné, aient déjà été recherchés [8], ces valeurs théoriques n'ont d'intérêt que si les agents connaissent l'état du système. En pratique, cette condition n'est pas remplie et les agents doivent deviner l'état de celui-ci sur base de leur expérience. Ainsi, par exemple, le taux d'agents défectueux est continuellement estimé par chaque robot, en se basant sur la liste des agents connus et les valeurs de confiance pour ceux-ci. L'intégration d'un mécanisme d'apprentissage sur les poids des règles de comportement permet aux agents de

calibrer ceux-ci en fonction de la représentation interne qu'ils ont de l'environnement et des résultats obtenus précédemment.

Pour choisir efficacement comment ajouter cet apprentissage au système multi-agents, il est nécessaire de caractériser celui-ci. L'environnement est :

Partiellement observable. Un agent ne perçoit qu'une fraction de l'environnement qui l'entoure, déterminée par la portée de ses senseurs. De plus, il ne sait pas quels agents sont fiables ou non.

Multi-agents.

Non déterministe. Il n'est pas possible, à partir de l'état courant du système et de l'action entreprise, de déterminer quel sera l'état résultant de cette action, ni même de déterminer les probabilités d'arriver dans cet état.

Séquentiel. Les actions de déplacement ne sont pas exécutées instantanément. Celles-ci déterminent en grande partie les actions de communication subséquentes, celles-ci ayant lieu uniquement avec les autres agents à portée de communication au cours du trajet.

Dynamique. L'environnement évolue même sans actions de l'agent. En effet, les actions des autres agents influencent celui-ci, changeant les valeurs de confiance ou les connaissances des dits agents.

Continu. L'ensemble des états de l'environnement et celui des agents sont des ensembles continus. L'espace d'état lié au temps, lui, est discret. En effet, chaque itération correspond à un déplacement dont la longueur dépend de la vitesse de déplacement du robot et, éventuellement, à une communication vers les agents à portée.

Inconnu. Les agents ne connaissent aucune règle leur permettant de connaitre le fonctionnement de l'environnement.

Les systèmes multi-agents définis par le modèle *Danger mapping* présentent une infinité d'états et un grand nombre d'actions possibles à chaque instant. Pour intégrer efficacement un apprentissage sur les valeurs des poids des règles de comportements, il est nécessaire de réduire l'ensemble des états

et celui des actions à ceux qui sont pertinents. L'ensemble des états pertinents est constitué de toutes les combinaisons possibles des valeurs des poids associés aux règles de comportement. Pour simplifier l'apprentissage et en diminuer fortement la complexité, l'ensemble des valeurs de ces poids est discrétisé et réduit à 11 classes pour chacun des poids correspondant à l'arrondissement de ces valeurs à la première décimale. Les actions, quant à elles, seront limitées à l'augmentation de l'un des poids pour chacun des trois composants définissant le comportement de l'agent, ce qui implique la diminution des autres poids de ce composant pour respecter la contrainte sur les poids d'un même composant. Le choix de maintenir les poids tels qu'ils sont est également une action possible. Ainsi, l'espace des états est réduit à 11^{10} états et l'espace des actions à 80 $(5 * 4 * 4)$ actions.

L'apprentissage par renforcement convient à un tel environnement, où l'agent n'a pas de modèle de l'environnement. Avec un tel apprentissage, l'agent n'a besoin que de connaitre l'action qu'il effectue et le résultat de celle-ci. Ce dernier peut être calculé en fonction de différents paramètres indicateurs de l'utilité d'une telle action. Dans le cas présent, la quantité de nouvelles informations reçues, leur fiabilité et le nombre d'informations inutiles reçues sont autant d'indices sur l'utilité, ou l'inutilité, de l'exécution de cette action.

