*Laboratoire GREYC*


*Département Informatique*

# Apprentissage Multi-Agents par Systèmes de Classeurs : étude préliminaire

**Rapport de Stage de DEA**
**Soutenu le 30 septembre 2003**

**Encadrant : Serge Stinckwich**
**Jury : Anne Nicolle**
**Serge Stinckwich**

**Erwan Livolant**
**DEA Intelligence Artificielle et Algorithmique**

# Résumé


## Apprentissage Multi-Agents par Systèmes de Classeurs

Un système de classeur (LCS Learning Classifier System) est un système à base de règles permettant à un agent d'apprendre un comportement adéquat à son environnement. Cette technique d'apprentissage par renforcement est particulièrement efficace pour des problèmes de décision mono-agent et markoviens. Elle apporte également des capacités de généralisation de règles intéressantes.

Le méta-apprentissage, selon J. Schmidhuber, permet d'adapter toute technique d'apprentissage par renforcement à des applications multi-agents. Le but du stage est d'intégrer aux LCS les actions auto-modificatrices du méta-apprentissage ainsi que l'algorithme « Success-Story » (SSA) qui valide ce dernier.

Le projet MAAM, qui représente le domaine applicatif du stage, vise à établir une architecture de robotique collective. Celle-ci est basée sur des atomes robotiques pouvant se connecter ensemble pour former des molécules en vue de réaliser des tâches complexes. Les LCS adaptés à l'apprentissage multi-agents peuvent permettre la planification des déplacements de ces architectures robotiques dans un environnement inconnu.


**Mots clés :** Système de Classeurs, Apprentissage par Renforcement, Méta-Apprentissage, Système Multi-Agents, Robotique Collective.

# Abstract


## Multi-Agent Learning Classifier System

A learning classifier system (LCS) is a rule-based system which allows an agent to learn a correct behaviour in his environment. This reinforcement learning method is specialy efficient for markovian mono-agent decision problem. This method gives although interesting generalization capabilities.

J. Schmidhuber's metalearning adapts every reinforcement learning methods to multiagent applications. The training period aims at integrate, in LCS, the automodification actions of the metalearning and the «Success Story» Algorithm (SSA) which valids the metalearning.

MAAM project, which is the training period application field, aims at establish a collective robotic architecture based on robotic atoms. These atoms can connect themselves in order to create a molecule that deals with complex tasks. Multi-agent LCS should allow the movement planification of these robotics architectures in an unknown environment.


**Key words :** Learning Classifier System, Reinforcement Learning, Metalearning, Multi-Agent System, Collective Robotics.

# Remerciement

Je remercie tout particulièrement Serge Stinckwich, mon maître de stage, pour avoir proposé ce sujet, et pour m'avoir prodigué ses conseils tout au long de ce semestre.

Je tiens à remercier Anne Nicolle pour me faire l'honneur de participer au jury de soutenance.

Mes remerciements s'adressent également à Michaël Piel pour l'excellent travail qu'il a réalisé sur la plate-forme en Squeak « Classifier Learning ».

J'adresse finalement un grand merci à tous mes camarades du DEA pour leur bonne humeur et leur soutien qui m'a permis de travailler dans une ambiance très agréable.

# Table des matières

# Chapitre 1

# Introduction

Ce rapport présente le travail effectué lors de mon stage de recherche réalisé dans l'équipe MAD (Modèles, Agents et Décision) au sein du laboratoire GREYC. Ce stage est encadré par Serge Stinckwich. Le sujet de cette recherche vise à adapter une technique de méta-apprentissage (ML pour *Metalearning*) à la méthode d'apprentissage par renforcement que sont les Systèmes de Classeurs (LCS pour *Learning Classifier Systems*).

Nous présentons tout d'abord, dans ce premier chapitre, le domaine applicatif du stage, ainsi que la problématique de la recherche engagée. Dans la deuxième partie, nous décrivons les théories et les méthodes qui forment l'état de l'art de notre étude. Ensuite, le troisième chapitre montre le travail réalisé pendant ce stage. Avant de conclure, nous exposerons et commenterons dans le chapitre 4 quelques résultats expérimentaux.

## 1.1 Domaine applicatif

Le cadre d'application de ce stage est celui du projet MAAM[1]. Son nom est un acronyme récursif pour Molécule = Atome | (Atome + Molécule). Ce projet de recherche fait partie du programme CNRS/INRIA Robea (Robotique et Entité Autonome)[2]. Plusieurs équipes de recherche en électronique, robotique et informatique y participent :

- LESTER et VALORIA de l'Université de Bretagne Sud,
- LIP6 et LRP de l'Université Paris IV,

[1] site de l'équipe de Caen http ://www.iutc3.unicaen.fr/serge/ProjetMAAM
[2] site officiel http ://www.laas.fr/robea/

– GREYC de l'Université de Caen,
– LIST du CEA.

Le but est de réaliser une nouvelle architecture de robotique collective basée sur des composants auto-organisés. Chaque composant élémentaire de ce système, nommé atome robotique, est conçu sur le principe des atomes de la physique : il dispose de degrés de connectivité qui lui permet de s'assembler pour former des « molécules » de diverses structures. La figure n° 1.1 montre un schéma prototype de ces robots. Chaque atome est autonome du point de vue moteur, énergétique et décisionnel. La possibilité de s'agglomérer en macrostructure permet de compenser les relatives limitations individuelles des robots pour réaliser des tâches complexes, tout en assurant une continuité de service en cas de défaillance d'un élément.

Ce système multi-agents pose des problèmes de planification répartie du déplacement des robots. Les techniques classiques employées dans le cadre des processus décisionnels de Markov ont posé des problèmes d'explosion combinatoire lors de précédentes recherches au sein de l'équipe MAD [Cal02]. Il apparait donc nécessaire de se diriger vers d'autres méthodes.

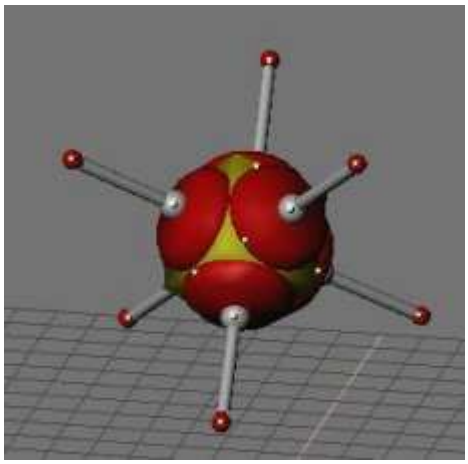

FIG. 1.1 – Représentation d'un atome robotique du projet MAAM

## 1.2 Problématique

Pour éviter ce problème d'explosion combinatoire, nous proposons d'utiliser les LCS. En effet, le principal avantage des LCS est leur capacité intrinsèque à généraliser les règles de comportement qu'ils apprennent. Ceci représente un atout majeur pour notre application.

