Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Mohamed Khider Biskra

Faculté des Sciences Exactes, des sciences de la Nature et de la Vie

Département d'Informatique

N° d'ordre :…..……
Série :……

Mémoire

Présenté en vue de l'obtention du diplôme de Magister en Informatique

Option**: Synthèse d'images et vie artificielle**

Titre :

Comportement adaptatif d'agents dans des environnements virtuels

Par : M^elle^ TORKI Fatima Zohra

Soutenu le : 31/05/2011

Devant le jury :

DJEDI Noureddine	**Professeur**	**Université de Biskra**	**Président**
KAZAR Okba	**MCA**	**Université de Biskra**	**Examinateur**
ZIDANI Abdelmadjid	**MCA**	**Université de Batna**	**Examinateur**
CHERIF Foudil	**MCA**	**Université de Biskra**	**Rapporteur**

Résumé

Un environnement virtuel est peuplé d'agents en interaction. La création de ces environnements repose sur un principe d'autonomie des entités. Ça implique que la simulation de comportements des entités virtuelles devient un problème très complexe lorsque ces entités sont plongées dans ces environnements virtuels dynamiques et confrontées à des situations imprévues. Les agents réactifs agissent en temps réel mais ils s'adaptent difficilement à des environnements dynamiques.

Notre étude sera concentrée sur les approches issues de la vie artificielle offrant des caractéristiques d'adaptation, d'évolution et d'apprentissage. Nous créons une architecture combinant différents mécanismes afin d'augmenter les capacités de raisonnement des entités virtuelles. Cette architecture se compose de deux modèles, le premier permet de coupler fortement la perception et l'action pour fournir une réponse plus rapide à l'environnement, et il est basé sur l'approche de motor schemas qui a déjà montré son efficacité à résoudre des problèmes de navigation d'entités autonomes. Cependant, cette solution est très coûteuse. Le programmeur doit en effet définir les paramètres des schémas de façon manuelle. Pour pallier ce problème, nous définissons un modèle supplémentaire permettant aux entités autonomes d'obtenir des comportements adaptatifs basant sur un système d'apprentissage. Notre approche se base sur les systèmes de classeurs dont le rôle est d'apprendre et d'évaluer des actions correspondant aux paramètres mentionnés.

Abstract

A virtual environment is a set of agents in interaction. The creation of these environments is principal of autonomies entities. That implies that the behavioral simulation of the virtual entities becomes a very complex problem when these entities in the virtual dynamic environments and confronted with unforeseen situations. Reactive agents interact with the environment in real time, but they do not easily adapt their behaviors to dynamic environments.

Our study will precise of mechanisms stemming from the artificial life witch offering characteristics of ability to react, adaptation and evolution. We propose an architecture coupling various mechanisms to increase the capacities of reasoning of the virtual entities. This architecture consists of two layers, the first one of which allows them to couple the perception and the action to act quickly on the environment. This layer is based in the approach of motor schemas which has already shown its efficiency to resolve problems of navigation of autonomous entities. However, this solution is very expensive because the designer has to define the parameters of motor schema in a manual way. To obtain adaptive behavior we add other layer based on a system of learning as are classifier systems.

ملخـــص

البيئـــة الافتراضية هي مجموعة من الكيانات المستقلة و المتفاعلة مع بعضها البعض ، الذي يعتمد إنشاءها على مبدأ الاستقلاليـــة للكيانات؛ والذي يستـــلزم محاكـــاة (نمذجة) السلوك للكيانات الذي أصبح مشكلة معقدة جدا عندما تكون في مثـــل هذه البيئـــات الافتراضية الدينامية ومواجهة الحالات غير المتوقعة .

في بحثنا هذا ركزنا على دراسة آليات أو الميكانزمات المنبثقـــة عن الحياة الاصطناعيـــة ذات الخصائص التالية : الاستجابة ،التأقلـــم (التكيف) والتطـــور.

وعليه تشتمل دراستنا في تصميم هندســـي يجمـــع بين العديـــد من التقنيات وهذا لزيادة قدرات التفكيـــر عنـــد الكيـــانات الافتراضيـــة.والتي تتكـــون –الهندســـة– من نموذجين ، الأول يجمـــع بيـــن التصور والعمل وهذا من اجل استجابـــة سريعة في البيئـــة، تستنـــد هذه الطبقة على تقنيـــة (نمط المحرك) الذي أظهر فعاليته في حل مشاكل الملاحة للكيانات الافتراضية ومع ذلك هذا الحل مكلف جدا ، لان المصمم هو في الحقيقة الذي يحدد يدويا معايير نمط المحرك ، ومن اجل الحصول على سلوكيات المكيفة قمنا بإضافة نموذج ثاني للنموذج الأول مستندا على أنظمة التعليم وذلك لتسهيل دور المصمم فالهندسة المستعملة في هذا البحث مستندة على أنظمة المصنفـــات التعليميـــــة الذي يتمثـــل دوره في تعلـــم وتقييـــم الإجراءات المطابـــقة للمعايير المذكورة .

Table de matières

Table de figures

Liste des tableaux

Introduction générale

La simulation comportementale des entités virtuelles s'agit de modéliser les interactions qu'entretiennent les entités entre elles, avec l'environnement ou avec l'utilisateur au travers d'un avatar. Au niveau de l'individu, nous observons que chaque entité apprend, évolue et s'adapte pour résoudre le problème posé. Au niveau du groupe, les entités coopèrent afin d'accomplir une tâche collective. Cependant, le monde dans lequel les entités virtuelles sont immergées est dynamique et inconnu ; la simulation comportementale devient alors un problème très complexe. Les techniques classiques générant des agents délibératifs ne permettent pas de produire un comportement efficace face à des situations imprévues. Les mécanismes réactifs comme l'architecture de Brooks [Bro86] ou l'approche *motor schema* d'Arkin [Ark92], [Ark98] génèrent des agents réactifs : l'émergence des comportements au niveau de l'observation résulte de la combinaison pondérée de comportements de base par les poids appropriés. Les agents réactifs sont aussi capables de répondre au message entrant en temps réel. Lorsque les poids sont établis par un concepteur humain, comment les agents réactifs s'adapter aux changements de l'environnement ?. Donc, l'architecture proposée va intégrer aux modèles des agents réactifs des modèles d'apprentissage pour obtenir des comportements adaptatifs permettant d'ajuster automatiquement tous les paramètres en réponse à des situations différentes.

Pour ce type de traitement, Les systèmes de classeurs (LCS (*Learning Classifier System*)) fournissent une réponse performante aux problèmes d'apprentissage Les LCS les plus étudiés sont le ZCS, le XCS et le YCS. Ils offrent des caractéristiques de réactivité, d'adaptation et d'évolution. Les LCS couplent une base de règles qui permet d'obtenir une

réponse à un état de l'environnement, avec un mécanisme de rétribution favorisant les règles les plus productives, et un mécanisme d'évolution pour renouveler les règles de la base. L'exploration pseudo-aléatoire des solutions entraîne la création de règles novatrices pouvant s'améliorer en parallèle avec l'évolution de l'environnement.

Les facultés d'apprentissage, couplées à des capacités d'évolution, font des systèmes de classeurs une méthode robuste pour la génération de comportements, et une méthode adaptative face à toute éventualité comme une modification dans le déroulement de la simulation. En effet, sans connaissance initiale, un tel système construit lui même les liens entre les capteurs et les effecteurs en utilisant uniquement le résultat de l'évaluation des classeurs qu'il a déclenchés.

Les applications des LCS sont très nombreuses : la simulation d'entités virtuelles, la robotique [KY03], l'économie [GRS04], [Won07], le commerce [BHL04] et le contrôle routier [BST+04].

Plan du mémoire

Afin de donner un aperçu des différents sujets qui d'une manière où d'une autre contribue à la génération des comportements autonomes, et de pallier la difficulté pour que le concepteur de définir les paramètres qui permettent de simuler ces comportements, et aussi pour mettre en valeur la contribution de ce travail, nous organisons cette thèse de la manière suivante :

Le premier chapitre recentre le sujet sur les différents systèmes de simulation comportementale existants et les différents types d'agents et analysé leurs atouts et faiblesses.

Nous verrons ensuite, dans le deuxième chapitre les principaux paradigmes de la vie artificielle ; après une brève introduction aux algorithmes génétiques, nous décrivons le fonctionnement interne des systèmes de classeurs. Nous effectuons ensuite un tour d'horizon des principales améliorations et variantes apportées aux systèmes de classeurs depuis quelques années. Ce chapitre sera conclu par une étude comparative prenant en considération les points communs entre les différents types et les points qui leur diffèrent.

Le troisième chapitre traite notre contribution qui est la simulation comportementale des agents dans un environnement virtuel. Nous présenterons notre modèle d'apprentissage, qui couple différents mécanismes.

Le dernier chapitre permet d'évaluer notre architecture et de tester ses performances à travers une application coopérative pour transporter un objet et de présenter les résultats obtenus. Nous montrons les capacités d'adaptation d'un agent doté d'un système d'apprentissage (système de classeurs).

Chapitre 1 : Agents Autonomes et Simulation Comportementale

Chapitre1

Agents autonomes et Simulation comportementale

Tu me dis, j'oublie.
Tu m'enseigne, je me souviens.
Tu m'implique, j'apprends.

BENJAMIN FRANKLIN

La simulation comportementale, présentée dans la section 1, est une discipline récente faisant partie de l'animation et astreinte à la recherche des comportements pour les entités (agents) et qui peuplent les environnements virtuels. Ces agents doivent être avant tout réalistes et accomplir de manière autonome des objectifs assignes par l'animateur (le concepteur). Pour ce faire, ils possèdent un contrôleur, en charge du mécanisme de sélection de l'action. En s'inspirant de disciplines variées, les agents mettent en œuvre différentes approches : on trouvera, des sections 3.1 à 3.4, les agents délibératifs, réactifs, évolutionnistes et hybrides. Pour mieux comprendre le rôle de l'environnement dans les comportements des agents virtuels, nous présenterons les types des environnements ainsi que les techniques d'adaptation qui sont déjà appliquées avec succès.

1. La simulation comportementale

La simulation comportementale est une partie de l'animation qui se rapproche des systèmes réels de par son principe de fonctionnement en assignant aux entités (agents) ou systèmes animés des comportements indépendants. Ces derniers ne seront alors plus régis par un système global gérant le mouvement de toutes les entités mais par un

mécanisme de décision local placé dans chaque entité. Chaque entité de la simulation prendra donc les décisions comportementales concernant son mouvement au pas de temps suivant, selon son état et celui de l'environnement l'entourant à cet instant de la simulation. La simulation comportementale est donc un moyen de faire interagir de manière naturelle des entités en simulant leurs capacités dans un environnement.

Terzopulos [TTG94] a été le premier à définir un modèle comportemental. Un découpage en trois éléments distincts : Capteurs, Module comportemental et Effecteur lui ont permis de réaliser des animations réalistes de bancs de poissons naviguant au milieu de vestiges de l'ancienne Rome. Depuis ce modèle a été repris de nombreuses fois ([Bec98], [SDD99]…) et est quasiment devenu un standard. Chaque individu d'une simulation modifie son comportement en fonction de données fournies par des capteurs et retransmet ses actions à l'environnement au travers des effecteurs. L'utilisation exclusive de ce couple acteurs/effecteurs limite la connaissance et l'effet d'un individu à leurs seules capacités d'acquisition et d'action.

2. Définitions usuelles

2.1 Agents autonomes

Dans la littérature, les définitions des *agents* sont diverses. Les différences relèvent du domaine d'application et du degré de complexité.
D'un point de vue étymologique, le mot *agent* vient du latin « *agere* » qui signifie agir littéralement, l'agent est donc celui qui agit.

Jacques Ferber [Fer95] définit un agent comme étant une entité physique ou virtuelle évoluant dans un environnement dont il n'a qu'une représentation partielle et sur lequel il peut agir. Il est capable de communiquer avec d'autres agents et est doté d'un comportement autonome. Cette définition aborde une notion essentielle : l'*autonomie.* En effet, ce concept est au centre de la problématique des agents. L'autonomie est la faculté d'avoir ou non le contrôle de son comportement sans l'intervention d'autres agents ou d'êtres humains.

P. Maes [Mae94] définit un agent autonome adaptatif.

- Un agent est une entité qui a comme tâche d'accomplir un ensemble d'objectifs dans un environnement dynamique et complexe.

- Un agent est autonome s'il peut prendre ses propres actions par rapport aux perceptions dans l'environnement pour satisfaire ses objectifs et ses croyances avec un degré de satisfaction.
- Un agent est adaptatif s'il est capable d'apprendre de son environnement, d'évoluer pour s'améliorer dans le temps, c'est à dire s'il est meilleur dans l'accomplissement de ses objectifs avec l'expérience.

Les travaux de J. Ferber [Fer95] et P. Maes [Maes94] proposent les premières notions d'agent et introduisent les concepts d'autonomie et d'adaptation dans des systèmes complexes et dynamiques.

Les agents autonomes désignent des entités plus précises ; un agent autonome est situé, au sens de la définition de Russel et Norvig [RN95]. *[«Un agent est toute chose qui peut être vue comme percevant son environnement à travers des capteurs et agissant sur cet environnement par le biais d'effecteurs »]*. Cette définition peut être étendue pour la plupart des logiciels. Au contraire, la définition de Coen[1] [Coe95] apparaît beaucoup plus limitée.

Pour Mickael Wooldridge [WJ99], un agent est un système informatique capable d'agir de manière autonome et flexible dans un environnement

Franklin et Grasser[FG96] présente la définition suivante "un agent autonome est un système faisant partie d'un environnement dans lequel il est situé, il perçoit cet environnement et agit sur lui au cours du temps, en fonction de son propre agenda et de façon à modifier sa perception future**.**

Nadin Ridchard [Ric01] voit que l'agent est une entité active et autonome, qui interagit avec un environnement dynamique, et qui peut être intelligente, adaptative ou sociable.