La méthode *Q-learning*, particulièrement, est une *méthode sans modèle*. Elle calcule la valeur des actions, les valeurs-Q au lieu d'apprendre l'utilité des états. Les valeurs-Q sont liées à l'utilité des états par la relation suivante :

$$U(s) = \max_a Q(s, a) \tag{5.1}$$

À chaque fois qu'une action est exécutée, la valeur-Q associée à cette action et à l'état d'origine est mise à jour comme suit :

$$Q(s, a) \leftarrow r(s, a) + \gamma \max_{a'} Q(s' + a') \tag{5.2}$$

où γ est un facteur compris entre 0 et 1 et qui détermine l'importance donnée aux récompenses non immédiates de l'action et s' est l'état résultat de l'action a depuis l'état s.

Cette équation de mise à jour n'est valable que si on connait avec certitude l'état s' et la récompense $r(s, a)$ résultants de l'action, ce qui n'est pas le cas dans *Danger mapping*, puisque l'environnement est considéré comme séquentiel et dynamique. Le problème n'est donc pas, comme il devrait l'être pour être résolu par la méthode Q-learning, représentable par un MDP déterministe mais bien un MDP non-déterministe. Il existe une variante non-déterministe du Q-learning qui définit la valeur-Q sur base de la moyenne des récompenses directes et ultérieures (2.7).

Cette équation ne permet cependant pas aux mises à jour de la table des valeur-Q de converger. Il est nécessaire de changer l'équation de mise à jour pour (2.8), ainsi l'influence des nouvelles récompenses obtenues en effectuant une action a depuis l'état s diminue au plus cette action a déjà été effectuée depuis cet état. Plus un couple état-action a été exploré, moins l'exploration de celui-ci devient importante et plus les valeurs déjà calculées le sont, ce qui atténue les variations dues aux mises à jour et qui permet aux valeur-Q présentes dans la table de converger vers l'optimum.

La probabilité d'arriver dans l'état s' est calculable. En effet, si la valeur du poids choisi pour être augmenté est fixe, il se peut que la diminution des autres poids fassent basculer ceux-ci dans une autre classe de poids. La probabilité qu'un poids non choisi décroisse d'une classe de valeur est de $\frac{1}{n_{\text{poids}}-1}$. Par exemple, le composant régissant le déplacement de l'agent a un probabilité $p_3 = \frac{1}{3} * \frac{1}{3} * \frac{1}{3} = \frac{1}{27}$ que les 3 autres poids diminuent d'une classe de valeur, une probabilité $(p_2 = \frac{1}{3} * \frac{1}{3} - p_3) = \frac{2}{27}$ pour chaque paire de poids diminués pour que ceux-ci changent de classe, une probabilité $p_1 = \frac{1}{3} * \frac{2}{3} * \frac{2}{3} = \frac{4}{27}$ pour chacun des poids que seul celui-ci ait changé de classe et une probabilité $p_0 = \frac{2}{3} * \frac{2}{3} * \frac{2}{3} = \frac{8}{27}$ qu'aucun poids autre que celui augmenté

ne change de classe de valeur. La somme de ces probabilités vaut bien 1 ($\frac{1}{27} + 3 * \frac{2}{27} + 3 * \frac{4}{27} + \frac{8}{27} = \frac{27}{27}$).

Pour calculer la valeur-Q associées à un couple état-action, il est nécessaire de définir comment calculer la récompense de celui-ci. Pour estimer l'utilité d'un changement de stratégie, par le biais d'une action sur le poids des règles de comportement, il est nécessaire de tester les résultats de cette stratégie. Pour ce faire, les données collectées par l'agent, qu'elles soient directes ou déduites de communications, pendant un certain échantillon de temps sont analysées et la proportion d'informations fiables et nouvelles parmi toutes celles rassemblées donne une mesure de l'efficacité de la stratégie nouvellement adoptée, et donc de l'intérêt d'avoir changé la précédente. Le calcul de la récompense d'une action a depuis un état s pour l'agent i est définie de la façon suivante :

$$r(i, n) = \alpha\pi_{\text{Iutiles}}(n) + \beta\pi_{\text{Ifiables}}(n) \tag{5.3}$$