Les LCS sont aujourd'hui efficaces sur des problèmes d'apprentissage décisionnel pour un unique agent possèdant une boucle sensori-motrice dans un environnement markovien. Le mécanisme d'apprentissage des classeurs implique que l'agent « pense » être le seul à modifier son environnement [BGS99]. Ceci pose naturellement problème pour notre application multi-agents.

La problématique de recherche consiste en l'adaptation des LCS aux environnements multi-agents, tout en bénéficiant de leurs caractéristiques propres. La direction de recherche choisie ici est de conjuguer la méthode de méta-apprentissage, conçue par J. Schmidhuber [SZW96], avec les systèmes de classeurs. Nous nous attacherons dans cette étude à expérimenter cette direction de recherche sur des problèmes classiques résolus par les LCS, dans le futur but de l'appliquer au projet MAAM.

# Chapitre 2

# État de l'art

Nous présentons dans ce chapitre les théories classiques d'apprentissage tirées de l'étude des comportements animaux, ainsi que leurs transpositions dans le domaine informatique avec l'apprentissage par renforcement (RL pour *Reinforcement Learning*). Ensuite, nous rappelons les définitions et principes qui concernent les Systèmes de Classeurs, qui prennent appui sur le RL. Enfin, nous exposons la méthode de méta-apprentissage conçue par J. Schmidhuber permettant d'aborder le travail de stage exposé par la suite.

## 2.1 Apprentissage par renforcement et apprentissage latent

L'apprentissage par renforcement est une méthode permettant à un agent plongé dans un environnement, d'apprendre un comportement adéquat. L'agent construit également un modèle déterminant la dynamique de ses interactions avec son milieu par un apprentissage latent. Ces techniques d'apprentissage sont basées sur des études psychologiques de l'apprentissage animal.

### 2.1.1 Apprentissage animal

Le conditionnement animal du début du vingtième siècle a été dominé par les idées « béhavioristes ». Pavlov introduit en 1927 un conditionnement que l'on qualifiera par la suite de « classique » en opposition au condi-

tionnement opérant de Skinner (1938) [Gér02].

Le conditionnement classique modélise l'apprentissage animal d'associations entre stimuli conditionnels et inconditionnels. Par exemple, dans la célèbre expérience du chien de Pavlov, l'odeur de la nourriture présentée à l'animal est un stimulus inconditionnel, qui provoque le réflexe de salivation. On présente au chien plusieurs fois de suite ce stimulus inconditionnel accompagné d'un autre stimulus, un son de cloche, dit « neutre » car il n'engendre aucun réflexe de prime abord. Par la suite, le stimulus neutre entraîne systématiquement la salivation du canidé sans la présence du stimulus inconditionnel. Une association est ainsi apprise entre les deux stimuli qui rend le neutre attractif pour l'animal.

Si le conditionnement pavlovien s'attache à décrire les associations entre stimuli, le conditionnement de Skinner s'intéresse aux contingences entre les actions et les stimuli. Il existe deux types de contingences. La première lie une action de l'animal à sa conséquence attractive ou répulsive. L'exemple de l'expérience de la boîte de Skinner met en scène un pigeon dans une cage. L'action d'appuyer sur un levier lui procure de la nourriture. Cette contingence à deux termes est renforcée au fur et à mesure du renouvellement de cette action par l'animal. La seconde contingence, dite différenciée, nécessite un autre simulus qui donne à l'animal l'indice d'une possible récompense. Dans notre expérience, la nourriture n'est donnée à l'oiseau qu'à la condition qu'une lampe soit allumée lorsqu'il actionne le levier.

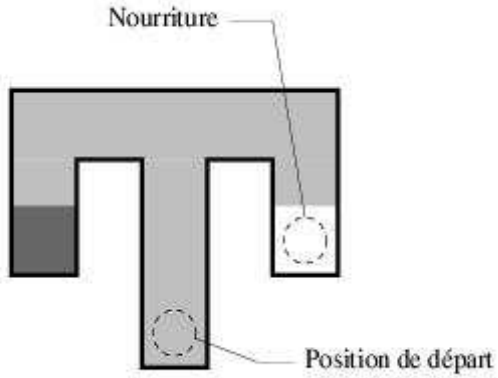


FIG. 2.1 – Expérience de Seward sur l'apprentissage latent

Dans les deux théories précédentes, l'apprentissage n'est motivé que par une récompense. Or des expériences, en rupture avec le « béhaviourisme », montrent que l'animal se crée un modèle de son environnement par apprentissage, indépendamment d'une quelconque satisfaction.

L'expérience de Seward en 1949 [Gér02] [Sto98], illustrée par la figure n° 2.1, place un rat dans un labyrinthe en T dont l'une des extrémités est blanche et l'autre noire. Après avoir laissé l'animal se déplacer librement dans son environnement pendant une certaine période sans aucune nourriture à sa disposition, on l'en retire pour lui faire subir un conditionnement classique : on lui présente de la nourriture sur une case blanche à plusieurs reprises. Lorsque l'on replace le rat dans le labyrinthe, il se dirige directement à la case blanche. L'animal a donc anticipé les conséquences de ses actions dans son environnement, pour choisir le meilleur chemin jusqu'à la nourriture.

### 2.1.2 Méthodes informatiques

Différentes méthodes ont été tirées de ces études psychologiques pour résoudre informatiquement des problèmes de décision nécessitant plusieurs actions successives [Gér02].

Une manière classique de représenter les interactions entre l'agent et l'environnement dans un problème de planification, est d'utiliser un Processus Décisionnel de Markov (MDP pour *Markov Decision Process*) défini par :

- un espace fini d'états $S$,
- un ensemble fini d'actions $A$,
- une fonction de transition $T : S \times A \rightarrow \Pi(S)$ où $\Pi(S)$ est une distribution de probabilités sur l'espace des états $S$,
- une fonction de récompense $R : S \times A \times S \rightarrow \Re$ qui associe une récompense immédiate à chaque transition possible.

Lors de la planification, la récompense obtenue lorsque l'agent arrive à son but doit être répercutée à l'ensemble des actions qui ont amené à cette dernière. On associe alors une valeur à chaque état représentant le cumul des récompenses espérées dans le futur et une qualité à chaque couple état-transition :

- fonction de récompense espérée $V : S \rightarrow \Re$,

– fonction de qualité $Q : S \times A \rightarrow \Re$.