Enfin, Le terme d'agent est couramment utilisé en informatique et lui trouver une définition qui puisse satisfaire à toutes les disciplines relève de la gageure. Nous simulons un agent autonome, Son comportement est basé sur ses perceptions mais aussi sur des connaissances à priori ou acquises lors de l'interaction ou la communication avec d'autres

[1] *"Les agents logiciels sont des programmes qui dialoguent, négocient et coordonnent le transfert d'informations".*

agents ou son environnement. dont les propriétés seront énoncées dans la section suivante, et y est incarnée par une représentation graphique.

2.1.1.Les propriétés d'agents

Un grand nombre de propriétés permettent de définir un agent. David Panzoli donne dans sa thèse [Pan08] la liste des principales propriétés suivantes :

- L'autonomie permet à l'agent de sélectionner les actions qu'il va accomplir sans intervention extérieure (et en particulier humaine). Pour cela, il doit posséder une représentation du problème, de son environnement ainsi que de son état interne.
- La réactivité donne le temps de réponse de l'agent lorsqu'il est confronté à un problème.
- L'intentionnalité permet à l'agent de produire des plans à partir de buts explicites.

  Elle peut s'opposer à la propriété de réactivité si le temps de production de ces plans est trop long.
- La sociabilité est la capacité de l'agent à communiquer avec les autres agents de l'environnement
- La situation dénoté l'importance de l'environnement pour la définition du comportement de l'agent.
- L'adaptabilité est l'aptitude de l'agent à s'adapter aux modifications de son environnement.

2.2. L'approche «orientée-comportements»

En l'Intelligence Artificielle (IA) classique, les différents modules d'un système intelligent sont organisés verticalement, par grandes fonctions telles que la vision, la perception, la construction de la représentation du monde ou encore la planification. Les flux de données circulent selon un unique cycle *«perception-décision-action»*, qui ne garantit pas la rapidité de réponse du système. L'approche «orientée-comportements» propose une organisation horizontale, dans laquelle chaque module correspond à un cycle perception-décision-action spécialisé dans la réalisation d'une fonctionnalité [Bro89].Ce type d'architecture distribuée permet d'organiser des modules qui réagissent indépendamment les uns des autres à des

aspects précis de l'environnement. Il ne repose pas sur un système centralisé de perception, chargé d'interpréter les signaux en provenance de l'environnement et de les traduire en données symboliques : chaque module perçoit uniquement les stimuli qui le concernent. De plus, les modules ne partagent pas de représentation centralisée de l'environnement ou de l'état de l'agent : chaque module manipule exclusivement ses données propres. L. Steels définit un *comportement* d'un système (agent ou groupe) comme une «régularité» observée dans la dynamique d'interaction entre le système et son environnement.

Le comportement est qualifié d'intelligent s'il permet au système de survivre au mieux dans son environnement [Ste94]. L'approche «orientée-comportements» tente de dégager des motifs représentatifs de l'interaction d'un agent avec son environnement et de les exprimer sous la forme de règles comportementales.

Le problème de la perception et de l'action dans un environnement physique, ou tout au moins finement simulé, est important. Dans certains cas, le comportement peut émerger, c'est-à-dire être observé sans pour autant avoir été décrit explicitement sous forme de règles. L'adaptabilité et les capacités d'apprentissage sont souvent mises en exergue, car elles améliorent l'intelligence en permettant au système d'évoluer dans un environnement dynamique qu'il doit explorer.

Pour décrire les architectures basées sur les comportements, L. Steels distingue les comportements (définis précédemment), les systèmes comportementaux et les mécanismes comportementaux d'un agent. *Les systèmes comportementaux* sont des modules capables d'assurer une *fonctionnalité* particulière pour l'agent, comme la locomotion, l'évitement d'obstacle ou la recherche de nourriture. Pour être efficace, un système comportemental doit être adapté à l'environnement et aux objectifs de l'agent.

Une fonctionnalité est réalisée par un ou plusieurs comportements, et un système comportemental peut contribuer à la réalisation de plusieurs fonctionnalités ; il n'y a donc pas de relation directe entre un comportement et un système comportemental.

Un système comportemental est constitué d'un ensemble de *mécanismes comportementaux*. Un mécanisme comportemental est un principe ou une technique permettant de créer un comportement particulier, par exemple en couplant des capteurs et des effecteurs ou en construisant une carte cognitive. Un tel mécanisme est implémenté en

utilisant des structures et des processus (capteurs, effecteurs, programmes, structures de données, *etc*.). Les mécanismes comportementaux doivent être les plus simples possibles, afin que ce soient les interactions entre ces mécanismes qui fassent émerger le comportement attendu.

Pour résumer, un comportement est ce qui est observé, alors qu'un système comportemental est ce qui est implémenté dans l'agent par des mécanismes comportementaux de façon à faire émerger le comportement.

Principe de sélection de l'action

Produire le comportement d'un agent revient à définir et modéliser ses interactions avec l'environnement. Selon le modèle général défini dans [TT94], et schématisé dans la (figure1.1), celles-ci sont synthétisées à travers une boucle **perception→décision→action**. En prenant en compte le principe d'autonomie de l'agent, c'est lui qui est responsable de gérer la perception et l'action. Il a aussi la charge de la phase de décision, ce que l'on résume à travers le terme de sélection de l'action. A cet effet, il est muni d'un système appelé contrôleur comportemental.

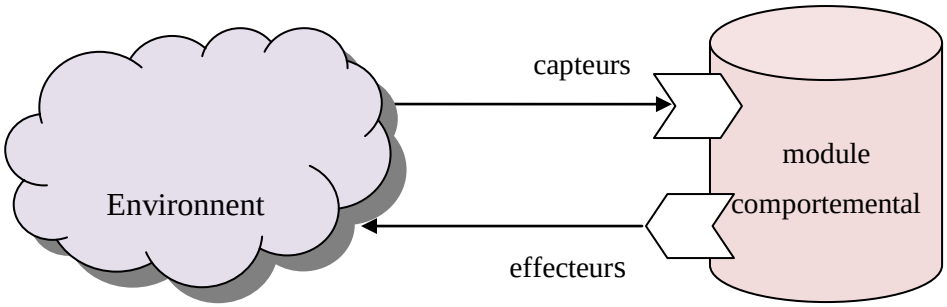


Figure 1.1 : Le modèle général comportemental tel que défini dans [TT94]

La boucle perception→décision→action est répétée à l' infini, ou du moins tant que la simulation n'est pas arrêtée. Elle consiste en 3 phases :

La phase de perception représente le moyen d'extraire des informations sur l'environnement dans le quel l'agent évolue et les transformer en valeurs exploitables, elle est effectuée à partir de capteurs plus ou moins spécialisés dans l'extraction de caractéristiques perceptives particulières. On trouve des capteurs réalistes comme des capteurs de vision qui reprennent le fonctionnement de l'œil ou des capteurs de collision qui simulent le sens du toucher, A

l'inverse, on peut utiliser des capteurs imaginaires comme un capteur de position absolue dans l'espace, un capteur de distance à un objet particulier, etc..

La phase de décisions : qui a pour but de prendre en compte la perception précédemment effectuée pour choisir la meilleure action à effectuer en fonction du but de l'agent, de son environnement perceptible et de son état interne,

La phase d'action : d'après J. Ferber [Fer95], l'action est en premier lieu ce qui modifie l'état global du monde qui est prise en charge par des effecteurs. Ils agissent dans l'environnement en traduisant chaque action décidée par le processus de décision en un changement d'état dans l'environnement.

La grande majorité des agents utilisés actuellement en simulation comportementale possède ce principe de fonctionnement.

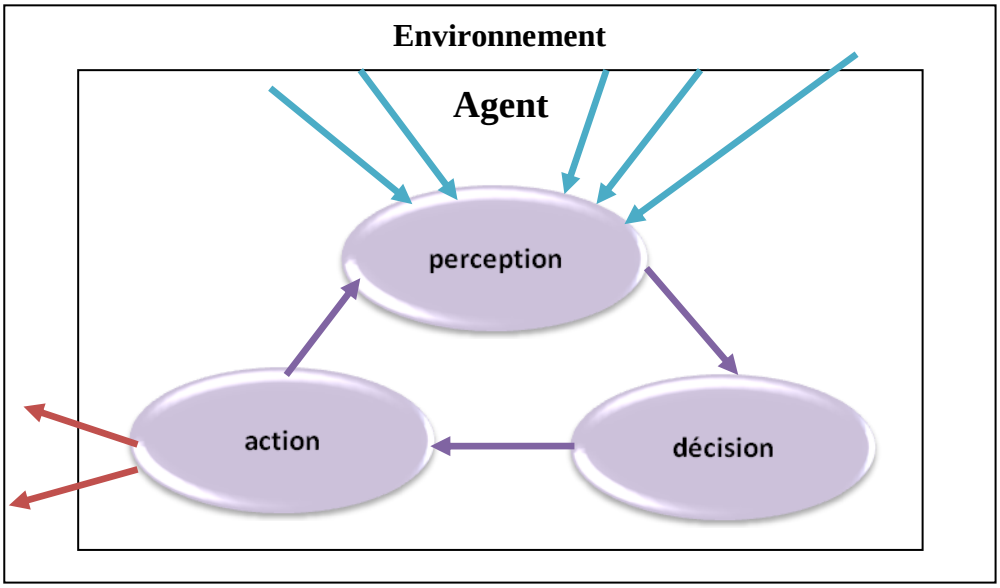


Figure 1.2 : La boucle Perception-Décision-Action d'un agent virtuel lui permet de sélectionner la meilleure action en fonction de son environnement local et de son but propre.

Le déroulement du cycle perception-décision-action se déroule de la façon suivante: l'unité de traitement (module comportemental (décision)) reçoit les informations sur l'environnement à l'aide des capteurs (perception), ces informations sont analysées afin de prendre une décision appropriée, l'action choisie est alors exécutée par les effecteurs (action).

L'architecture modulaire d'un agent (entité autonome agissant sur l'environnement) favorise l'amélioration, l'ajout et la réutilisation des modules principaux, chaque module est responsable de la réalisation d'une fonction dont le résultat sera utilisé par le module suivant.

3. Les types d'agents et les architectures comportementales

3.1. Les agents délibératifs

L'approche délibérative (appelée aussi cognitive) est issue de l'intelligence artificielle (IA). Elle se base sur des algorithmes de planification afin de trouver la suite d'actions à effectuer en fonction des buts courants de l'agent. La figure 1.3 montre la partie décisionnelle d'un agent délibératif.

La couche modélisation permet à l'agent de créer un modèle cognitif de son environnement. Leur principe a été modélisé par John D. Funge en 1999 [FTT99]. La planification est la phase de délibération de l'agent. Donc il utilise une représentation interne de son environnement afin de formuler des plans d'actions pour satisfaire ses buts. Ceci est généralement réalisé à l'aide d'un moteur d'inférences fonctionnant sur le principe suivant:

état courant du monde + but → plan

Le principal intérêt des *agents délibératifs* est leur capacité à pouvoir former des plans à long terme en anticipant plus ou moins les états futurs de l'environnement

De nombreuses techniques existent telles que le raisonnement a partir de cas [OMS+ 07], la logique floue, les systèmes experts, etc. Ces deux couches sont souvent très couteuses en temps de calcul et il est difficile de garantir leur temps d'exécution. Il est de plus obligatoire de les relancer intégralement après l'exécution d'une ou plusieurs d'actions.

Comme nous l'avons vu précédemment, la sélection de l'action est le point faible de ce type de modèles. L'incapacité de choisir une bonne action dans un court temps de calcul ne permet pas leur intégration facile dans des systèmes temps réels tels que les robots car ils doivent parfois prendre des décisions rapides afin de protéger leur intégrité.

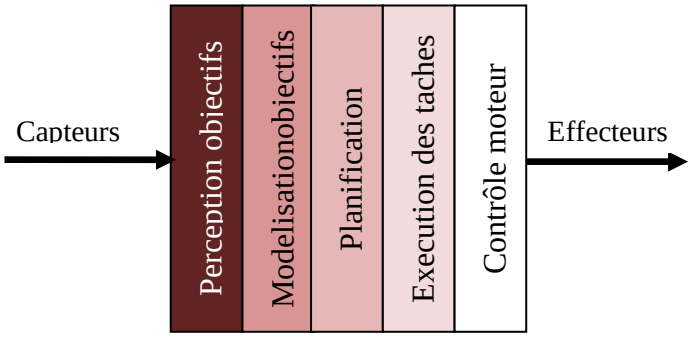


Figure 1.3 : L'architecture en couche de l'agent cognitif lui permet d'avoir une représentation du monde avant de faire une planification dessus qui lui permettra de choisir la ou les actions à déclencher [FTT99].

3.2. Les agents réactifs

Afin de pallier le manque de robustesse, de réactivité et d'adaptabilité des *agents cognitifs*, des chercheurs ont proposé, dès le milieu des années 80, des agents plus simples, n'utilisant pas de représentation interne de leur environnement et prenant leur décision de façon réflexe à partir d'un stimulus. En effet, d'après Brooks [Bro86], un comportement intelligent peut être généré sans représentation symbolique explicite ni raisonnement abstrait: il émergera des interactions de l'agent avec son environnement complexe. Ainsi, un *agent réactif* est un agent qui, pour chaque configuration de ses capteurs, sans passer par la phase de planification à partir d'une représentation interne de son environnement, déclenchera une action prédéfinie. Ce type d'agent peut être implémenté à l'aide d'un système de règles simples, ou d'un automate, selon le principe suivant:

état courant du monde → action

L'avantage de l'approche réactive est que le calcul des comportements de base est rapide et ne requiert pas de représentation interne de l'environnement. Parce que la représentation interne doit être mise à jour régulièrement lorsque les changements dans l'environnement apparaissent, cela ralentit le temps de réponse.