où n est l'échantillon de données reçues, α et β sont des paramètres servant à calibrer les importances respectives de l'utilité des informations et de leur fiabilité et dont la somme vaut 1. $\pi_{\text{Iutiles}}(n)$, le taux d'utilité des informations reçues dans l'échantillon n se calcule comme suit :

$$\pi_{\text{Iutiles}}(n) = \frac{\#\text{infos-utiles}(n)}{\#n} \tag{5.4}$$

où infos-utiles(n) est un sous ensemble de n contenant toutes les informations jugées utiles, c'est-à-dire celles qui donnent des informations nouvelles sur la carte ou les autres agents. $\pi_{\text{Ifiables}}(n)$, le taux d'informations fiables dans l'échantillon, se calcule comme suit :

$$\pi_{\text{Ifiables}}(n) = \frac{\sum_{k \in n} T_k}{\#n} \tag{5.5}$$

où T_k est la confiance qu'a l'agent i en l'émetteur de l'information k.

Le choix de l'action à entreprendre lorsque l'agent est dans un état s doit permettre à l'agent de maximiser l'utilité attendue de l'état résultant de cette action. Néanmoins, pour éviter d'exécuter sans cesse une action sous-optimale, il est nécessaire d'explorer les autres actions. C'est pourquoi le choix de l'action à entreprendre pendant la phase d'exploration doit permettre d'estimer la valeur-Q de chacune de ces actions avec autant de précision que possible. Pour ce faire, nous adoptons une approche probabiliste. La probabilité de choisir une action a est proportionnelle à l'estimation de sa valeur-Q, celles-ci devant être non nulles pour toute action conforme. Dans la recherche d'un équilibre entre l'exploitation des estimations des valeurs-Q connues et de l'exploration des paires état-action méconnues, nous appliquons à chaque action exécutable une probabilité suivant la fonction de distribution de Boltzmann. La probabilité $Pa_i|s$ de choisir une action a_i depuis un état s parmi j actions possibles se définit comme décrit en (2.5). Dans cette équation, la température T doit être initialement élevée et progressivement décroitre au fur et à mesure des itérations de l'algorithme.

Le grand nombre d'états possibles du système pose plusieurs problèmes. Premièrement, la table des valeur-Q ne présente pas moins de 800.000.000.000 d'entrées ($5*4*4\text{actions}*10^4*10^3*10^3\text{états}$) ce qui, outre la durée colossale de parcours de celle-ci, occuperait plus de 54 téraoctets d'espace disque. De plus, si la convergence de l'algorithme a été prouvée, la vitesse de celle-ci dépend fortement du nombre de couples état-action possibles. Pour réduire le nombre d'états, il est possible par diviser le processus d'apprentissage en trois processus distincts ne portant que sur un seul composant. En effet, l'apprentissage des poids des règles optimaux pour chacun de ces composants peut se faire sans connaitre l'état des deux autres. Cette séparation de l'effort d'apprentissage par composants réduit le nombre de couples état-action à 50.000 pour le composant de déplacement ($10^4 * 5$) et 4000 pour les deux autres ($10^3 * 4$), ce qui revient à 58.000 entrées toutes tables confondues. Il est également possible de répertorier l'ensemble de ces entrées état-action dans un arbre

de recherche plutôt que dans un tableau. Ainsi, la recherche d'un élément dans celui-ci prendra jusque 4 itérations au lieu de 58.000. Enfin, les valeurs des poids des règles pour un même composant ne sont pas indépendantes puisqu'elle sont contraintes par l'équation $\sum_{k \in \text{composant}} w_k = 1$. Le nombre d'états effectifs est donc bien moindre que celui annoncé précédemment. Si la valeur de W_1 peut prendre 11 valeurs, celle de W_2 ne peut en prendre que $11 - 10 * W_1$, celle de W_3 que $11 - 10 * W_1 - 10 * W_2$ et celle de W_4 ne peut prendre qu'un seule valeur, à savoir $1 - W_1 - W_2 - W_3$. Pour éviter de réserver une quantité excessive de place mémoire, il convient de créer l'arbre contenant les valeurs-Q dynamiquement. Les feuilles nouvellement créées de l'arbre ont une valeur-Q maximale, c'est-à-dire de 1. Ainsi, les feuilles encore non explorées ont une grande probabilité d'être explorées au passage suivant de l'agent dans cet état.