Ce formalisme nécessite de pouvoir déterminer l'ensemble du problème avant de tenter de le résoudre. Il n'est donc d'aucune aide lorsque l'agent est plongé dans un environnement inconnu. En 1989, Watkins propose l'algorithme incrémental *Q-learning* [Gér02] qui permet de modifier une table des qualités, associant les états et les actions en fonction de l'expérience de l'agent. Cet algorithme ne met à jour qu'une qualité de sa table en fonction des qualités précédemment calculées à chaque pas de temps. En effet, il ne possède pas le modèle des transitions $T$ puisque l'environnement est inconnu. Ce n'est que lorsqu'une transition survient qu'elle est modifiée.

Cet algorithme est donc relativement lent puisque, s'il faut $n$ étapes pour qu'un agent se déplace de sa position de départ à son but, la mise à jour de la table des qualités nécessite $n$ essais. Cette inefficacité vient du fait que l'agent ne met pas à profit toute l'information de sa boucle sensori-motrice. De la même manière que lors d'un conditionnement opérant, il n'apprend que ce qui relève de la récompense. Cette boucle sensori-motrice offre pourtant l'information nécessaire à un apprentissage latent de la fonction de transition $T$. En effet, les conséquences d'une action ne se résument pas à une récompense mais également à une nouvelle situation.

Sutton propose, en 1992, l'architecture *Dyna* [Gér02] pour apprendre un modèle de la fonction de transition et l'utiliser pour accélérer l'apprentissage des qualités. L'algorithme *DynaQ+* réalise à l'instar de *Q-learning* la mise à jour des qualités à chaque pas de temps. Mais, en plus, le modèle environnemental est utilisé pour simuler des actions et ainsi mettre à jour l'ensemble de qualités associées aux transitions effectuées par l'agent dès le premier essai.

## 2.2 Systèmes de classeurs

Les systèmes de classeurs sont des systèmes à base de règles qui permettent de résoudre des problèmes d'apprentissage par renforcement et latent [Gér02] [GS01]. La principale originalité des LCS, par rapport aux méthodes précédentes est qu'ils procèdent à une généralisation de leurs règles de comportement.

### 2.2.1 Principe

Un exemple d'application classique pour l'apprentissage avec les LCS est présenté à la figure n° 2.2. Dans celui-ci, l'agent doit apprendre à rejoindre la position de récompense en un minimum de déplacements. Par déplacement, on entend ici un mouvement d'une case à une autre ou un déplacement vers un mur (auquel cas l'agent ne bouge pas).

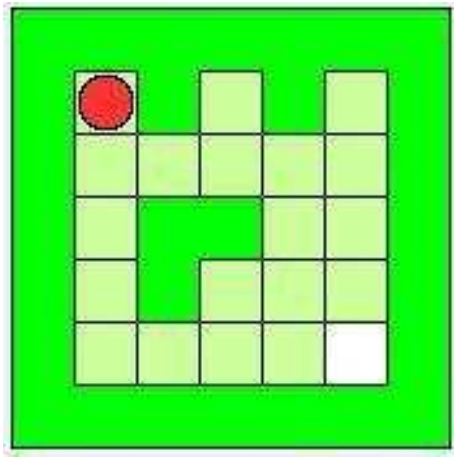

FIG. 2.2 – Exemple d'un environnement 2D « Maze228 ». L'agent (rond en haut à gauche) doit apprendre à rejoindre le point de nourriture (case blanche en bas à droite), où il obtient une récompense, en évitant les obstacles (parties foncées).

L'agent est immergé dans son environnement et possède une boucle sensori-motrice lui permettant d'agir sur son milieu et également de le percevoir à chaque pas de temps. Les LCS sont des systèmes à base de règles où ces dernières sont appelées « classeurs ». Chaque classeur est composé au moins d'une partie [Condition], d'une partie [Action] et d'une valeur sélective comme représente la figure n° 2.3.

| Condition | Action | Valeur Sélective |
|---|---|---|
| [10110010] | [a] | 0.8 |

FIG. 2.3 – Représentation minimale d'un classeur.

La situation dans laquelle se trouve l'agent est décrite par un ensemble d'attributs de perception locale dans la partie [Condition]. L'agent recherche

dans sa base de règles si une ou plusieurs [Condition] se recoupent à sa situation courante. S'il en trouve, il choisit de réaliser l'[Action] qui correspond au classeur qui possède la plus forte valeur sélective (celle qui lui apportera potentiellement la plus grande récompense). Sinon, il créé une nouvelle règle pour cette situation et réalise une action au hasard. Dans notre exemple, la perception locale de l'agent, représentée dans la figure n° 2.4, est [11100111] . On décrit par convention l'environnement à partir de la case située au nord de l'agent en tournant dans le sens des aiguilles d'une montre.

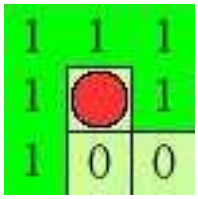


FIG. 2.4 – Exemple de la perception de l'agent de la figure n° 2.2.

Une fois que l'agent est arrivé à la case de nourriture, il reçoit une récompense de la part de l'environnement. Celle-ci est rétropropagée à toutes les actions qui ont mené à cette récompense par un algorithme comme le *Bucket Brigade* [Gér02].

La base de règles doit évoluer tout au long de l'interaction de l'agent avec son environnement. Pour cela, le système utilise généralement des algorithmes génétiques classiques. Pour favoriser l'exploration de l'environnement par l'agent, le LCS élimine un certain nombre de classeurs les moins bien adaptés à chaque pas de temps et les remplacent par d'autres. Ces nouveaux classeurs sont créés par croisements et mutations des classeurs qui ont les plus fortes valeurs sélectives. Les « gènes » considérés ici sont les attributs de la [Condition], ou la valeur de l'[Action].

### 2.2.2 Généralisation

Par rapport aux algorithmes d'apprentissage par renforcement de la famille Q-learning, les LCS ont l'avantage de posséder un mécanisme de généralisation qui exploite les régularités dans la dynamique d'interactions entre l'agent et l'environnement.

Dans la partie [Condition] d'une règle, chaque symbole spécifie la valeur de l'attribut correspondant pour que le classeur soit choisi. Le symbole #, appelé « peu importe » (don't care), exprime le fait que l'on peut ignorer cet attribut dans le processus d'appariement d'un classeur avec la situation courante de l'agent. Un classeur dont la partie [Condition] contient des symboles # est dit plus général que ceux qui n'en ont pas car il correspond à plusieurs situation de l'agent. Ainsi, un classeur contenant [11#0011#] pourra être associé à quatre situations différentes de l'agent : [11000110], [11000111], [11100110], [11100111].