3.2.1. Historique des champs de potentiel

Les méthodes traditionnelles pour des problèmes de navigation sont basées sur les représentations symboliques et les modèles internes du monde. Les systèmes reçoivent l'information sensorielle de leurs détecteurs, puis la convertissent en un niveau élevé de la représentation symbolique de l'environnement. L'information transformée est utilisée pour

mettre à jour leurs modèles internes et le *pathfinding* (la planification de chemin) est basé sur ces derniers. L'avantage de ces approches traditionnelles est que les systèmes sont très efficaces dans des environnements connus, mais leurs performances diminuent dans des environnements dynamiques qui nécessitent de fournir des réponses réactives comme l'environnement du monde réel.

Les méthodes les plus souvent utilisées dans le domaine de la robotique mobile sont celles faisant appel aux champs de potentiel. Les champs de potentiel ont été développés tout d'abord comme une approche d'évitement d'obstacles en temps réel et présentés par Latombe [Lat91]. Le robot a pour objectif d'atteindre une cible sans toucher les obstacles qui peuvent jalonner son parcours. Pour cela, Latomb a introduit deux types de champs de potentiel : le champ de potentiel attractif induit par la cible à atteindre et les champs de potentiel répulsifs induits par les obstacles à éviter. Formellement ceci est défini par une fonction différentiable U appelée fonction potentiel qui à tout point de l'espace de l'environnement $\Re^n$ (ici $\Re^2$ ou $\Re^3$) fait correspondre un scalaire. La fonction potentielle est construite par la somme de deux fonctions potentielles élémentaires U_{att} et U_{rep} où U_{att} est une fonction attractive associée à la cible à atteindre et U_{rep} est la fonction répulsive correspondant aux obstacles à éviter. On définit donc :

$$U = U_{att} + U_{rép}$$

A partir de ce champ de potentiel, le champ des forces artificielles $\vec{F}$ est défini par :

$\vec{F}(p) = -\vec{\nabla}U(p)$ où $\vec{\nabla}U(p)$ est le vecteur gradient de U au point $p \in \Re^n$

D'où l'on déduit les forces attractives et répulsives :

$$\vec{F}_{att} = -\vec{\nabla}U_{att} \text{ et } \vec{F}_{répt} = -\vec{\nabla}U_{rép}$$

Suivant la position p du robot, cette force $\vec{F}$ lui indiquera alors la direction la plus prometteuse à prendre pour son prochain déplacement.

3.2.2. Approche de subsomption de R.Brooks

[Bro86] décompose le problème du contrôle d'un robot mobile en *niveaux de compétence*

dont chacun définit un comportement pour le robot mobile, par exemple : *Evitement d'obstacles*, *Vagabondage*, *Exploration*… (figure 1.4). Un niveau supérieur implique un comportement plus spécifique. Le principe de l'architecture de subsomption est qu'on peut

ajouter des niveaux supérieurs dans les niveaux actuels sans devoir modifier ces derniers. Les niveaux supérieurs peuvent manipuler les entrées et sorties des niveaux inférieurs (figure 1.5).

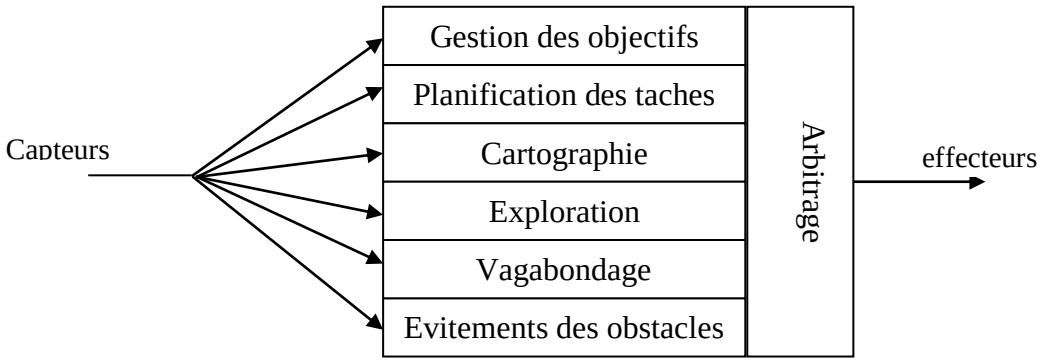


Figure 1.4 : Décomposition du contrôle d'un robot mobile en comportements [Bro86].

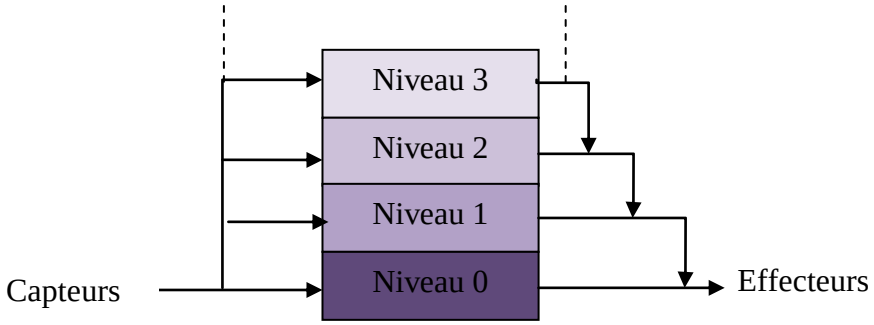


Figure 1.5 : Architecture de subsomption. Le contrôle est décomposé en niveaux de compétence. Un niveau supérieur subsume le rôle des niveaux inférieurs s'il veut prendre le contrôle [Bro 86].

Le regroupement de ces comportements forme des niveaux de compétence. Ces systèmes sont hiérarchisés selon le principe de subsomption: les niveaux supérieurs manipulent les entrées et sorties des systèmes inférieurs, les contrôlant de manière indirecte. Pour cela Brooks à introduit deux opérateurs (figure1.6) : un opérateur *d'inhibition* et un opérateur de *suppression.* L'opérateur de suppression agit sur les entrées des niveaux inférieurs en bloquant l'accès à certains capteurs pour les remplacer par ses propres données. Grâce à cette méthode, il contrôle l'information sur lequel travaille le niveau inférieur. Le deuxième opérateur, l'inhibition, travaille sur les sorties des niveaux

inférieurs en empêchant la diffusion de messages sur la sortie du comportement. Ces deux mécanismes sont dits de subsomption.

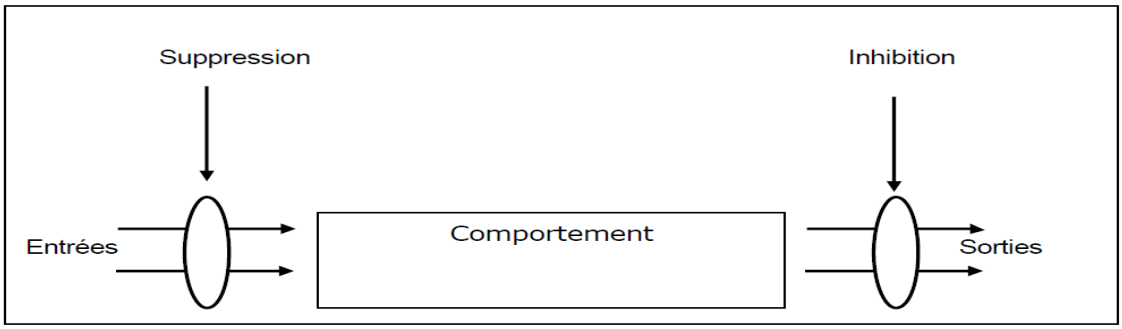


Figure 1.6: Opérateurs de subsomption [Bro 86].

La figure 1.7 montre un exemple d'architecture de subsomption pour le contrôle du robot mobile avec trois niveaux de compétence (*avoid objects*, *wander* et *explore*).

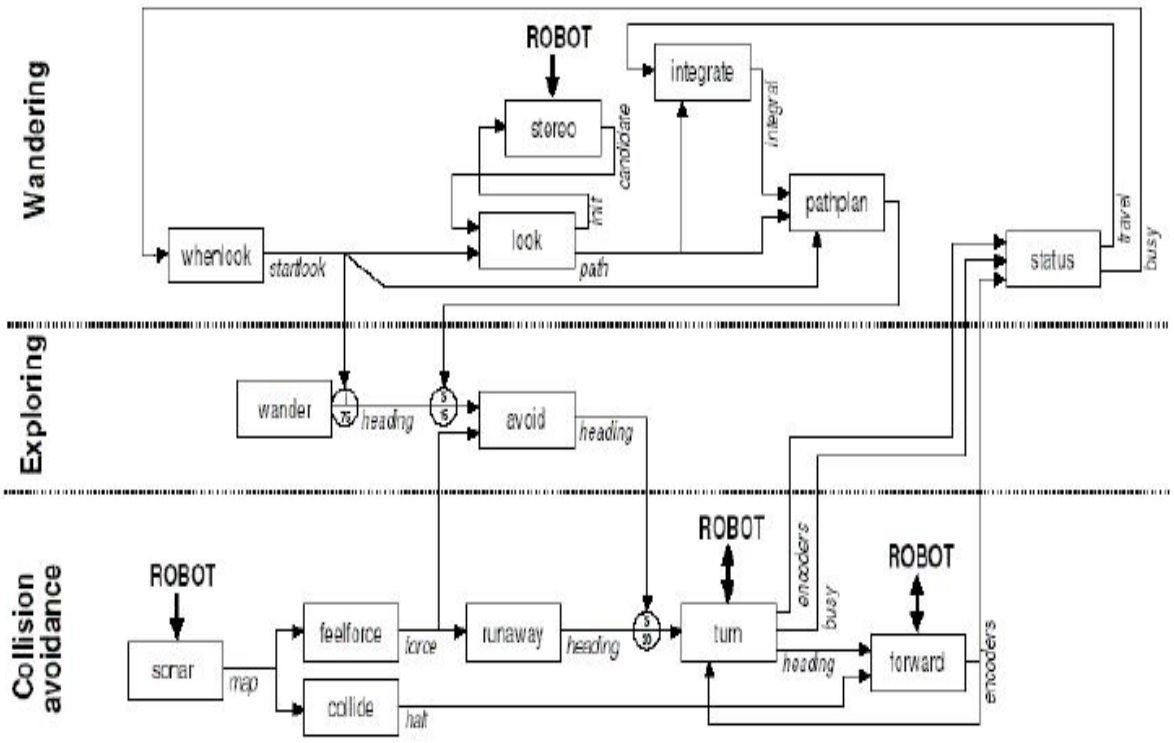
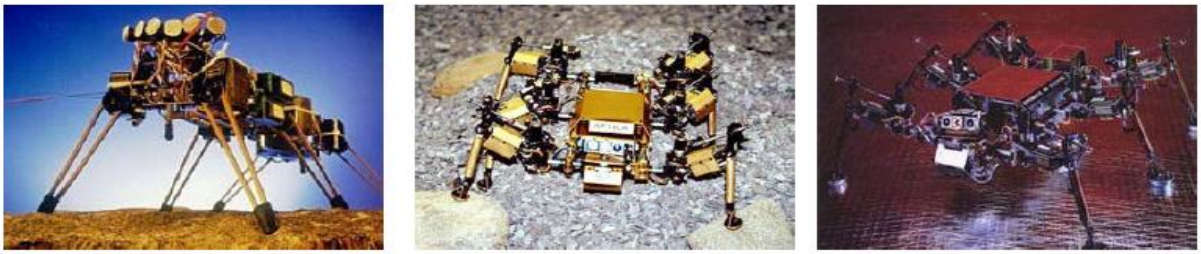


Figure 1.7 : Opérateurs de subsomption. Le temps est noté dans le cercle [Bro 86].

Ce modèle a été implante avec succès sur de nombreux autres robots (figure 1.8) mais a très vite dévoile quelques points faibles :

- Le manque de modularité due aux interpénétrations des niveaux de compétences rend le système difficile à faire évoluer. En effet, il est tr`es difficile de décrire des comportements totalement indépendants les uns des autres.
- Les problèmes lies à la hiérarchisation des modules au sein d'un niveau de compétence, qui n'est pas toujours triviale.
- La complexité de définir une durée pour chaque opération de subsomption.



(a) Genghis (b) Attila (c) Hannibal

Figure 1.8 : Les robots de Rodney Brooks [Bro86].

3.2.3. Boids

Dès 1986, C. Reynolds [Rey87] a proposé une architecture pour la simulation des comportements de groupes d'entités autonomes appelées Boids. Ces entités sont des tortues simplifiées, caractérisées par une position, une orientation et une vélocité. Elles sont capables d'avancer ou de reculer en ligne droite, et de tourner à gauche ou à droite. Chaque boid a accès à la position, à l'orientation et à la vélocité de tous les autres objets du système (boids et obstacles). Cette perception n'est pas réaliste mais elle permet d'obtenir des animations très fluides. Les voisins d'un boid sont les boids présents dans une zone sphérique centrée sur le repère d'origine du boid. Le calcul de la vélocité à appliquer au prochain pas de temps à chacun des boids est effectué par sommation des résultats de chacune des règles.

Les règles définissant les comportements de base sont:

- la cohésion qui permet de maintenir les *boids* en formation,
- la séparation qui permet d'éviter les agrégats et les collisions,
- l'alignement permet d'uniformiser les déplacements de plusieurs entités.