En définitive, l'algorithme utilisé sera la suivant :

Algorithm 2 Q-Learning appliqué à Danger Mapping

Créer l'arbre des valeurs-Q
initialiser l'état de départ s_0
while la simulation n'est pas terminée **do**
  choisir a_t en suivant l'approche probabiliste
  l'action produit s_{t+1} et r_t est estimé sur un échantillon d'informations n collectées pendant un intervalle de t_{inter} unités de temps.
  if s_{t+1} n'existe pas dans l'arbre des valeurs-Q **then**
    ajouter ce dernier et initialiser les valeurs-Q de ses actions à 1.
  end if
  $Q_{t+1}(s_t, a_t) \leftarrow (1 - \alpha_t)Q_t(s_t, a_t) + \alpha_t(r_t + \gamma * \sum_{s' \in S} p(s_{t+1}|s_t, a_t)Q_t(s_{t+1}, a'))$
  $t \leftarrow t + 1$
end while

L'implémentation de cet algorithme dans le modèle existant Danger Mapping demande l'introduction de nouveaux paramètres :

- La fréquence à laquelle un agent choisit d'effectuer une action sur le poids des règles de comportement et, donc, à laquelle il met à jour ses arbres de décision. Cette fréquence est exprimée en nombre de cycles entre chaque changement.
- Le poids associé à la précision des informations collectées pour l'évaluation de la récompense d'une action.
- Le poids associé à la fiabilité des informations collectées pour l'évaluation de la récompense d'une action.
- Les poids associés aux règles de comportement deviennent les poids initiaux de ces mêmes règles.

Chapitre 6

Résultats

Une fois la solution proposée implémentée dans le modèle Danger Mapping, il convient d'analyser les résultats produits et de comparer ceux-ci avec les résultats obtenus sans apprentissage. Ces derniers peuvent être produits en utilisant des valeurs moyennes pour le poids des règles de comportement ou en exploitant les valeurs optimales découvertes par l'application de l'algorithme génétique. Le choix des valeurs initiales des poids associés aux règles de comportement, de la fréquence des cycles d'apprentissage, de la proportion d'agents défectueux et de l'importance donnée à la précision et à la fiabilité des informations reçues sont autant de paramètres affectant le nombre de cycles nécessaires pour arriver à une solution pour un système donné.

6.0.1 Conditions d'expérimentation

Pour parvenir à des résultats concluants, les tests expérimentations doivent respecter certaines contraintes. Pour chaque série de tests, la proportion d'agents non fiables doit être constante. En effet, cette valeur est le paramètre le plus déterminant dans la recherche des valeurs optimales des poids associés

aux règles de comportement. Des séries de 10 tests sont effectuées pour un même ensemble de paramètres, d'abord sur la version initiale de Danger Mapping, ensuite sur la même version mais en utilisant les poids optimaux [8] et enfin sur la version modifiée du modèle. Pour ce dernier, les poids initiaux sont les mêmes que ceux choisis arbitrairement à la première étape. Ces séries de tests permettent d'évaluer les performances que permet l'ajout de l'algorithme d'apprentissage décrit précédemment en les comparant à celles du modèle original.

Des tests sont également réalisés à l'aide du mode "batch" proposé par GAMMA afin de déterminer l'impact qu'ont les différents paramètres sur ce gain ou cette perte de performances. Pour ce faire, le test précédent est réalisé pour 5 valeurs différentes du paramètre choisi. La taille de l'environnement, la proportion d'agents non fiables, les valeurs initiales des poids des règles et la fréquence des cycles d'apprentissage sont les différents paramètres testés.