Le mécanisme de généralisation réalise un groupement de plusieurs classeurs. La figure n° 2.5 illustre ce mécanisme. Les classeurs en question possèdent la même action et la même valeur sélective. On généralise en ne gardant dans la partie [Condition] que les attributs qui sont identiques. Les autres symboles sont remplacés par un #. On obtient ainsi, à la fin de ce mécanisme, un unique classeur qui convient à plusieurs situations.

| | |
|---|---|
| [10111010] | [a] |
| [00110010] | [a] |
| [10110000] | [a] |
| [00110010] | [a] |
| [#011#0#0] | [a] |

FIG. 2.5 – Exemple de généralisation à partir de la population de classeurs du système.

### 2.2.3 Différentes architectures

L'architecture générale d'un LCS, illustrée par la figure n° 2.6, est composée de quatre éléments :

- l'interface d'entrée qui formate les perceptions de l'environnement en messages pouvant être interprétés par le système,
- l'interface de sortie qui traduit les messages en actions effectives sur l'environnement,
- la liste des messages permettant de stocker tous les messages d'entrée, de sortie et internes,
- la liste des classeurs représentant la politique de l'agent.

A chaque pas de temps, les messages sont appariés aux classeurs. Les messages mis en correspondance disparaissent alors de la liste. Les messages d'actions, choisies parmi les classeurs activés par le mécanisme d'appariement, sont postés sur la liste. Dans notre exemple de la figure n° 2.6, on peut remarquer un message $MO$ qui n'est pas un message de sortie. C'est en fait un message interne qui sera apparié aux classeurs au pas de temps suivant.

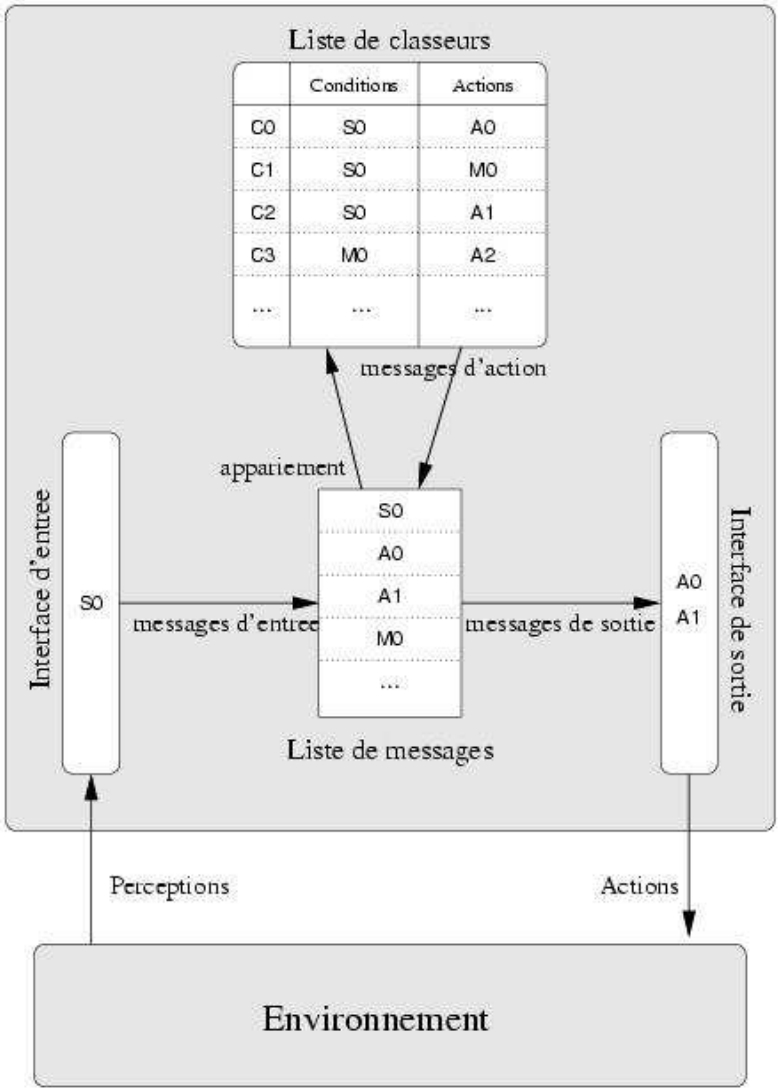


FIG. 2.6 – Architecture générale des systèmes de classeurs

L'architecture ZCS (pour Zeroth-level Classifier System) est l'une des plus simplifiées par rapport à l'architecture générale vue précédemment. Elle ne contient pas de liste de messages : les perceptions sont directement appariées avec les classeurs. Ces derniers sont composés du strict minimum :

- une condition,
- une action,
- une valeur sélective $f$ qui est ici la récompense attendue si l'action est effectuée.

La rétro-propagation de la récompense se fait avec l'algorithme *Bucket Brigade*. Le principal problème des ZCS est que leur valeur sélective dépend directement de la récompense. Ainsi, un classeur proposant une action, pourtant optimale mais très éloignée d'une source de récompense se verra peut être éliminée car sa valeur sélective sera faible. En effet, l'algorithme *Bucket Brigade* répercute seulement une partie de la récompense aux actions précédentes. Plus une action est loin du but, moins elle est renforcée.

Pour palier à ce problème, l'architecture XCS [But00] décorréle la récompense attendue et la valeur sélective des classeurs. Les classeurs de XCS contiennent :

- une condition,
- une action,
- une récompense attendue $p$,
- une erreur de précision $e$ qui représente la différence entre les récompenses attendues et effectives.
- une valeur sélective $f$ qui est une fonction de $e$.

Ces deux architectures sont basées sur l'apprentissage par renforcement. Il est possible d'intégrer également aux LCS la notion d'apprentissage latent. L'architecture ACS (pour *Anticipatory Classifier System*) [But01] permet ainsi l'apprentissage d'un modèle de l'environnement. Les classeurs d'ACS possèdent en plus de celle de XCS, une partie [Effet] qui anticipe la situation future de l'agent si l'[Action] du classeur est réalisée. L'agent apprend donc la dynamique de ces interactions avec son milieu.

Dans la partie [Effet], la future situation de l'agent est décrite avec les mêmes attributs que pour la [condition]. Le symbole = est utilisé dans la partie [Effet] du classeur pour exprimer le fait qu'un attribut reste inchangé une fois l'action réalisée.

## 2.3 Méta-apprentissage

Le Méta-Apprentissage selon J. Schmidhuber [SZW96] [ZS96] [Sch96] consiste à permettre au système d'apprentissage de modifier par lui-même la politique de comportement qu'il apprend, indépendamment de l'environnement. Schmidhuber introduit plusieurs notions permettant la réalisation d'un méta-apprentissage (MA) que nous définissons dans le paragraphe suivant.