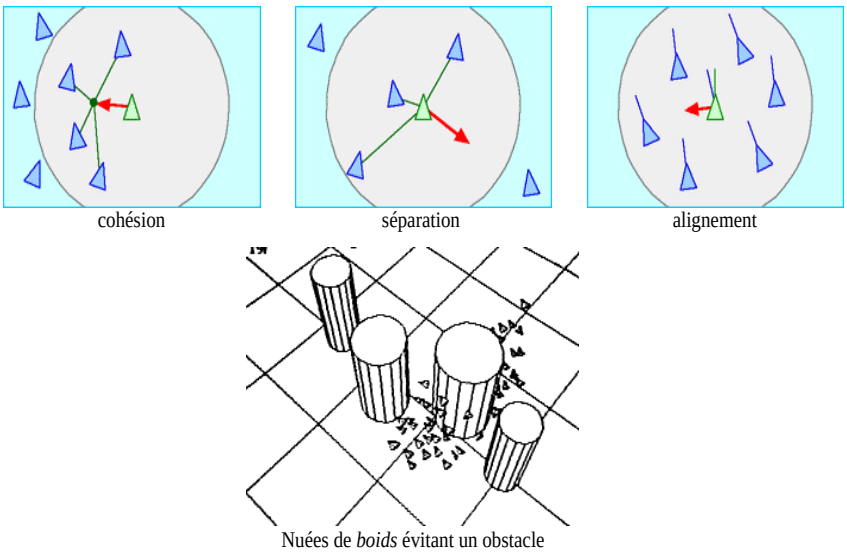


Figure 1.9: Règles de déplacement des *boids* et simulation de nuées d'oiseaux par des *boids*[Rey87].

Cette technique relativement basique pour l'objectif de faire émerger le comportement du groupe à partir des réactions de chacun de ses éléments vis-à-vis de son environnement proche, et non pas de contrôler l'animation globalement. En 1999, C Reynolds modifie son modèle pour contrôler le déplacement d'entités autonomes [Rey99]. Il étend la base de règles afin d'obtenir des comportements de proie/prédateur, de suiveur dans un groupe, ou plus simplement d'évitement d'obstacles. Les comportements s'insèrent dans une structure hiérarchique permettant l'activation de combinaisons prédéfinies des règles en fonction de la situation et du comportement global désiré pour une entité spécifique. Il choisit d'affecter des priorités strictes aux différentes règles comportementales. Les comportements moins prioritaires pourront être ignorés au profit des comportements devant réagir dans l'urgence ; par exemple, la règle de centrage par rapport au groupe sera ignorée le temps d'éviter un obstacle. Dans [Rey99], C. Reynolds présente un ensemble de règles comportementales supplémentaires, permettant de simuler des comportements plus évolués. Un *boid* peut alors fuir ou poursuivre un autre *boid*, ou encore suivre un *boid* considéré comme le chef du groupe.

3.2.4. L'architecture ascendante de P. Maes

L'architecture ascendante de P. Maes [Mae89] s'appuie sur une sélection de l'action qualifiée d'ascendante. A partir de comportements de base d'un agent (s'approcher d'une

source, boire…), on établit un réseau non hiérarchique de nœuds d'actions. Un seul nœud peut être exécuté simultanément sachant que, lors de l'activation, plusieurs nœuds peuvent
être sélectionnés. Le choix se porte alors sur le nœud de niveau le plus élevé, ce niveau décroissant automatiquement à chaque pas de réaction de l'agent.
Un nœud perçoit l'environnement de l'agent sous forme de pré-conditions correspondant à la présence ou à l'absence de certaines caractéristiques de l'environnement au moment de leur évaluation. Un nœud est éligible seulement si toutes ses pré-conditions sont satisfaites, alors que tous les nœuds participent au choix du nœud à exécuter. Chaque nœud calcule son niveau d'activation en fonction des motivations de l'agent et de sa connexion avec d'autres nœuds. Ces connexions permettent de propager une excitation ou une inhibition, c'est-à-dire d'augmenter ou de réduire le niveau d'activation d'autres nœuds.

Cette approche originale, illustrée sur la figure 1.10, a été mise en œuvre et critiquée par T. Tyrrell [Tyr93]. D'après lui, le mécanisme est efficace lorsqu'il s'agit d'interrompre un comportement en cas d'urgence ou pour profiter des opportunités. Par contre, il ne satisfait pas la caractéristique de persistance et de préférence des comportements consommatoires. De plus, il n'est pas efficace avec un grand nombre nœuds et d'objectifs conflictuels, en particulier parce qu'aucun mécanisme ne permet de combiner les actions des comportements**.**

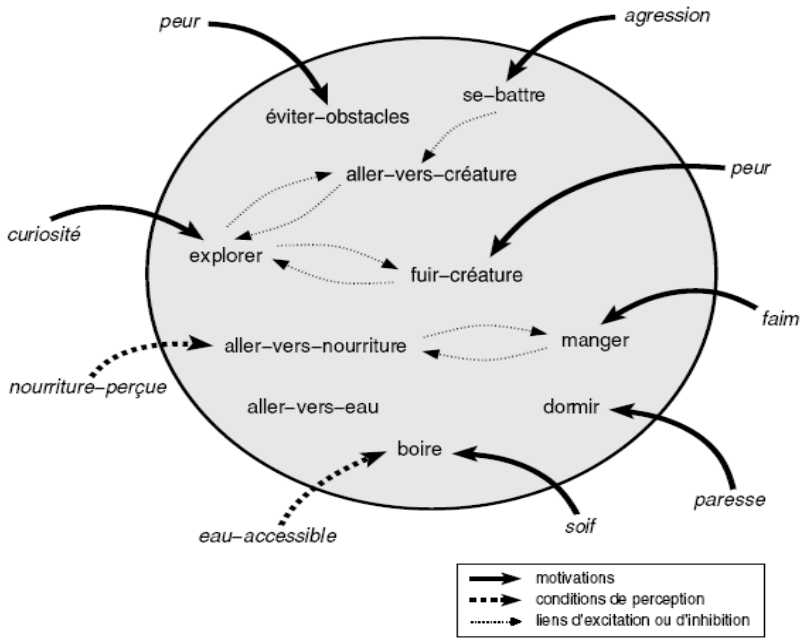


Figure1.10 : Exemple d'architecture ascendante de P.Maes [Mae89].

3.3. Les agents évolutionnistes

Les agents évolutionnistes ne proviennent pas de l'Intelligence Artificielle (IA) mais de la vie artificielle (VA), une discipline ayant pour but d'´etudier et de reproduire des mécanismes du vivant en informatique. Parmi ces mécanismes, il en est un qui est largement exploité par la VA : les méthodes évolutionnistes [Hol75, GR89] sont une application de la théorie de l'´evolution par sélection naturelle de Charles Darwin à la résolution de problèmes. Contrairement aux méthodes analytiques, qui nécessitent de représenter la démarche permettant de trouver une solution, l'approche évolutionniste se contente de décrire le problème puis de laisser la solution émerger par un processus d'´evolution simulée.

Fondamentalement, cette description repose sur deux compétences. D'abord la modélisation d'une solution générique, ensuite, la spécification des contraintes qui permettent de mesurer la pertinence (on parle de *fitness*) de chaque solution par rapport au problème initial. .

Dans le cas de l'animation comportementale, il suffit de considérer le comportement désiré comme la définition du problème, et modéliser un contrôleur générique. Par pression de l'environnement sur les populations successives d'agents, un comportement adéquat doit émerger.

L'intégration de travaux de la vie artificielle (VA) en animation a été initiée par les travaux de Karl Sims. Des travaux préliminaires [Sim91] sur la production d'images avaient laissé penser que les techniques évolutionnistes répondaient parfaitement à l'attente de l'animation en termes de réalisme. Ses Creatures [Sim94a, Sim 94b] qui font encore référence aujourd'hui, sont basées sur l'évolution génétique de l'agent dans un environnement physique. Il en résulte des mouvements incroyablement réalistes et l'impression d'observer des créatures véritablement vivantes.

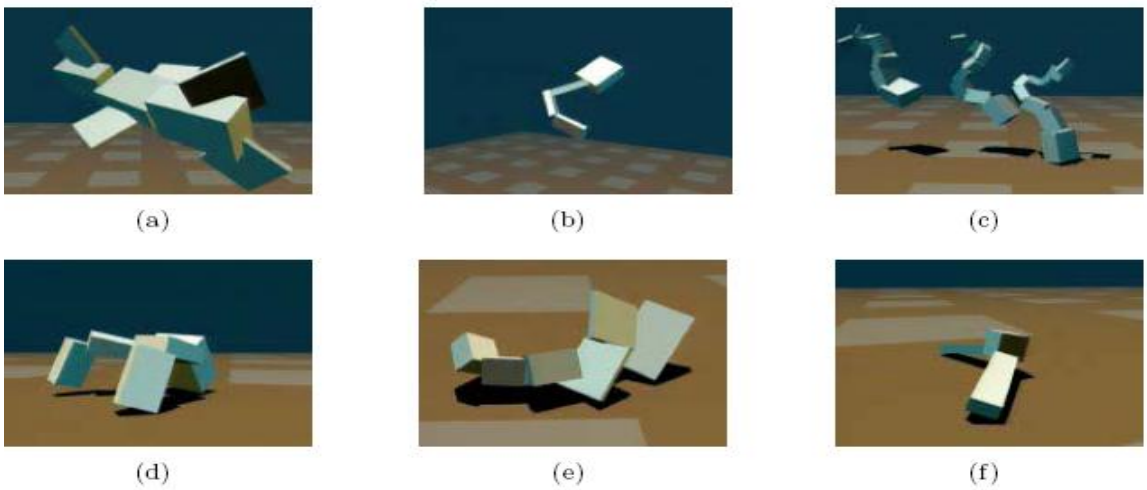


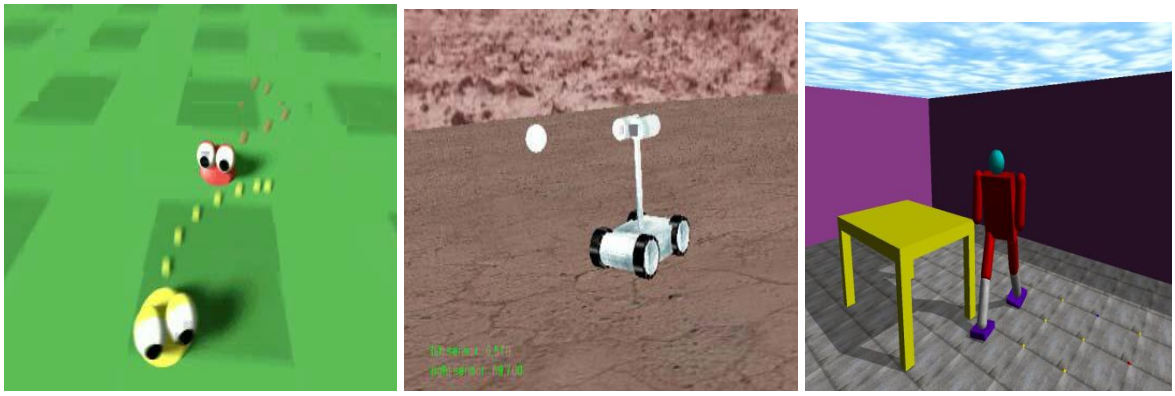
Figure 1.11 : Les créatures de Karl Sims sont capables de se déplacer aussi bien dans l'eau que sur terre car elles ont évolué dans ce but.

Le même principe d'évolution est alors appliqué à la production du comportement. En plus de fabriquer des créatures qui se déplacent de manière réaliste, on cherche désormais à produire des créatures qui se comportent de manière cohérente [SLD08, Syl09]. Parmi l'ensemble des techniques compatibles avec l''evolution génétique, deux ce sont révélées remarquablement adaptées a la production de comportements : les réseaux de neurones et les systèmes de classeurs.

Dave Cliff et Geoffrey Miller [CG95, CG96] utilisent l''evolution pour obtenir des comportements simples de poursuite et de fuite (voir la figure 1.12(a)). Les deux créatures sont contrôlées par des réseaux de neurones par une évolution génétique. Un point important de ce travail, il s'agit d'une coévolution, ce qui signifie que l'évolution ne concerne pas chaque agent dans son environnement mais les 2 agents, l'un par rapport à l'autre.

Panzoli [Pan03], a également faire évoluer génétiquement des réseaux de neurones pour le contrôle d'agents sur un plan (figures 1.12(b)).

Ouannes [ODD+09] a également faire coupler les réseaux de neurones avec l'algorithme génétique pour contrôler un robot humanoïde de faire évoluer ses capacités à se déplacer dans l'environnement simulé (i.e un comportement de locomotion) (figure (1.12(c)).



(a) Cliff et Miller font co-évoluer 2 créatures. La jaune doit apprendre à fuir, la rouge `a attraper la jaune.

(b) Panzoli fait évoluer un agent contrôlé par un réseau de neurones,qui doit être capable suivre une lumière.

(c) un robot de Ouannes.

Figure 1.12 : Quelques agents évolutionnistes : le contrôleur est un réseau de neurones que l'on fait évoluer génétiquement.

Les systèmes de classeurs [Hol75, GR89] (voir chapitre2) sont des architectures basées sur les algorithmes génétiques et dédiées à l'apprentissage de règles de type condition→action. Un système de classeurs contient plusieurs règles, lesquelles sont capables d'´evoluer génétiquement, donc, il constitue le contrôleur de l'agent.

Il existe littéralement des dizaines d'agents contrôles par un système de classeurs. Cédric Sanza [San01] (figure 1.13 (b)) puis Olivier Heguy [Heg03] (figure 1.13 (c)) ont utilisé des classeurs pour produire des agents capables d'´elaborer des stratégies dans le cadre de jeux de basket ou de football virtuels

Marco Ramos [Ram07] (figure (1.13 (a)) utilise un système de classeurs pour reproduire puis étudier des comportements réactifs anticipatifs dans une simulation de chien de berger.