Enfin, il est utile de noter que le modèle original Danger Mapping demande une puissance de calcul conséquente. La plateforme GAMA v1.3 n'utilisant qu'un seul coeur du processeur de l'ordinateur, les performances diminuent rapidement lorsque le nombre d'agents augmente. Il a donc été, jusqu'ici, impossible de tester le modèle original et celui modifié pour plus de 10 agents, la simulation d'un système de 15 agents durant jusqu'à une heure pour un environnement de 400x400.

6.0.2 Analyse des résultats

Le modèle Danger Mapping, auquel est ajouté un mécanisme d'apprentissage sur le poids des règles de comportement, produit des résultats supérieurs à ceux du même modèle lorsque ces poids sont choisis arbitrairement. Le nombre de cycles nécessaires pour cartographier le terrain diminue sensiblement. Cette amélioration s'explique facilement : les calibrages des poids des

règles ayant produit des informations nouvelles et fiables sont plus susceptibles de produire à nouveau des résultats intéressants. On observe néanmoins qu'il faut à l'algorithme un bon nombre de cycles avant que le système ne tende à effectuer les bons calibrages. Cela s'explique par le grand nombre d'entrées dans les arbres de décision. Ce dernier impose une longue phase d'exploration avant d'être exploitable, ce qui indique un manque de performances de l'algorithme d'apprentissage dû à la taille du domaine. Bien que l'algorithme du Q-Learning non déterministe converge lorsque l'impact des nouvelles récompenses sur les valeurs-Q est décroissant au cours des itérations, cette convergence est très lente dans ce cas précis. Il serait intéressant d'optimiser l'algorithme utilisé pour accélérer cette convergence. Un façon de réaliser cela serait de modifier plusieurs valeurs-Q à chaque itération de l'algorithme d'apprentissage.

La taille de l'arbre et son temps de parcours sont indépendants de la taille du problème, c'est-à-dire du nombre d'agents, de la taille de la carte et de la fiabilité de ces agents. L'algorithme d'apprentissage est donc plus adapté à des problèmes de taille conséquente, où le temps de calibrage de l'arbre sera rentabilisé par le temps gagné une fois les poids des règles optimisés. L'apprentissage permet tout particulièrement de réduire le temps passé dans la phase de "confirmation", c'est-à-dire lorsque tous les agents possèdent des informations sur toute la carte, mais qu'ils doivent confirmer la fiabilité de ces informations jusqu'à dépasser un seuil choisi.

Les performances observées sont toutefois loin d'être aussi bonnes que celles induites par les valeurs optimales des poids pour un système donnée. Ces dernières sont définies par l'application de l'algorithme génétique à un système donné et sont théoriques puisqu'elles nécessitent une connaissance totale du système et, particulièrement, du taux d'agents non fiables. Si la version de Danger Mapping dont les agents sont doués d'apprentissage convient davantage aux situations pratiques où l'environnement est inconnu, il est

nécessaire que l'algorithme estime l'utilité des actions sur les poids des règles avant de modifier celles-ci de manière efficace.

Appliqué à un problème extrêmement grand, l'algorithme pourrait même prétendre à des performances supérieures à celles proposées par la version sans apprentissage dont les poids des règles sont optimaux. La cause en est que les poids des règles dans Danger Mapping avec apprentissage sont dynamiques et peuvent s'adapter à la phase d'exploration en cours. En effet, si les poids donnés par l'algorithme génétique sont optimaux sur la durée totale de la simulation, les poids optimaux associés à différentes phases de la simulation peuvent être différents. Ainsi, la phase de confirmation nécessite davantage de communications avec les agents fiables et de déplacements vers des zones dont la fiabilité est faible alors que la phase d'exploration préliminaire profite davantage des informations récupérées en visitant des zones non explorées et en communiquant avec des agents proposant un grand nombre de nouvelles informations. Pour profiter au mieux de l'aspect dynamique des valeurs des poids associés aux règles de comportement, il est néanmoins nécessaire de ne pas implémenter l'influence régressive des récompenses sur le calcul de la valeur estimée des valeurs-Q. Ce faisant, l'algorithme de Q-Learning non déterministe perdrait la propriété de convergence et il serait impossible de garantir mathématiquement la convergence des poids des règles vers leur valeur optimale.