Cette méthode est selon Schmidhuber applicable à n'importe quel algorithme d'apprentissage par renforcement. Dans [SZW96] par exemple, le MA est conjugué avec *Q-learning*.

Schmidhuber avance que sa méthode de méta-apprentissage convient aux applications multi-agents. Le système tient compte, selon lui, du fait que ce qu'un agent apprend à un moment donné affecte les conditions d'apprentissage des autres agents et même de lui-même par la suite.

### 2.3.1 Définitions

L'idée est de permettre à l'agent d'améliorer par un apprentissage la façon dont il apprend un comportement en se basant sur la notion d'accélération de la récompense acquise. Cette notion est introduite à l'aide d'un rapport renforcement/temps. Entre deux temps donnés de la vie de l'agent $t_1$ et $t_2$ ($t_2 > t_1$), il est possible de calculer l'accélération du renforcement par le critère suivant :

$$Q(t) = \frac{R(t_2) - R(t_1)}{t_2 - t_1}$$

Schmidhuber entend pouvoir valider grâce à ce critère les directives imposées par le méta-apprentissage, pendant ce laps de temps, à la politique apprise par l'algorithme d'apprentissage par renforcement classique.

Pour que le système puisse appliquer ces directives en question, il doit pouvoir réaliser, en plus des actions sur son environnement, des actions

sur son propre comportement. Ces dernières sont appelées actions auto-modificatrices (AAM).

De plus, le système doit pouvoir vérifier si les modifications qu'il réalise sont intéressantes pour le reste de la vie de l'agent. Pour cela, Schmidhuber propose un algorithme avec « backtrack » nommé « Success-Story Algorithm » (SSA) qui assure l'accélération de l'apprentissage.

### 2.3.2 Actions auto-modificatrices

Les AAM sont des actions qui modifient la politique que l'agent construit à l'aide d'un algorithme d'apprentissage par renforcement. La particularité principale de ces actions, dans la proposition de Schmidhuber, est qu'elles sont du même niveau que les actions sur l'environnement. Il faut entendre par là que c'est une seule et même politique qui mène à effectuer les actions sur l'environnement et les AAM. Ainsi les AAM sont auto-référentielles puisqu'elles agissent sur la politique qui les engendre et donc potentiellement sur elles-mêmes.

Classiquement les modifications réalisées sur la politique sont des actions d'incrémentation ou de décrémentation de la valeur discriminante qui détermine le choix d'action de l'agent suivant une situation.

Un processus de modification de la politique (PMP) commence lorsque l'une de ces AAM est effectuée. Il s'arrête lorsqu'une AMM particulière est choisie : l'action d'arrêt de mode auto-référentiel (ESM pour *End Self Mode*). Une deuxième phase du méta-apprentissage se déclenche alors. Elle se terminera lorsqu'une autre AAM modifiante sera exécutée.

Cette période peut être comparée à une phase d'exploitation du méta-apprentissage avant que ce dernier soit validé par l'algorithme SSA. Il est important de souligner que la durée de ces deux phases est variable et choisie par le système puisque l'occurence des AAM est déterminée par la politique de l'agent.

### 2.3.3 Algorithme « Success-Story »

Cet algorithme permet d'assurer une accélération des performances du système au cours du temps. La notion de temps est déterminée ici par le nombre d'actions réalisées (y compris les actions auto-modificatrices).

L'idée directrice de cet algorithme avec « backtrack » consiste en la sauvegarde de la politique courante avant la modification de celle-ci par une AAM. Si les modifications réalisées lors d'un PMP se révèlent inefficaces à accélérer le renforcement, elles seront invalidées et l'ancienne politique sera restaurée. Pour sauvegarder chaque ensemble de modifications correspondant à un PMP, on utilise une pile.

La validation d'un PMP se fait à chaque fin de la phase d'exploitation du Méta-apprentisage. SSA calcule l'évolution de la valeur du renforcement donné par l'environnement à l'aide du rapport renforcement/temps $Q(t)$.

L'algorithme vérifie si le PMP$_i$ courant a été bénéfique par rapport aux précédents à l'aide du critère « Success-Story Criterion » (SSC) défini par les deux inégalités suivantes :

- $Q(i,t) > \frac{R(t)}{t}$, le rapport renforcement/temps de l'unique PMP existant dans la pile est supérieur au renforcement moyen depuis la « naissance » de l'agent.
- $Q(i,t) > Q(k,t)$, avec $k < i$, le rapport renforcement/temps du PMP courant est supérieur à celui du PMP précédent dans le cas où au moins deux PMP existent dans la pile.

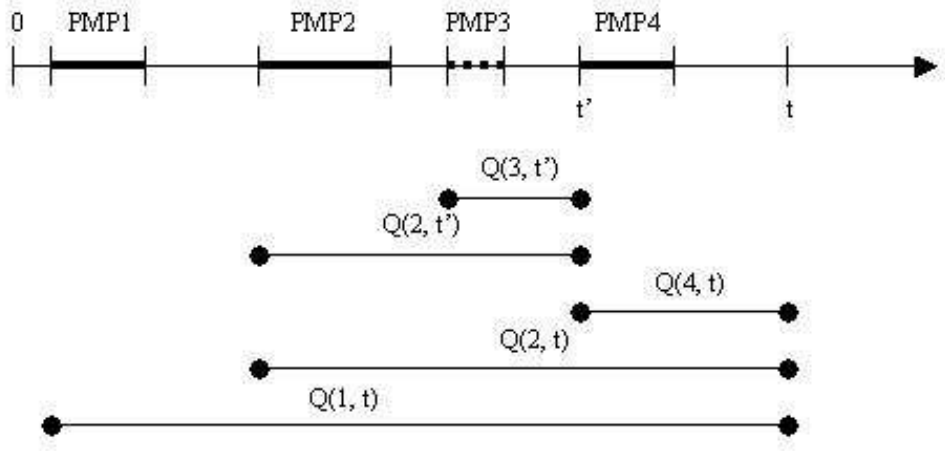


FIG. 2.7 – Exemple de fonctionnement de l'algorithme SSA

La figure n° 2.7 illustre le fonctionnement de SSA par un exemple. Dans celui-ci, lorsque le PMP2 commence, L'algorithme valide le PMP1 avec la première inégalité. Au début du PMP3, la seconde inégalité valide le deuxième processus. Ensuite, au temps $t'$, on obtient $Q(3,t') \leq Q(2,t')$. Le PMP3 s'est révélé inefficace, la politique obtenue par le PMP2 est alors restaurée. Les modifications de ce dernier sont conservées car $Q(2,t') > Q(1,t')$. Enfin, au temps $t$, le PMP4 est conservé car $Q(4,t) > Q(2,t) > Q(1,t)$.