(a) Marco Ramos (b) Olivier Heguy (c) Cédric Sanza

Figure1.13 : Quelques créatures évolutionnistes de l'IRIT.

3.4. Les agents hybrides

Les caractéristiques présentées pour les deux grands types d'agents, cognitifs et réactifs, proposent des solutions apparemment diamétralement opposées. Pourtant, elles peuvent être vues comme étant complémentaires. Afin de construire une architecture qui répond au mieux à un problème, on peut associer les deux types de modèles pour construire des architectures d'agents plus souples qu'on appelle *hybrides*.

Les agents hybrides sont des agents ayant des capacités cognitives et réactives mais restent capables d'être réactifs ou opportunistes lorsque la situation l'exige.

Touring Machine d'Innes Ferguson [Inn92] synthétise parfaitement l'architecture classique d'un agent hybride. Comme illustré dans la figure 1.14, il s'agit d'une architecture en 3 couches. La couche "haute" reprend celle de l'agent délibératif. Elle contient donc une représentation de l'environnement, et des capacités d'inférences sur ces connaissances. La couche "basse" est une architecture réactive typique, comme celle de Brooks. Enfin, une couche médiatrice, appelée séquenceur, arbitre entre les comportements délibératifs et les comportements réactifs.

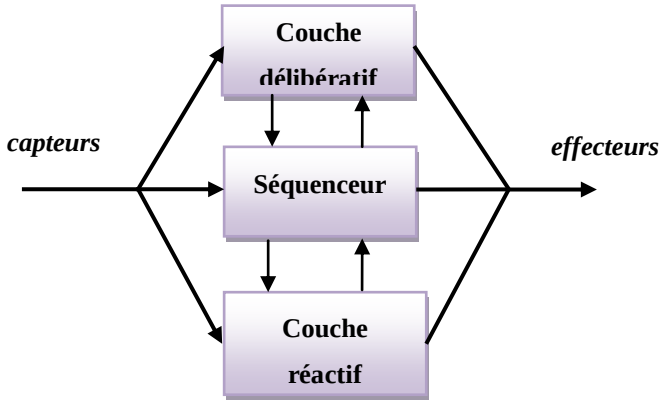


Figure 1.14: Le modèle hybride ou TLA (Three Layers Architecture) comporte 3 couches.

L'agent hybride est capable de planifier une action sur le long terme, en utilisant les connaissances qu'il possède de son environnement, mais il est parallèlement capable de réagir rapidement, lorsqu'il rencontre un danger par exemple. La puissance des architectures hybrides les destine le plus souvent au contrôle d'humanoïdes virtuels. Parmi les résultats les plus significatifs, nous pouvons citer ceux du VR-LAB de Lausanne [CT05] ou encore les projets SIAMES et BUNRAKU de l'IRISA de Rennes, basés sur l'architecture HPTS.

Dans le cadre du projet V-Man, Stéphane Sanchez [San04, SLD+04] a conçu l'architecture ViBes (*Virtual Behaviors*), qui propose une décomposition modulaire des compétences de l'agent. Il s'agit d'une architecture hybride traditionnelle dans laquelle différents gestionnaires sont responsables des capacités de l'agent : perception, navigation, décision, communication.

Un module de connaissances concentre les représentations de l'environnement, afin que l'agent puisse identifier et manipuler les objets qui l'entourent.

(a) Pour le projet V-Man, des humanoïdes virtuels doivent être capables de se comporter de manière réaliste.

(b) En outre, ils doivent apprendre à manipuler et interagir avec les objets qui les entourent.

Figure1.15 : Les humanoïdes du projet V-Man sont contrôlés par l'architecture ViBes[San04].

4. Notions d'interactions

Si nous revenons sur les concepts proposés par J. Ferber Nous pouvons constater qu'un système multi-agents (SMA) dépend directement de son environnement mais aussi des messages internes entre les agents (communication). Un système multi-agents est composé par les éléments suivants :

- Un environnement *M*, c'est à dire un espace disposant généralement d'une métrique.
- Un ensemble d'objets *O*. Ces objets sont situés, c'est à dire que, pour tout objet, il est possible à un moment donné, d'associer une position dans *M*. Ces objets sont passifs, c'est à dire qu'ils peuvent être perçus, créés, détruits et modifiés par les agents.
- Un ensemble d'agents *A*, qui sont des objets particuliers (*A* inclus dans *O*), lesquels représentent les entités actives du système.
- Un ensemble de relations *R* qui unissent des objets (et donc des agents) entre eux.

- Un ensemble d'opérations *Op* permettant aux agents de A de percevoir, produire, consommer, transformer et manipuler des objets de *O*.
- Des opérateurs chargés de représenter l'application de ces opérations et la réaction du monde à cette tentative de modification que l'on appellera les lois de l'univers.

La communication est un problème primordial dans les systèmes multi-agents, raison pour laquelle il est nécessaire de définir les lois d'interaction entre les agents et l'environnement.

Dans les paragraphes suivants nous reviendrons sur le concept d'environnement.

- Les interactions entre l'environnement et ses agents.
- Les interactions entre les agents.

4.1. Interactions environnement - agents

Ce type d'interaction se produit lorsque l'environnement contient des ressources utiles à l'agent. Le terme de ressource peut alors correspondre à une valeur énergétique, un outil, ou encore un espace mémoire, pour des agents de type processus. Les objectifs de ces agents étant liés aux ressources, ceux-ci doivent les prendre et / ou les utiliser. En conséquence, l'environnement doit être capable de réagir à une telle requête, en s'altérant, localement ou globalement. Une altération locale pour une prise de ressource énergétique peut correspondre à la disparition de cette ressource de l'environnement, dans un cas global, elle peut déclencher des réactions en avalanche, comme la suppression de cette ressource, puis la redistribution de nouvelles ressources et enfin une modification des lois environnementales.

Les ressources, sont utiles à l'accomplissement des objectifs des agents, mais certains objets de l'environnement peuvent être sans rapport entre eux. Les agents peuvent également être confrontés à ce genre d'objet, provoquant alors un autre type d'interaction, plus involontaire de la part des agents. Ce cas correspond par exemple, dans un environnement situé, à la rencontre entre un agent et un obstacle. Suivant la vitesse de l'agent et le poids des deux entités, l'agent peut être capable de pousser l'obstacle, ou non. L'environnement, par l'intermédiaire de ses lois, peut alors gérer ce genre de rencontre, et donc déclencher une suite de réactions.

4.2. Interactions agent - agent

D'autres types d'interactions se situent cependant entre les agents et permettent d'établir des dynamiques de groupes pour le travail coopérative la communication devient une priorité chez les agents. De cette façon ont défini une situation d'interaction par un ensemble de comportements résultants du regroupement d'agents qui doivent agir pour satisfaire leurs objectifs en tenant compte des contraintes provenant des ressources plus ou moins limitées dont ils disposent et de leurs compétences individuelles.

Il y a divers types d'interaction possibles entre agents où les agents coopèrent, il est possible de distinguer plusieurs degrés de coopération.

- *Le regroupement* où les agents forment des groupes pour atteindre leurs objectifs. Ce type de coopération permet d'améliorer de nombreux critères d'efficacité. Ainsi, dans un problème de survie, un groupe d'animaux est plus sûr qu'un individu seul, un prédateur ayant souvent plus de difficultés à pénétrer une masse compacte d'individus. De plus, la découverte de nourriture bénéficie à l'ensemble de la collectivité plutôt qu'à l'animal seul.

- *La communication* permet aux agents de se partager des informations. Les animaux possédant ce mode de coopération peuvent par exemple prévenir l'ensemble de leur groupe dès qu'ils détectent un prédateur.

- *La spécialisation* définit la capacité d'un agent à s'adapter à certains types de tâches très particulières, en développant soit un comportement très pointu, soit une propriété physique adéquate. Par exemple, dans une fourmilière, il existe des fourmis ouvrières, des fourmis soldats, ou encore la reine, chacune ayant une fonction spécifique dans la communauté pour assurer la pérennité du groupe.

- *La collaboration par partage de tâches et de ressources* consiste à remplir un objectif commun à plusieurs agents. Ainsi, des agents peuvent se répartir soit des tâches, soit des ressources, pour réaliser un travail spécifique.

- *La coordination d'actions* consiste à organiser la distribution du travail à un ensemble d'agents pour atteindre un objectif. Ce type de coopération se retrouve lorsque l'exécution de certaines tâches non productives à court terme est cependant utile pour accomplir un objectif

dans les meilleurs délais. Par exemple, la construction d'un bâtiment implique un certain nombre d'intervenants, il est nécessaire de les coordonner pour aboutir à un résultat positif en un temps raisonnable.

5. Adaptation de l'agent à son environnement

L'application de la théorie écologique aux agents virtuels est une idée exploitée depuis un certain nombre d'années par l'approche animat.

Le terme animat est une contraction d'animal artificiel. Il s'agit d'une créature virtuelle ou robotisée évoluant dans un environnement complexe et dynamique et mettant en œuvre des techniques inspirées des animaux [MG91]. Fondamentalement, l'animat possède des motivations et des besoins qu'il doit apprendre à satisfaire, grâce a des capacités d'apprentissage [Kha03] ou d'évolution [MSD+03, FS05], et ce malgré une intervention humaine minimale sinon existante. Cette approche s'inscrit totalement dans la logique des travaux de Brooks puisqu'elle replace l'interaction au cœur de l'intelligence. Contrairement à l'IA classique, l'approche animat ne vise pas à créer des créatures possédant une intelligence humaine mais plutôt des créatures parfaitement adaptées à leur environnement et possédant les moyens de maintenir cette adaptation.

L'approche animat poursuit des buts différents de l'animation d'acteurs virtuels. Elle s'inscrit dans une réelle volonté de comprendre et de modéliser les processus cognitifs des animaux alors que l'animation ne vise qu'au réalisme factuel des images. De ce fait, malgré un comportement cohérent et convaincant, les créatures de l'approche animat sont trop complexes pour être intégrées par l'animation comportementale. Cependant, il n'est pas vain de penser que ces idées peuvent être transposées, au prix d'une éventuelle simplification**.**

5.1. L'environnement

En revenant au dictionnaire, la définition informatique de l'environnement, c'est: l'ensemble de la structure dans laquelle un utilisateur, un ordinateur, ou un programme évolue.

Dans le cadre des SMA, l'environnement est l'ensemble des conditions extérieures susceptibles d'agir sur le fonctionnement d'un système. Un agent autonome évolue de façon continue dans un environnement, dans lequel peuvent exister d'autres agents.

Puisqu'un agent agit sur son environnement et que l'environnement agit sur l'agent, il existe une réelle interaction entre ces deux entités.

Du point de vue d'un agent, l'environnement correspond à «tout ce qui lui est extérieur»: les autres agents font aussi *a priori* partie de l'environnement [Ric01].

Passant à la définition globale de mot environnement, l'environnement est un espace, doté d'une topologie ou non, possédant des objets, passifs ou actifs basés sur un ensemble de conditions naturelles à savoir physiques, chimiques, biologiques et sociologiques et des lois d'interaction entre lui-même et ses objets.

5.1.1. Environnements dynamiques

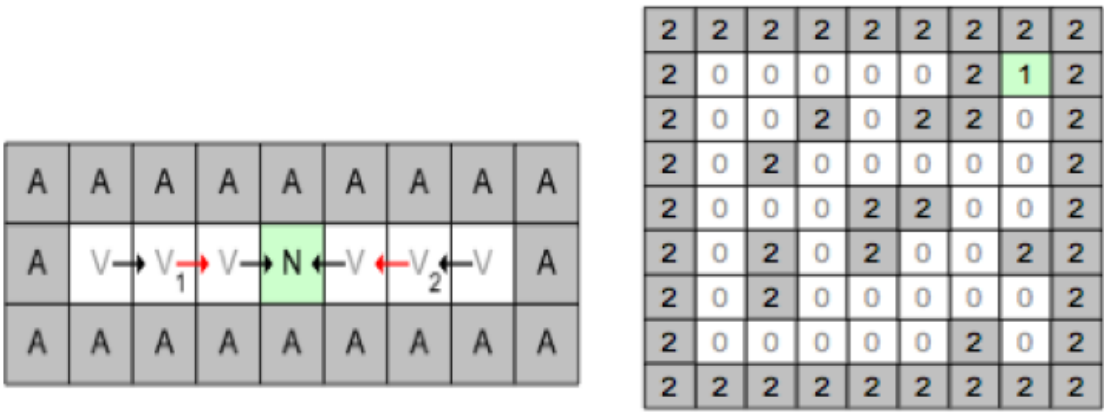
En effet, en dehors des obstacles et des objets, l'environnement comporte aussi toutes les autres entités de la simulation : des figurants au comportement automate, d'autres agents autonomes, ou encore les avatars des utilisateurs. Toutes ces entités ont en commun le fait qu'elles se déplacent et qu'elles interagissent également avec l'environnement.
Ensuite, il faut prendre en compte le fait que l'environnement est perçu de manière subjective par l'agent. Ainsi, les objets et les caractéristiques sont susceptibles de changer : par exemple, un objet peut se retrouver cache derrière un autre, la nuit peut tomber et rendre invisible les objets et les autres entités, etc.

5.1.2. Environnements Markoviens

Une autre propriété des environnements, dans le contexte de l'animation comportementale, est de savoir si la perception seule est nécessaire à la sélection de l'action optimale.
Dans un environnement dit Markovien, deux situations perceptuellement identiques impliquent la même action optimale. Dans un environnement non Markovien, il existe des situations ambigües. Dans ce cas, la sélection de l'action ne saurait reposer uniquement sur la perception de la situation.