Un autre avantage de l'apprentissage par rapport à l'utilisation d'une valeur fixe pour les poids des règles de comportement est que les agents peuvent tendre vers des comportements différents en fonction de leur fiabilité. En effet, un agent non fiable verra la plupart de ses communications refusées par les autres agents. Son comportement tendra donc, à l'inverse des agents fiables, à diminuer le nombre de ses communications portant sur des valeurs qu'il a lui-même collectées au fur et à mesure de la simulation, ainsi qu'à éviter de récolter directement les informations, ce qui tend à le confiner à

un rôle de "relai". La spécialisation des agents en "rôles" dépendants de leur fiabilité est un renforcement du mécanisme d'auto-organisation déjà présent qui permet au système d'atteindre une meilleure efficacité.

Si l'apprentissage permet une diminution sensible du nombre de cycles nécessaires pour cartographier le terrain, il n'y a aucune garantie que cela diminue le temps réel d'exploration. En effet, le mécanisme d'apprentissage vient avec un coût non négligeable en temps de calcul, ce qui a pour effet de ralentir l'exécution de la simulation. Ce coût est prohibitif dans des systèmes dont le temps nécessaire à l'exécution des actions est faible, mais il est négligeable, relativement parlant, lorsque ces actions demandent beaucoup plus de temps que les processus de décisions eux-mêmes. Ainsi un robot matériel qui doit se déplacer en terrain inconnu pourra modifier l'importance de ses règles de comportement au prix d'un traitement négligeable par rapport au temps dont il a besoin pour se déplacer vers une autre position. À l'inverse, l'apprentissage a un cout conséquent lors des simulations effectuées sur le modèle Danger Mapping où on entreprend un action d'apprentissage tous les quelques cycles.

Chapitre 7

Conclusion

Dans ce travail, nous avons tenté de résoudre le problème lié au calibrage de l'importance des performances individuelles et de celle associée à la coordination dans les systèmes multi-agents auto-organisés en calibrant ces importances par apprentissage, de manière continue. Pour ce faire, les notions de base liées aux systèmes multi-agents, à l'apprentissage et à l'auto-organisation ont été présentées. Après avoir étudié l'algorithme du Q-Learning, qui est un exemple d'apprentissage par renforcement, le modèle de systèmes multi-agents auto-organisé Danger Mapping a été décrit et l'algorithme a été adapté pour permettre à ce dernier d'apprendre les poids des règles de comportement qui maximisent l'efficacité des agents. Au terme de l'implémentation de cet apprentissage dans le modèle existant, ce document a analysé les résultats produits, en comparant notamment ceux-ci avec les résultats produits par des travaux précédents sur les valeurs optimales de ces poids.

Ces résultats sont, sans surprise, très dépendants de l'environnement dans lequel évoluent les agents. La taille du problème, en terme de nombre minimal d'opérations nécessaires pour le résoudre, est un facteur particulièrement important lorsqu'il faut comparer les performances du modèle Danger Map-

ping avec apprentissage et celles du modèle sans apprentissage, mais dont les valeurs associées aux poids des règles de comportement sont optimisées. L'intérêt de l'apprentissage dans un tel modèle est d'autant plus remarquable lorsque ces problèmes sont assez grands. Deux atouts de ce modèle par rapport au modèle théorique proposé précédemment sont l'aspect dynamique des poids des règles de comportement et l'hétérogénéité des comportements des agents.