# Chapitre 3

# Méta-apprentissage dans les systèmes de classeurs

Ce chapitre souligne, dans un premier temps, les buts recherchés dans le travail d'adaptation du méta-apprentissage de Schmidhuber au LCS. Ensuite, la méthode conçue est exposée et les choix concernant celle-ci sont discutés. Enfin, l'implémentation de l'algorithme SSA est décrite.

## 3.1 Buts recherchés

La direction de recherche choisie pour cette étude est d'associer les capacités de généralisation des systèmes de classeurs et la notion d'accélération de l'apprentissage de Schmidhuber.

La figure n° 3.1 présente le positionnement espéré de l'architecture réalisée pendant le stage par rapport aux autres systèmes selon les deux critères de la généralisation et de la vitesse d'apprentissage.

Pour réaliser la fusion des deux méthodes, il est nécessaire d'incorporer au LCS des actions auto-référencielles, c'est-à-dire des actions qui interviennent sur la population de classeurs du système, cette dernière représentant la politique de l'agent.

## 3.2 Actions auto-modificatrices spécifiques aux LCS

Les actions permettant d'apprendre à apprendre doivent pouvoir faire évoluer la politique de l'agent indépendamment de l'environnement. La

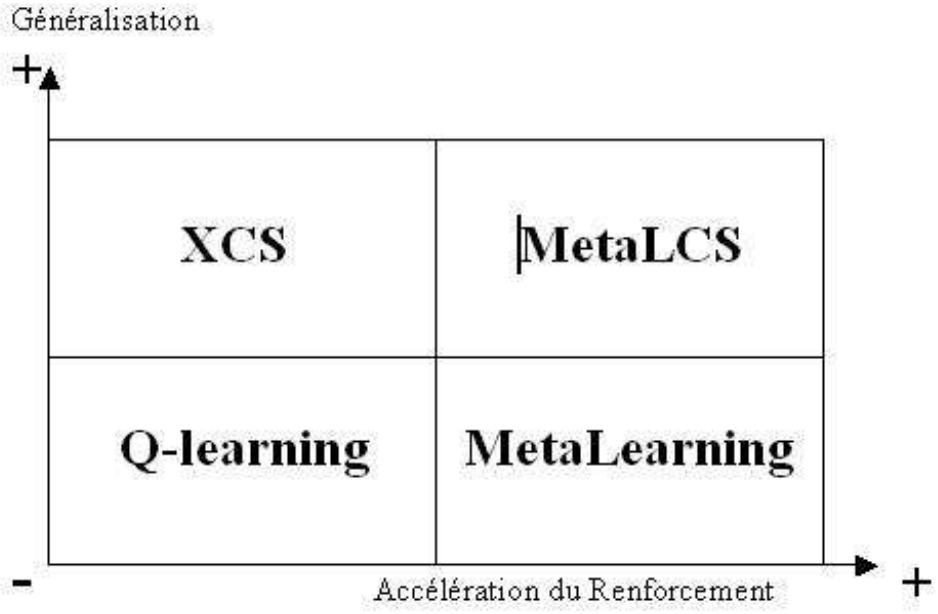


FIG. 3.1 – Positionnement espéré des méta-LCS par rapport à d'autres systèmes

prise de décision de l'animat, dans les LCS, se fait par la valeur sélective des classeurs. C'est donc sur cet élément des classeurs que les AAM doivent agir. Comme dans la méthode de Schmidhuber, on peut dénombrer trois AAM :

- incrémentation de la valeur sélective,
- décrémentation de cette même valeur,
- arrêt du mode auto-référentiel.

Dans l'achitecture de Schmidhuber, les AAM modifient une politique par un pourcentage variable suivant un paramètre. Dans notre étude, nous avons choisi de ne pas avoir ce pourcentage de modification variable mais constant. La valeur de ce pourcentage est à définir à l'aide de quelques expérimentations. Le choix d'un pourcentage fixe peut s'expliquer par le souci de ne pas alourdir la représentation des classeurs. Cependant, cette restriction peut impliquer une perte de performance car les possibilités d'exploration du méta-apprentissage sont réduites. Toutefois, le fonctionnement global de l'algorithme ne devrait pas être perturbé.

| Condition | Action | Paramètre | Valeur Sélective |
|---|---|---|---|
| [#011#0##] | [r] | [] | 0.8 |
| [10#100#1] | [l] | [] | **0.3** |
| ... | ... | ... | ... |
| [11#0#100] | [i] | [#00100#1] | 0.6 |
| [011##10#] | [f] | [] | 0.4 |
| ↓ | ↓ | ↓ | ↓ |
| [#011#0##] | [r] | [] | 0.8 |
| [10#100#1] | [l] | [] | **0.6** |
| ... | ... | ... | ... |
| [11#0#100] | [i] | [#00100#1] | 0.6 |
| [011##10#] | [f] | [] | 0.4 |

FIG. 3.2 – Exemple d'une population de classeurs avec une action autoréférentielle i incrémentant la valeur sélective du classeur (le second) pouvant être apparié au « pattern » [#00100#1]. Le valeur du pourcentage de modification est ici 100%. Les actions f, r et l sont environnementales.

Pour qu'une AAM s'applique à un classeur, il lui faut un paramètre désignant celui-ci. La représentation d'un classeur dans les méta-LCS est présentée à la figure n° 3.2. Ses différentes composantes sont définies ainsi :

- une condition (situation de l'animat),
- une action (sur l'environnement ou sur la politique de l'agent),
- un paramètre (vide pour les actions environnementales) désignant un pattern de classeurs sur lequel se porte la modification.
- les différentes valeurs permettant de calculer la valeur sélective du classeur (suivant ZCS, XCS, ACS) et cette dernière.

Le choix d'utiliser un pattern de condition pour associer une AAM à des classeurs s'explique par le problème du renouvellement de la population de classeurs. En effet, les classeurs sont constamment évalués par le système et les plus faibles sont éliminés. Cette élimination engendre un problème potentiel pour les AAM si celles-ci ne concernaient qu'un unique classeur. Ces AAM deviendraient obsolètes et devraient être supprimées à leur tour. Grâce au pattern, les AAM du méta-apprentissage peuvent perdurer indépendamment du mécanisme de suppression des classeurs inadaptés à l'environnement.