La figure 1.16 illustre le problème rencontré par un agent plongé dans un environnement, à la recherche de nourriture (case *N* ou 1). La perception de l'agent se limite aux 8 cases qui entourent sa position. L'environnement *woods* 100 (figure 1.16(a)) contient deux positions ambiguës: *V*1 et *V*2. Ces 2 états sont perceptuellement identiques mais chacun d'eux implique une action différente. L'environnement *maze* 6 (figure

1.16(b)) est Markovien. Quel que soit la position initiale de l'agent dans le labyrinthe, sa perception est toujours suffisante pour prendre la bonne direction vers la nourriture.



(a) "*woods* 100" (b) "*maze* 6"

Figure 1.16 : l'environnement « woods 100 » est non Markovien car il présente 2 états ambigus l'environnement « maze 6 » est Markovien

5.2. Mécanismes d'adaptation

La production d'un comportement peut être appréhendée comme la recherche d'une solution optimale dans un espace de solutions potentielles. En simulation comportementale, la recherche du comportement est ainsi appelée phase d'exploration, par opposition a la phase d'exploitation, lors de laquelle l'agent utilise ses connaissances pour accomplir ses objectifs.

Comme pour l'homme, l'adaptation de l'agent a son environnement résulte d'un mécanisme d'évolution ou d'apprentissage.

5.2.1. L'évolution

D'un point de vue anthropologique, l'évolution est le premier mécanisme d'adaptation adoptée par les êtres vivants.

La vie artificielle emploie avec beaucoup de succès une métaphore computationnelle du processus d'évolution Darwinien. Dans ce contexte, ce processus s'applique hors-ligne a une population d'espèces, qui sont initialement créées, évaluées durant un cycle de vie, puis sélectionnées (ou non) selon leur aptitude à survivre dans leur environnement ou à

accomplir des objectifs prédéterminés. L'évolution est arrêtée lorsqu'une solution acceptable est trouvée parmi les individus de la population. Cette solution est ensuite utilisée pour la phase d'exploitation. L'évolution génétique est généralement restreinte à la production de créatures simples aux comportements rigides. Il est toutefois possible d'employer un mécanisme d'´evolution comme méthode d'optimisation pour mettre au point les contrôleurs de créatures plus complexes. Cela étant, lorsque la créature possède un contrôleur flexible et susceptible d'être adapte durant le cycle de vie de celle-ci, on utilise plus avantageusement une procédure d'apprentissage.

5.2.2. L'apprentissage

L'apprentissage est la deuxième forme d'adaptation s'applique aux agents dont le contrôleur est capable d'être adapté facilement, sans avoir à le reconstruire entièrement, et indépendamment de l'agent lui-même. Contrairement à l'évolution où l'agent est jugé sur sa performance globale dans la simulation et en un temps donné, l'apprentissage nécessite que l'agent soit capable d'évaluer son comportement à tout instant. Pour cela, il doit être capable d'une part d'extraire des informations de l'environnement et d'autre part de modifier son contrôleur en conséquence. De ce fait, l'apprentissage est généralement mené enligne.

La plupart des architectures comportementales ne présentent des agents adaptatifs qu'au sens le plus strict du terme: ils sont capables de gérer des situations imprévues grâce à leurs facultés réactives, mais ils n'apprennent pas à partir du retour que peut leur fournir l'environnement.

Il existe plusieurs avantages d'introduire un système d'apprentissage dans les architectures comportementales:

- d'obtenir des agents robustes et autonomes sur de longues périodes de simulation;
- de faciliter l'implémentation de l'agent et de ses comportements: la création "à la main" d'un agent complexe est une tâche difficile car envisager, dès sa conception, toutes les situations possibles d'un environnement dynamique est quasiment impossible.

Parmi les différents systèmes d'apprentissage fréquemment utilisés en simulation comportementale, nous considérons que les deux types de systèmes majoritairement

rencontrés sont ceux basés sur les méthodes d'*apprentissage par renforcement* et les *systèmes évolutionnistes* (*algorithmes génétiques* et *système de classeurs*).
Suivant les informations que l'environnement met a disposition du système d'apprentissage, on distingue trois formes d'apprentissages :

- ***Apprentissage supervisé***

  Voit l'environnement fournir au système un ensemble de couples <*P*,*A*> qui définissent, pour chaque vecteur de perception *P*, la réaction attendue ou idéale *A*. L'entité bâtit donc son comportement par imitation d'un comportement idéal fourni par l'environnement, assurant un rôle de superviseur. On emploie généralement le terme de *feedback* instructif.

- ***Apprentissage non supervisé***

  Ne repose pas sur la notion de *feedback*. Le système n'extrait de l'environnement que des vecteurs de perception *P*.

- ***Apprentissage par renforcement***

  En effet, au lieu de fournir directement le comportement correct *A* pour chaque perception *P*, l'environnement fournit au système une valeur *R (récompense)* rendant compte de l'adaptation de l'agent dans l'environnement. A la charge du système de s'adapter afin de maximiser cette valeur. Par analogie avec le *feedback* instructif, on parle ici de *feedback* évaluatif. Une caractéristique fondamentale de cet apprentissage est la nécessité d'une exploration active de l'environnement [RA98]. En effet, le signal de renforcement indique à quel point, pour chaque état, une action est cohérente, mais en aucun cas si cette action est la plus pertinente de l'ensemble des actions possibles.

6. Conclusion

Tout au long de ce chapitre, nous avons présenté les agents classiques : réactifs, délibératifs hybrides et évolutionnaires. Les principaux inconvénients de ces agents sont d'une part la nécessité de connaître à l'avance toutes les situations possibles afin de générer des comportements cohérents, et d'autre part la difficulté pour le concepteur de définir les paramètres qui permettent de simuler ces comportements.

Nous nous inspirerons donc de coupler différentes techniques pour la conception de notre architecture qui sert à augmenter les capacités de raisonnement des agents autonomes.

Chapitre II : Les Créatures Artificielles

Chapitre2

Systèmes évolutionnaires

"The important thing in science is not so much to obtain new facts as to discover new ways of thinking about them."

WILLIAM LAWRENCE BRAGG

1. Introduction

Les algorithmes évolutionnaires sont inspirés de la théorie Darwinienne d'évolution des espèces. Les individus d'une population, les plus adaptatifs aux environnements, ont plus de chances de survivre et de se reproduire et ceux qui sont moins adaptatifs risquent d'être éliminés. Les algorithmes évolutionnaires se divisent en plusieurs branches : l'algorithme génétique, la programmation évolutionnaire, la stratégie d'évolution, la programmation génétique et les systèmes de classeurs.

Dans ce chapitre, nous allons décrire le fonctionnement des algorithmes génétiques notés (AGs) et de leur extension aux systèmes de classeurs (appelés LCS Learning Classifier System). Nous proposons ensuite une vue d'ensemble au travers d'une présentation progressive identifiant les différents problèmes rencontrés par les différentes versions de système de classeurs. Le chapitre est articulé come suit : la section 3 présente les principes utilisés dans les systèmes de classeurs (LCS) en proposant une formalisation de leur structure et de leur dynamique interne, la section 2.2 montre les différentes versions de systèmes de classeurs : nous allons présenter les systèmes de classeurs ZCS, XCS et YCS. Enfin, la section 3.2.5 dresse une étude comparative des systèmes de classeurs permettant d'aider au choix d'un modèle en fonction de l'analyse du problème à traiter.

2.1. Algorithme génétique

Les algorithmes génétiques sont des méthodes adaptatives qui peuvent être utilisés pour résoudre des problèmes d'optimisation ou de recherche de solutions. Leurs principes de base dans le cadre de l'optimisation mathématique ont été rigoureusement définit par John Holland dès 1975 [Hol75]. Cependant, la puissance de calcul des ordinateurs de l'époque n'était alors pas suffisante pour l'utilisation des algorithmes génétiques sur des problèmes réels de grande taille. Leur développement s'est accéléré à partir de 1989, notamment grâce aux travaux de synthèse présentés par David Goldberg [Gol89].

2.2. Structure d'un algorithme génétique

La structure générale d'un algorithme génétique est relativement simple. En effet, cette technique d'optimisation consiste à explorer un espace des solutions à partir d'une population initiale d'éléments, jusqu'à l'apparition de la solution au problème posé. Chaque élément est codé par une chaîne de bits de longueur fixe, ce qui permet de représenter très simplement des valeurs entières ou réelles et donc d'optimiser les fonctions mathématiques. Il existe d'autres méthodes de représentation comme les arbres pseudo-LISP [Koz92], mais la méthode des chaînes de bits demeure plus directe à mettre en œuvre.

Le cycle d'évolution par algorithme génétique est présenté sur la figure 2.1 .L'application commence par la création aléatoire d'une population 'individus représentant les premières solutions envisagées au problème posé (génération0) Chacun de ces individus est ensuite évalué afin de déterminer son indice de performance associé, appelé traditionnellement *fitness*[2]. Puis, une population intermédiaire, similaire à un bassin de reproduction, est créée à l'aide d'un opérateur de *sélection* probabiliste favorisant les individus les plus performants. Les membres de cette population intermédiaire sont appareillés afin de se reproduire en appliquant des opérateurs génétiques (le *croisement* qui permet l'échange de matériel génétique et la *mutation* qui transforme aléatoirement une partie du génome) sur la structure des individus parents.
Les individus enfants issus de la phase de reproduction, ainsi que ceux qui n'ont pas

2fitness: indice d'adaptation d'un individu à son environnement, mesure de performance d'un individu dans un problème d'optimisation.

réussi à se reproduire malgré leur présence dans le bassin de reproduction (on parle alors de "survivants"), constituent la nouvelle génération. Les membres de cette dernière sont évalués et le processus est renouvelé jusqu'à l'obtention d'un critère d'arrêt généralement en relation avec la performance obtenue pour le problème posé.

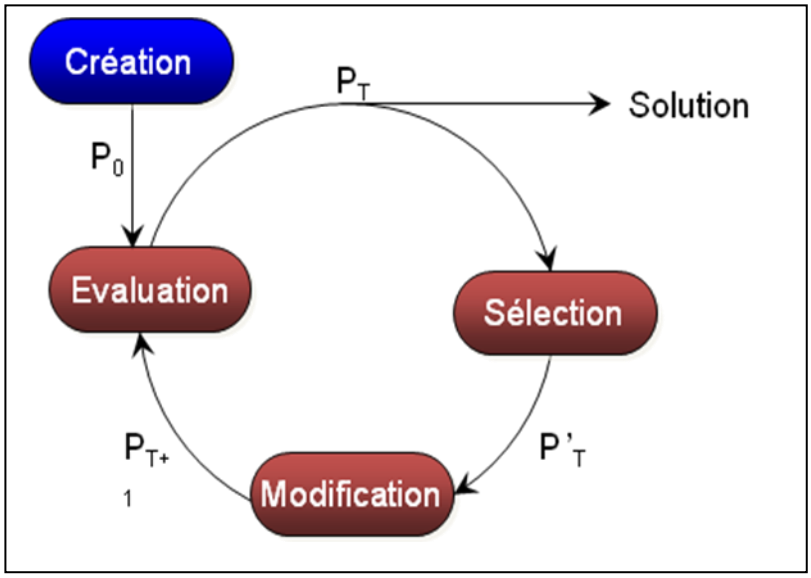


Figure 2.1 : Cycle d'évolution par algorithme génétique

2.3. Les opérations sur l'algorithme génétique.

2.3.1.Génome des individus

Traditionnellement, le génome des individus d'un algorithme génétique est constitué par un chromosome unique de longueur fixe et composé de gènes issus de l'alphabet binaire {0,1}. On fait souvent référence au chromosome d'un individu par l'appellation "chaîne de bits". Cette représentation a été choisie par Holland car elle permet un support simple des opérations évolutionnistes et elle est, de plus, applicable à un grand nombre de problèmes.

Figure 2.2 : Un chromosome selon Holland.

2.3.2. Création de la population initiale

Au début de l'évolution, l'AG crée une population initiale. Elle contient des individus générés aléatoirement. Elle n'influence pas sur la performance globale de l'AG car l'AG garde les meilleurs individus et élimine les moins efficaces au fil de l'évolution. Les individus dans la population sont ensuite évalués par une fonction objective. La première génération est créée. Normalement, la solution obtenue par la première génération ne satisfait pas les critères d'optimisation, alors une nouvelle génération commence. La solution optimale est obtenue après plusieurs générations.

2.3.3. Evaluation

L'évaluation de la population consiste à attribuer une note à chaque individu en fonction de son aptitude à résoudre le problème posé. La note est généralement appelée *fitness* ou la fonction objectif *fonction de fitness*. Cette fonction est dépendante du problème et sera définie par l'utilisateur.

A chaque début de génération, tous les individus de la population sont notés afin de permettre aux techniques de *sélection* de les comparer.

2.3.4. Sélection

Cet opérateur consiste à sélectionner un certain nombre d'individus de la population actuelle $P(t)$ de manière à créer une nouvelle génération $P(t+1)$.

Le principe est de copier les individus de la population dans une population intermédiaire, les individus les plus adaptés étant recopiés plusieurs fois alors que les plus faibles auront tendance à ne pas être copiés en se basant sur leur fitness. Une fois la population intermédiaire constituée, des paires d'individus sont choisies aléatoirement afin de se reproduire (phase de *reproduction*).

Les parents sont croisés pour créer les enfants, ensuite ces derniers sont mutés avec une certaine probabilité. Enfin, ils sont rajoutés dans la population.

Il existe de multiples operateurs de sélection, nous allons voir les plus connues, la roulette pipée, le reste stochastique, le tournoi et la sélection par rang.

2.3.4.1. La roulette pipée

Dans le cas de la sélection par roulette pipée, la population intermédiaire est assimilée à une roue divisée en secteurs. Chaque secteur est occupé par les copies d'un individu unique. La roulette pipée effectue un tirage aléatoire sur un des secteurs dont la taille des secteurs est proportionnelle à la valeur P_{sel} calculée par la formule:

$$P_{sel} = \frac{f(i)}{\sum_{i=1}^{n} f(i)} \qquad (2.1)$$

Avec $f(i)$ la fitness de l'individu i,

n le nombre d'individus de la population.