Ces résultats permettent néanmoins de confirmer qu'ajouter un apprentissage sur le poids des règles de comportement est plus efficace que le choix arbitraire de ceux-ci. L'apprentissage permet donc bien d'améliorer l'efficacité d'un système multi-agent auto-organisé en agissant sur le calibrage de l'importance associée aux performances individuelles des agents et celle associée à la coordination des agents. L'avantage de l'apprentissage, bien que les performances soient moindres par rapport à l'utilisation des poids optimaux obtenus grâce à l'algorithme génétique, est que cette solution est applicable en pratique, lorsque les agents ne connaissent pas leur environnement.

Il est possible d'étendre cette recherche en comparant différents algorithmes d'apprentissage et en appliquant ceux-ci à des environnements dotés de caractéristiques différentes. Il serait intéressant de considérer des systèmes dans lesquels l'aspect dynamique des poids des règles et l'hétérogénéité des comportements des agents peuvent avoir un impact important sur les performances du système. Un exemple d'un tel système pourrait voir son environnement changer sans interactions des agents et ces derniers pourraient avoir des caractéristiques et des performances différentes, augmentant ainsi l'avantage relatif des deux atouts sus-cités.

Bibliographie

[1]

[2] Olivier Buffet. *Apprentissage par renforcement dans un système multi-agents*. PhD thesis, Université Henri Poincaré, 2000.

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

[4] Sherief Abdallah Chongjie Zhang, Victor R. Lesser. Self-organization for coordinating decentralized reinforcement learning. *AAMAS*, pages 739–746.

[5] Danny Sutanto Dayong Ye, Minjie Zhang. Self-organisation in an agent network via learning. In *9th International Conference on Autonomous Agents and Multiagent Systems : Volume 1*, 2010.

[6] A. Boucher A. Drogoul E. Amouroux, C. Quang. Gama : an environment for implementing and running spatially explicit multi-agent simulations. In *10th Pacific Rim International Workshop on Multi-Agents (PRIMA), Thailand*, 2007.

[7] S. Desvaux E. Amouroux, A. Drogoul. Towards virtual epidemiology : an agent-based approach to the modeling of h5n1 propagation and persistence in north-vietnam. In *PRIMA08, Lecture notes in Artificial Intelligence*, 2008.

[8] Inconnu. Applying genetic algorithm to optimise behaviour of agents in a self-organisation. In *Inconnu*, Inconnue.

[9] François Mairesse Jean-Yves Lawson. *Apprentissage de la coordination dans les systèmes multi-agents*. PhD thesis, Université Catholique de Louvain, 2004.

[10] Erwan Livolant. *Apprentissage Multi-Agents par Systèmes de Classeurs : étude préliminaire*. PhD thesis, Université de Caen, 2003.

[11] Frédéric Armetta Benoit Gaudou Richard Canal Quang-Anh Nguyen Vu, Salima Hassas. Trustsets - using trust to detect deceitful agents in a distributed information collecting system. In *IEEE-RIVF International Conference on Computing and Communication Technologies, Best Student Paper Award*, 2010.

[12] Frédéric Armetta Benoit Gaudou Richard Canal Quang-Anh Nguyen Vu, Salima Hassas. Combining trust and self-organization for robust maintaining of information coherence in disturbed mas. In *SASO 011 : 5th IEEE International Conference on Self-Adaptive and Self-Organizing Systems, IEEE Computer Society*, 2011.

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

[14] Peter Norvig Stuart Russel. *Intelligence artificielle*. Pearson Education, 2010.

[15] Richard S. Sutton. Introduction : The challenge of reinforcement learning. *Machine Learning*, 8 :225–227, 1992.

[16] Alan M. Turing. Computing machinery and intelligence. *Mind*, 59 :433–460, 1950.

[17] Marie-Pierre Gleizes Valérie Camps. Réflexions sur l'apprentissage en univers multi-agents. In *Les systèmes multi-agents*, 1996,1997.

[18] Quang-Anh Nguyen Vu. *Cohérence et robustesse dans un système multiagent perturbé : application à un système décentralisé de collecte d'in-*

formation distribué. PhD thesis, Université Claude Bernard Lyon I, 2011.

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