## 3.3 Implémentation de SSA

Pendant chaque PMP$i$, le système sauvegarde dans une pile $S$ les informations suivantes décrivant chaque modification $m$ :

- le renforcement : $S(m).R$,
- le temps : $S(m).t$,
- l'ancien classeur avant modification : $S(m).old$,
- la position dans la pile de la première modification du PMP$i$ : $S(m).f$.

L'algorithme SSA consiste en la vérification du SSC et le dépilement éventuel d'une mauvaise modification de politique, accompagné de la restauration de l'ancien classeur dans la politique $P$. Cet algorithme peut être décrit par cette boucle :

**Algorithme** *SSA*

**Tant que** $(sp \quad != \quad 0)$ **et**

$$\left(\frac{R(t) - S(S(sp).f).R}{t - S(S(sp).f).t} \leq \frac{R(t) - S(S(S(sp).f - 1).f).R}{t - S(S(S(sp).f - 1).f).t}\right)$$

**Alors** $sp = sp - 1$ ;
$P \leftarrow S(sp).old$ ;

Le premier élément de la pile, qui selon l'algorithme ne peut être dépilé, est une modification fictive. L'utilisation de cette modification fictive est utile pour n'avoir qu'un seul algorithme pour la validation des deux inégalités du SSC. Elle simule la naissance de l'agent à l'aide des valeurs d'attibuts suivants :

- $S(0).t = 0$,
- $S(0).R = 0$ car la récompence est nulle à la naissance de l'agent,
- $S(0).f = 0$,
- $S(0).old$ non initialisé.

# Chapitre 4

# Validation expérimentale

Ce chapitre expose une expérimentation réalisée sur le méta-apprentissage par système de classeurs à l'aide de la plate-forme « Classifier Learning ». Le but de l'expérimentations est, dans un premier temps, de valider la méthode d'apprentissage présentée au chapitre précédent en la comparant aux méthodes déjà existantes. De futures expérimentations viseront à confronter cette méthode à un apprentissage multi-agents.

## 4.1 Plate-forme « Classifier Learning » en Squeak

En parallèle de cette étude, une plate-forme d'expérimentations des LCS, programmée en Smalltalk/Squeak, est en cours de développement par Michaël Piel, stagiaire de maitrise de l'université de Caen. Une librairie des différentes architectures (ZCS et XCS) des LCS ainsi que des environnements de tests classiques (Maze et Woods) sont déjà partiellement livrés. Les expérimentations présentées par la suite utilisent cette plate-forme.

## 4.2 Expérimentation réalisée

L'architecture ZCS est la première à avoir été développée et fixée, c'est donc celle-ci que nous utilisons dans cette expérimentation. Nous avons choisi de valider le méta-apprentissage par rapport à une technique classique d'apprentissage par renforcement comme ZCS à l'aide d'un environnement qui sert de banc d'essais pour le test des LCS. Cet environnement est le « Maze 228 » de la figure n$^o$ 2.2 utilisé par Gérard [Gér02].

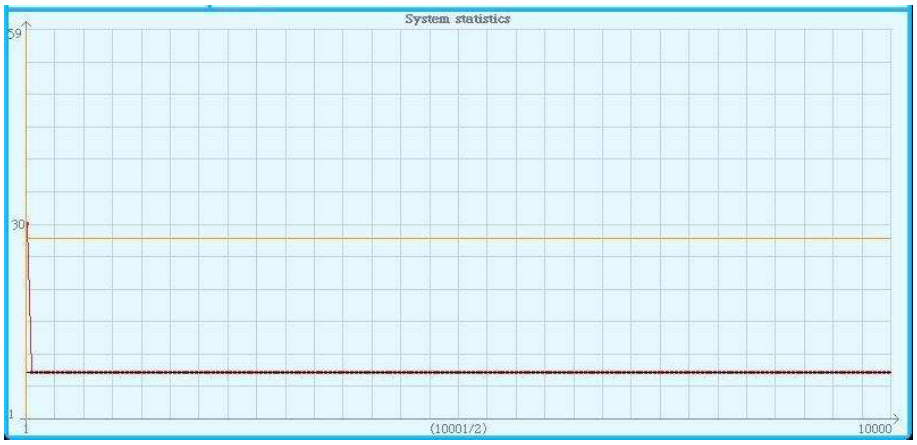


FIG. 4.1 – Résultats d'expérimentation de Meta-ZCS dans l'environnement « Maze 228 ». La courbe du haut représente les modifications empilées et celle du bas le nombre de déplacements de l'agent.

Les résultats obtenus avec les Méta-ZCS apparaissent encourageant de prime abord. Comme on peut le voir sur la figure n° 4.1, l'agent arrive à la valeur optimale dans cet environnement (8 déplacements) très rapidement visiblement grâce aux modifications réalisées par le méta-apprentissage.

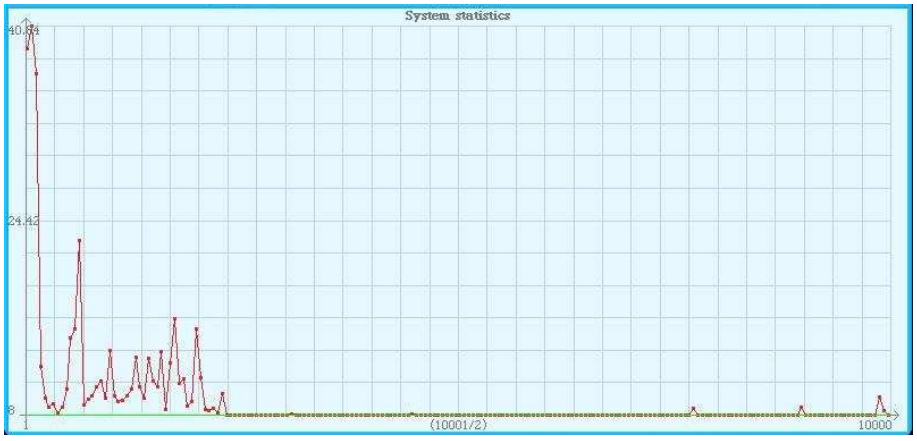


FIG. 4.2 – Résultats d'expérimentation de ZCS dans l'environnement « Maze 228 ».

En comparaison, les performances de l'architecture ZCS sont moindre car l'agent n'arrive au nombre de déplacements optimal que beaucoup plus tard. La figure n° 4.2 présente les résultats obtenus avec ZCS. Ces résultats

doivent être tout de même relativisés car aucune moyenne sur plusieurs expérimentations n'a été effectuée. Les conditions initiales pourraient ainsi influencer les résultats obtenus.