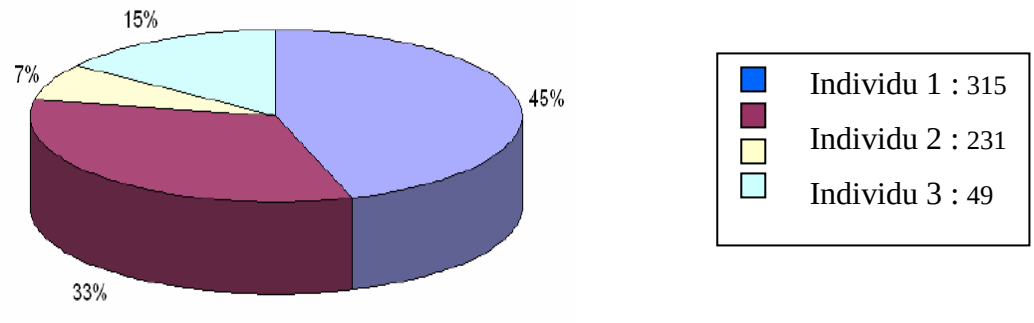


Figure 2.3: Exemple de roulette pipée pour une population de 4 individus en fonction de P_{sel}.

En théorie, le meilleur individu a plus de chance d'être choisi. Mais rien n'empêche qu'un mauvais individu soit choisi car le tirage reste toujours aléatoire

2.3.4.2. Le reste stochastique

Cette méthode est utilisée a la place de roulette pipée notamment sur des population de petite taille, Le nombre de copies d'un individu x est égal à la partie entière du rapport entre sa fitness f et la fitness moyenne $\bar{f}$ de la population.

2.3.4.3. La sélection par tournoi

La sélection par tournoi minimise le risque de ne pas choisir les meilleurs individus. Cette technique s'effectue en deux étapes : dans un premier temps, on effectue un tirage aléatoire *Ntournoi* individus participants parmi la population sans se préoccuper de leur fitness au tournoi Dans la deuxième étape, on détermine le vainqueur en choisissant celui qui a la fitness la plus forte parmi *Ntournoi* individus. *Ntournoi* prend une valeur entre 2 et le nombre total d'individus de la population.

Cette méthode de sélection est plus élitiste que les précédentes car l'individu ayant la force la plus faible ne pourra être sélectionné que s'il être tiré *Ntournoi* fois de suite, alors que celui qui a la meilleure force n'a besoin d'être tiré au sort qu'une fois sur *Ntournoi* pour être dans la nouvelle population.

2.3.4.4. La sélection par rang

La méthode de sélection par rang permet de limiter la prédominance d'un individu extrême qui peut apparaître lors de l'utilisation de la roulette pipée. Les individus sont classés en fonction de leurs notes .Les individus sont classes par ordre décroissant en fonction de leur note. Ainsi le meilleur individu aura une note égale à la taille de la population et le dernier aura une note égale à 1. La sélection par rang est souvent couplée avec une fonction de mise a l'échelle de réduire ou d'augmenter l'influence ces meilleurs rangs.

2.3.5. Reproduction

La modification d'un individu à l'aide de l'opérateur de *reproduction* peut s'effectue par deux manières différentes par le *croisement* ou la *mutation*. Le rôle de ces opérateurs est d'explorer l'espace autour des valeurs présentes dans la population (recherche local)

Ainsi qu'une exploration globale de l'espace de recherche (zones inconnues, non encore visitées par l'algorithme).

A partir de la population des individus précédemment sélectionnés, les individus sont regroupés par paire. Chaque paire subite l'operateur de croisement avec une probabilité t_{crois} appelée *taux de croisement*. Ensuite chaque individu subit l'operateur de mutation avec une probabilité t_{mut} appelée *taux de mutation.*

La population après application de la mutation constitue la génération suivante.

2.3.5.1. Le croisement

Le croisement consiste à mélanger des gènes de deux chromosomes parents pour but de produire des individus potentiellement mieux adaptés au problème. Il représente un partage de la connaissance entre les deux parents et les enfants héritent des propriétés de leurs parents. Le taux de croisement est assez fort afin d'augmenter la chance de la génération des enfants qui seraient plus performants que les parents.

Les opérateurs de croisement les plus courants sont le croisement à un point de coupe, le croisement à deux points de coupe et le croisement uniforme. Il est important de noter que, hormis la comparaison entre le croisement à 1-point et celui à 2-points, il n'existe pas a priori de croisement qui soit meilleur qu'un autre. En réalité, le choix de l'utilisation d'un type de croisement dépend de la nature du problème, de la structure de chromosome et de l'évolution souhaitée de l'algorithme génétique. De plus, il est fréquent d'utiliser des opérateurs de croisement alternatifs et des opérations additionnelles (inversion de gènes, réorganisation, contrôle de validité des enfants générés) afin d'adapter le croisement au problème pour obtenir une meilleure convergence de l'algorithme.

- **Croisement à un point de coupe**

Le principe est de couper en un point choisi aléatoirement les chromosomes parents en deux parties, Un chromosome enfant est formé par une copie de la partie du chromosome parent 1 avant la coupe et de l'autre partie du chromosome parent 2 après la coupe. Un deuxième chromosome enfant est créé de la même manière mais il faut permuter les deux parents.

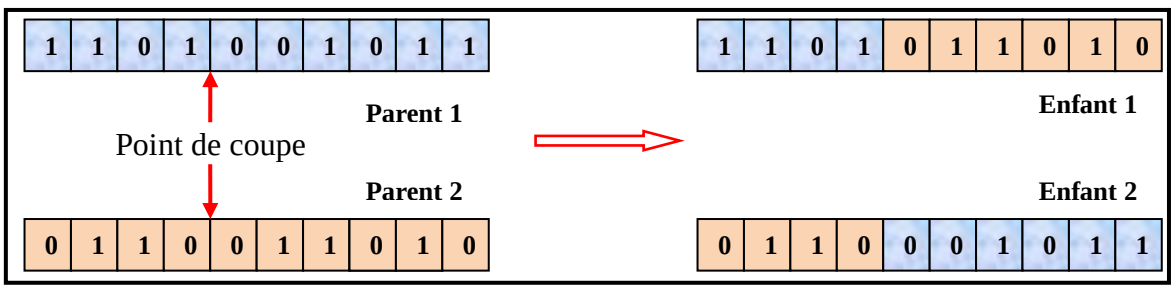


Figure 2.4: Croisement avec un point de coupe

- **Croisement à deux points de coupe**

Afin d'améliorer les performances du croisement, différents algorithmes utilisant plus d'un point de coupe ont été conçus. DeJong [Dej75] a étudié l'efficacité des croisements avec points de coupe multiples et en a conclu que, si deux points de coupe améliorent la convergence, un nombre plus élevé réduit les performances de l'algorithme. Le problème étant que l'utilisation de nombreux points de coupe perturbe la formation de séries de bits pertinentes à l'intérieur du chromosome ("building blocks").

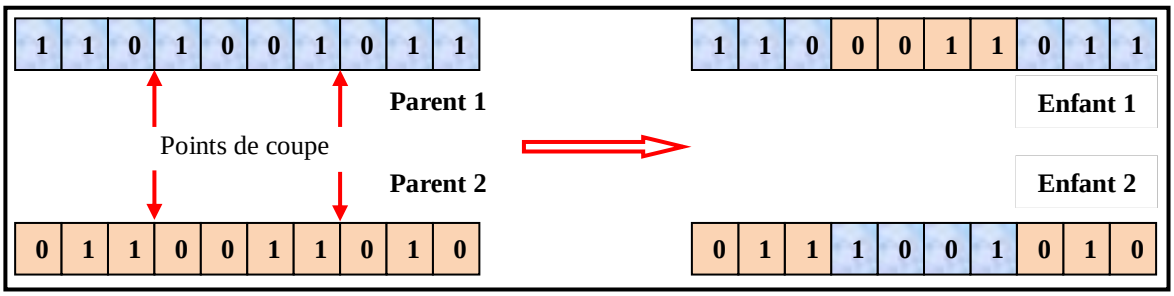


Figure 2.5: Croisement avec deux points de coupe

- **Croisement uniforme**

Le croisement uniforme est radicalement différent des croisements à un ou plusieurs points de coupe. Un masque de croisement est aléatoirement constitué. Le premier enfant est formé en copiant les gènes du premier parent dans le cas d'un bit de masque correspondant égal à 1 et ceux du second dans le cas contraire (bit de masque égal à 11zéro). Le second enfant est obtenu en inversant les parents. Habituellement, un masque de croisement est généré pour chaque paire de parents.

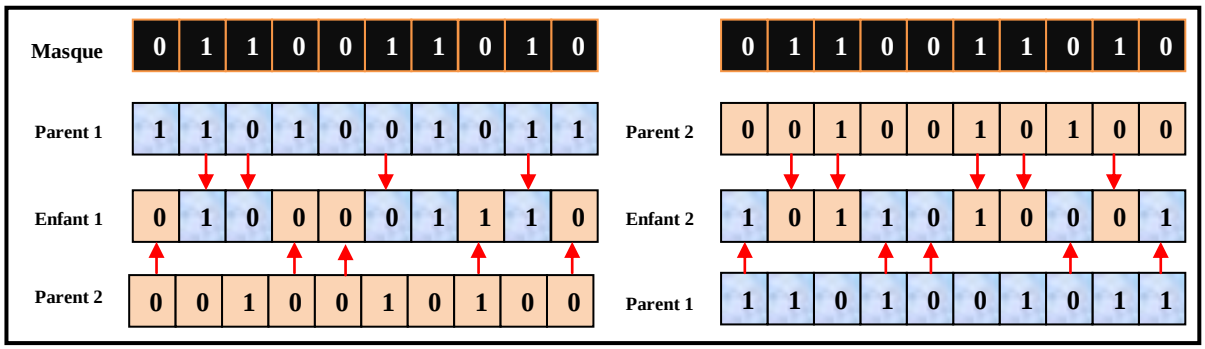


Figure 2.6: Croisement uniforme

2.3.5.2. La mutation

Le rôle de la mutation est de maintenir la diversité génétique au sein de la population d'individus. Cette fonction effectue une perturbation au niveau du code de l'individu (figure 2.7). Elle permet d'augmenter la diversité de la population et ainsi limiter la dérive génétique due à un processus de sélection trop élitiste. Cette dérive concerne le comportement du mécanisme de reproduction qui va éliminer des solutions peu prometteuses a priori mais qui peuvent être primordiales pour arriver à la solution cherchée. Le taux de mutation est faible et reste constante pendant l'évolution de l'AG.

En pratique il existe de nombreuses manières de muter un chromosome : modification d'un ou plusieurs gènes (s), changement d'un gène, suppression ou ajout d'un gène.

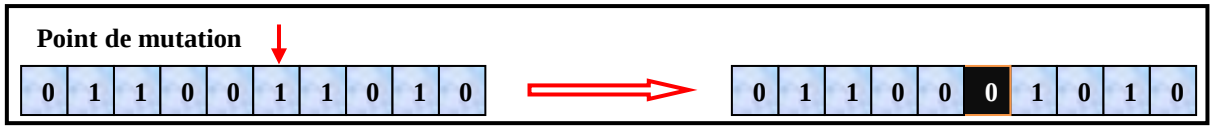


Figure 2.7: Mutation d'un individu

2.3.6. Le critère d'arrêt

Le critère d'arrêt intuitif d'un algorithme génétique est lorsque la solution définie par le génome d'un individu est la solution optimale connue du problème posé. Cependant, une telle solution n'est pas connue dans la plupart des problèmes pour lesquels est utilisée cette méthode d'optimisation. Afin de pallier à la difficulté de savoir si l'algorithme a effectivement trouvé la solution optimale, plusieurs techniques sont couramment utilisées :

- arrêter l'algorithme au bout d'un certain nombre de générations.
- arrêter l'algorithme lorsque le meilleur individu n'a pas été amélioré depuis un certain nombre de générations.
- arrêter l'algorithme lorsque la diversité génétique au sein de la population n'est plus suffisante pour explorer plus profondément l'espace de recherche des solutions.

3. Les systèmes de classeurs

3.1. Learning Classifier Systems (LCS)

Le système de classeurs, noté LCS, est un outil intéressant pour l'apprentissage machine (*machine learning*). Il s'agit d'une approche inspirée de la vie artificielle.

Le LCS furent introduits par John Holland en 1978 [Hol78] comme une architecture utilisant les algorithmes génétiques afin d'étudier l'apprentissage des systèmes à bases de règles.

La base des travaux de Holland est fondée sur deux idées essentielles. La première est que la théorie Darwinienne de survie du meilleur peut être utilisée pour conditionner l'adaptation d'un système artificiel à un environnement inconnu. La seconde est qu'un agent peut apprendre à effectuer une tâche juste en interagissant avec un environnement partiellement connu et en essayant de maximiser les récompenses qu'il reçoit de cet environnement en retour de ses actions.

Le principe d'un système de classeurs est donc de faire évoluer un ensemble de règles de production, les classeurs, afin d'apprendre à un agent la réalisation d'une tâche précise. La connaissance de l'agent étant représentée par la population des classeurs. L'agent perçoit la totalité ou une portion de l'état de l'environnement à l'aide de capteurs. Les messages issus de cette perception sont transmis à la base de classeurs afin de déterminer celui ou ceux qui doivent être activés (en fonction de leur performance) afin de produire une action. L'action est exécutée à l'aide d'effecteurs et un retour de l'environnement permet une mise à jour des indications de performance des classeurs (ce qui est similaire au mécanisme général d'apprentissage supervisé ou par renforcement). La découverte de nouvelles règles (nécessaire pour obtenir un système adaptatif) est assurée, les systèmes de classeurs ayant été conçus par Holland dans ce but, par l'utilisation d'un algorithme génétique.