## 4.3 Expérimentations futures

De futures expérimentations sont encore à mettre en oeuvre pour valider la méthode conçue pendant ce stage. Nous souhaitons étudier l'avantage que pourrait présenter le méta-apprentissage dans un environnement non-markovien comme le « Woods 7 » de la figure n° 4.3. La difficulté de ce type d'environnement est que certaines situations ne sont pas distinguables l'une de l'autre en raison de la perception limitée de l'agent.

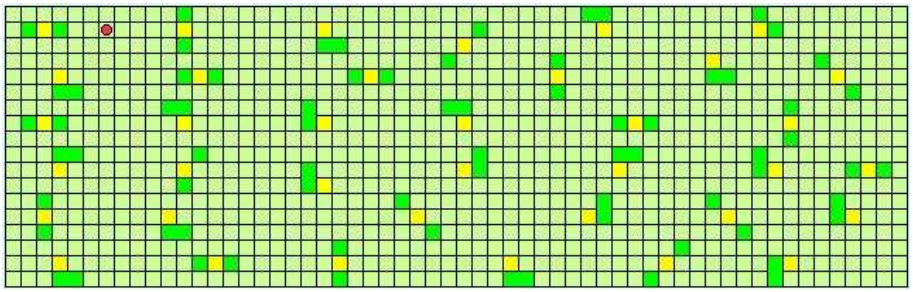

FIG. 4.3 – Représentation de l'environnement « Woods 7 ».

En plus des expérimentations mono-agent, l'étude des performances du méta-apprentissage en environnement multi-agents est essentiel pour pouvoir l'appliquer au projet MAAM. Un problème multi-agent dit « proie-prédateur » sera testé dans des environnements markovien ou non tels que les deux présentés précédemment.

Les trois agents utilisés dans ce problème sont cycliquement et deux-à-deux proies et prédateurs les uns des autres. Par exemple, l'agent $A_1$ est prédateur de $A_2$ qui est lui-même prédateur de $A_3$, ce dernier étant le prédateur du premier. Pour permettre un véritable jeu de prédation, les agents doivent posséder une perception plus large que leur rayon d'action unitaire. Ainsi, ils peuvent «voir» jusqu'à deux cases autour d'eux alors qu'ils ne peuvent se déplacer que d'une seule comme dans les expérimentations mono-agent.

En résumé, chaque architecture présentée dans l'état de l'art sera testée dans les différents cas suivant :

- système mono-agent en environnement markovien : un agent dans « Maze 228 »,
- système mono-agent en environnement non-markovien : un agent dans « Woods 7 »,
- système multi-agent en environnement markovien : trois agents dans « Maze 228 »,
- système multi-agent en environnement markovien : trois agents dans « Woods 7 ».

Un intérêt tout particulier sera porté sur l'architecture ACS qui pourrait apporter aux atomes robotiques du projet MAAM des capacités d'anticipation.

# Chapitre 5

# Conclusions et perspectives

Nous avons présenté, dans ce rapport, les bases théoriques concernant l'apprentissage par renforcement et l'apprentissage latent. Ces théories d'apprentissage animal ont été transposées dans le domaine informatique avec les systèmes de classeurs.

La limitation de leur efficacité, dans des environnements multi-agents, nous ont amené à étudier l'enrichissement de cette méthode avec celle du méta-apprentissage proposée par Schmidhuber. L'étude présentée ici répond à cette attente avec l'intégration, dans les LCS, d'actions auto-référencielles et de l'algorithme SSA.

Des résultats encourageants ont été observés et demandent à être confirmés par des études sur des moyennes de plusieurs essais. Plusieurs expérimentations restent à réaliser pour valider la nouvelle architecture sous différentes conditions. La validation du méta-apprentissage appliqué aux ACS sera particulièrement étudiée dans le but d'ajouter des capacités d'anticipation à l'architecture.

# Bibliographie

[BGS99] M. BUTZ, D. E. GOLDBERG, et W. STOLZMANN. New Challenges for Anticipatory Classifier System : Hard Problems and Possible Solution. Rapport technique 99019, ILLiGAL, 117 Transportation Building, 104 S. Mathews Avenue, Urbana, IL 61801, October 1999.

[But00] Martin BUTZ. XCSJava 1.0 : An Implémentation of the XCS classifier system in Java. Rapport technique 2000027, ILLiGAL, 117 Transportation Building, 104 S. Mathews Avenue, Urbana, IL 61801, June 2000.

[But01] Martin BUTZ. An Implementation of the Anticipatory Classifier System ACS2 in C++. Rapport technique 2001026, ILLiGAL, 117 Transportation Building, 104 S. Mathews Avenue, Urbana, IL 61801, August 2001.

[Cal02] Nicolas CALENGE. Processus décisionnels de Markov Multi-agent, Mémoire de DEA, Universite de Caen, septembre 2002.

[GS01] P. GÉRARD et O. SIGAUD. Généralisation et Apprentissage Latent dans les Systèmes de Classeurs. *Extraction des connaissances et apprentissage : Apprentissage automatique et évolution artificielle*, 3 (1) :87–114, 2001.

[Gér02] Pierre GÉRARD. *Systèmes de classeurs : étude de l'apprentissage latent*. Thèse de doctorat, spécialité informatique, Université Pierre et Marie Curie Paris 6, 2002.

[Sch96] Juergen SCHMIDHUBER. A General Method for Incremental Self-Improvement and Multi-Agent Learning. Dans X. YAO, éditeur, *Evolutionary Computation : Theory and Applications*. Scientific Publishing Company, 1996.

[Sto98] Wolfgang STOLZMANN. Anticipatory Classifier Systems. Dans John R. KOZA, Wolfgang BANZHAF, Kumar CHELLAPILLA, Deb KALYANMOYM, Marco DORIGO, David B. FOGEL, Max H. GARZON, David E. GOLDBERG, Hitoshi IBA, et Rick RIOLO, éditeurs, *Genetic Programming 3 : Proceedings of the Third Annual Conference, University of Wisconsin, Madison*, pages 658–664. Morgan Kaufmann, 1998.

[SZW96] J. SCHMIDHUBER, J. ZHAO, et M. WIERING. Simple Principle of Metalearning. Rapport technique 69-96, IDISIA, Corso Elvezia 36, CH-6900-Lugano, Switzerland, June 1996.

[ZS96] J. ZHAO et J. SCHMIDHUBER. Incremental Self-Improvement for Life-Time Multi-Agent Reinforcement Learning. Dans Pattie MAES, Maja MATARIC, Jean-Arcady MEYER, Jordan POLLACK, et Stewart W. WILSON, éditeurs, *From Animals to Animats 4 : Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior, Cambridge, MA*, pages 516–525. MIT Press, Bradford Books, 1996.

# Annexe A

# Capture d'écran de la plate-forme