Bien que les systèmes de classeurs originaux aient subi de nombreuses modifications et évolutions au cours des dernières années, la grande majorité de leurs variantes utilisent le schéma général de fonctionnement définit par Holland ainsi que le formalisme de codage des informations initial. De plus, tout système de classeurs comporte les éléments suivants: une base de règles (les classeurs), un mécanisme d'arbitrage des classeurs concurrents, un mécanisme de renforcement et un mécanisme de découverte des nouvelles règles.

3.1.1. Codage des informations et base de règles

Un classeur est une représentation générique formalisée d'une règle de type "**SI** *condition* **ALORS** *action*". Un classeur est donc composé de trois parties distinctes : une partie condition (qui correspond aux perceptions venant de l'environnement), d'une partie action (qui permet à l'agent d'agir sur son environnement) et d'une force (qui aide à faire un choix entre plusieurs règles). L'ensemble des classeurs est stocké dans la base de règles.

La figure 2.8 montre un exemple de classeur selon Holland. L'avantage du codage de classeur par les chaînes des bits permet de faciliter l'application de l'AG.

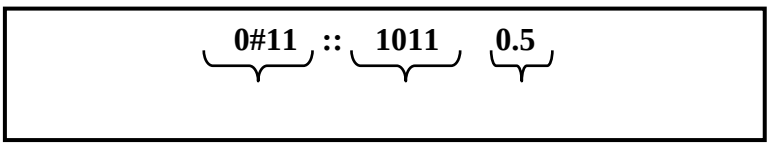


Figure 2.8: un classeur selon Holland

Condition de classeur

La condition de classeur de longueur fixe est initialement générée avec des valeurs aléatoires du triplet {0, 1, #} où # représente un joker (appelé *don't care*) pouvant à la fois représenter '0' ou '1'. L'utilisation du joker permet au système de généraliser la condition du classeur et d'obtenir des solutions plus compressées (moins des classeurs). Par exemple, un classeur ayant une condition de la forme 010# correspond aux messages entrants 0100 et 0101 mais elle ne correspond pas au message entrant 0110. La condition du classeur correspond 010# peut répondre à la fois à deux messages au lieu de l'utilisation de deux classeurs.

Plus un classeur a des symboles #, plus il est généralisé. S'il contient peu de symboles #, il est spécifique. Tim Kovacs [Kov97] a classé trois types de classeur :

- Un classeur surgénéralisé (over-general) contient trop de symboles# pour résoudre efficacement le problème posé parce qu'il peut répondre à plusieurs entrées mais il ne reçoit pas de récompenses constantes. Un classeur surgénéralisé doit être spécialisé en remplaçant le symbole # par une valeur spécifique 0 ou 1.

- Un classeur *généralisé au maximum* (*maximally general*) est un classeur si on ajoute un symbole # supplémentaire, il devient *surgénéralisé*. Le système doit converger vers une solution optimale constituant des classeurs *généralisés au maximum*.
- Un classeur *généralisé sous-optimal* (*suboptimally general*) pourrait être plus généralisé tout en conservant la même performance en remplaçant la valeur 0 ou 1 par un symbole #.Un classeur *généralisé sous-optimal* doit être généralisé.

Action de classeur

L'action du classeur doit avoir la même longueur que la condition du classeur et elle est codée sous la forme d'une chaîne de bits (0 ou 1) de longueur finie. Le joker # n'est pas admis dans l'action de classeur.

Force

La force associée à un classeur représente la performance passée du classeur au sein de l'environnement dans lequel le système apprend. Dans les LCS de Holland, la force joue à la fois deux rôles : la prédiction et la fitness. La force est considérée comme la prédiction à la sélection de l'action. La prédiction estime une prédiction de la récompense à venir au retour de l'exécution d'une action dans l'environnement. La force élevée prédit de recevoir une récompense élevée. La force est considérée comme la fitness lors de l'application de l'algorithme génétique à la population de classeurs. Ainsi, les classeurs les plus performants ont plus de chance de se reproduire afin d'améliorer la performance globale du système en générant des règles potentiellement mieux adaptées.

3.1.2. Fonctionnement des LCS

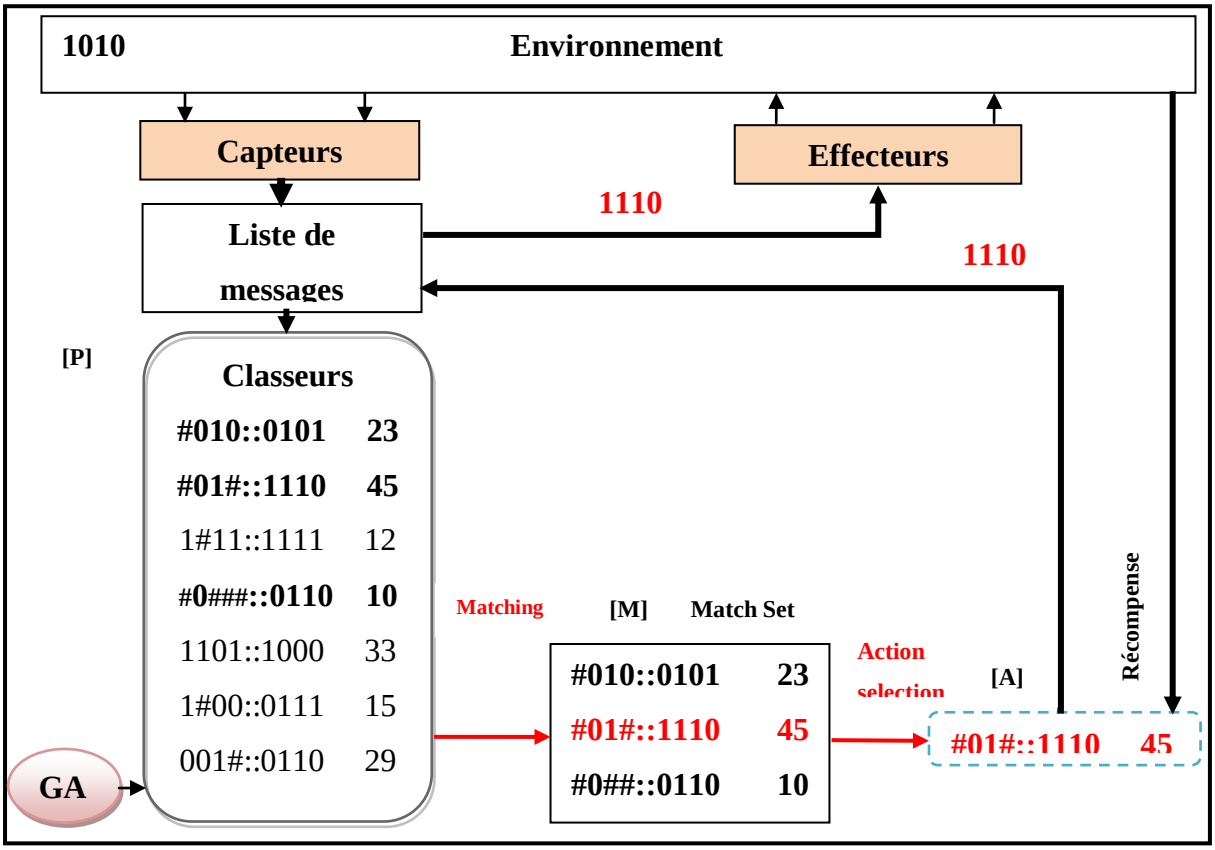


Figure 2.9: Schéma fonctionnel des Learning Classifier System

Les systèmes de classeurs fonctionnent de manière itérative jusqu'à ce que le système ait apprit à résoudre parfaitement la tâche désirée ou, si un apprentissage parfait n'est pas réalisable, jusqu'à l'exécution d'un nombre fixé d'itérations. La figure 2.9 décrit le schéma d'un système de classeurs LCS. Nous découpons son fonctionnement en six étapes. Une génération commence par l'étape 1, se termine après l'étape 6, et ensuite une nouvelle génération commence.

Etape 1 : Les capteurs perçoivent l'environnement et envoient au système un message x sous la forme d'une chaîne de bits de la taille L**.** Ce message est fourni à la *liste de messages*.

Etape 2 : La *base de règles* [P] est alors scannée pour déterminer quels sont les classeurs dont la partie *condition* correspond au premier message de la liste, l'objectif est de déterminer quel classeur peut répondre au message entrant et de le classer dans un ensemble [M] (*match set*).

Etape 3 : Normalement, l'ensemble [M] ne contient que les classeurs correspondant au message entrant ***x***. Le problème est alors de choisir un classeur pour activer son action ?. C'est le dilemme entre l'exploitation et l'exploration. L'*exploitation* signifie que le système utilise la connaissance obtenue dans le passé pour utiliser une action qui renvoie la récompense maximum, autrement dit, le système choisit l'action la plus efficace. Il s'agit donc du classeur ayant la force la plus élevée. Mais pour découvrir une telle action, le système doit essayer des actions différentes non encore testées. Il *explore* alors les actions pour déterminer les meilleures actions qu'il choisira dans le futur. Plus spécifiquement, il peut choisir un classeur parmi des classeurs ayant une force faible. La solution proposée par Holland est une méthode utilisée dans les enchères, permettant de favoriser les classeurs les plus performants mais aussi de laisser une chance aux classeurs les plus faibles. A l'itération *t*, chaque classeur *i* en concurrence pose une enchère $E_i(t)$, seule l'enchère du classeur victorieux (celui qui a posé l'enchère la plus élevée) sera payée et déduite de sa *force*. L'enchère $E_i(t)$ posée par un classeur dépend de sa *force* $F_i(t)$ qui représente l'utilité du classeur pour la résolution du problème. Son calcul s'effectue selon la formule:

$$E_i(t) = k_0 \times F_i(t) \tag{2.2}$$

avec k_0 une constante dont le rôle est celui d'un facteur de risque représentant la perte potentielle que peut subir la force du classeur sur une itération ($0 < k_0 \leq 1$),
$F_i(t)$ la force du classeur i à l'itération t.

Ainsi, plus la force d'un classeur est élevée (plus il est utile), plus son enchère est forte et plus il a de chance de remporter la compétition et d'activer son *action*.

Le calcul proposé de l'enchère présente le problème d'être purement déterministe et de favoriser l'exploitation des classeurs les plus performants par le passé au détriment de l'exploration de nouvelles solutions nécessaire pour une résolution complète du problème posé. Afin de promouvoir l'essai de nouveaux classeurs, l'enchère $E_i(t)$ n'est pas directement utilisée pour choisir le classeur victorieux. Une première méthode consiste à effectuer la sélection à partir d'une enchère effective $eE_i(t)$ constituée par la somme de l'enchère $E_i(t)$ et d'une fonction de bruit aléatoire $N(\sigma)$:

$$eE_i(t) = E_i(t) + N(\sigma) \tag{2.3}$$

Ensuite, la force du classeur gagnant est déduite d'une quantité correspondant à son enchère afin de diminuer sa chance de toujours gagner l'enchère. La partie action du classeur gagnant est envoyée à la liste de messages. Si l'action n'est pas éligible, elle est considérée comme un nouveau message et le *cycle d'inférences internes* (recherche en profondeur) activera d'autres classeurs. Dans le cas contraire, l'action du classeur gagnant est éligible, le système passe à`l'étape 4.

Etape 4 : L'action éligible sera transmise aux *effecteurs* pour modifier l'environnement.

Etape 5 : Un algorithme permettant l'apprentissage par renforcement d'un système de classeurs doit être capable de modifier les forces des classeurs afin d'optimiser les performances du système. Cette information est une valeur scalaire, appelée récompense, représentant la qualité de l'action exécutée. Cette récompense indiquera si le système se comporte correctement (récompense positive) ou non (récompense négative ou punition). Une récompense peut être de valeur fixe (par exemple -1, 0, +1) ou variable.
En fait, le système reçoit une récompense, met à jour la force des classeurs gagnants activés par le cycle d'inférences internes.

L'algorithme BB (*Bucket Brigade*) proposé par Holland [HHN+86] effectue une distribution des enchères entre les classeurs de cette chaîne. Son fonctionnement est le suivant:

1. La récompense (ou la punition) renvoyée par l'environnent est ajoutée à la force du classeur ayant produit l'action exécutée.

2. Chaque classeur paie l'enchère qu'il a posée afin d'être sélectionné au classeur ayant permis son activation (i.e. celui qui a posté le message correspondant à sa condition).

Ainsi, la force $F_i(t)$ d'un classeur *i* est mise à jour selon l'instanciation suivante:

$$F_i(t) \leftarrow F_i(t) - E_i(t) + R_i(t) \tag{2.4}$$

avec $F_i(t)$ la force à l'itération courante *t*,
$E_i(t)$ l'enchère du classeur *i* à l'itération courante *t*,
$R_i(t)$ la récompense issue de l'environnement dans le cas ou le classeur *i* est celui qui a déclenché l'action, l'enchère du classeur que le classeur *i* a activé sinon.

La propagation inverse des récompenses permet une distribution équitable des récompenses à travers tous les systèmes de classeurs permettant ainsi aux règles de coopérer afin de générer un comportement optimal face au problème posé. De plus, l'action cumulée du système d'arbitrage et de l'algorithme de distribution des crédits permet un renforcement important du système car la force des classeurs les plus performants ne cesse d'augmenter au fil des itérations, de ce fait ils proposent des enchères de plus en plus importantes, et ainsi accroissent les récompenses obtenues de toute la chaîne de classeurs menant à leur activation.

Les classeurs dans la population *P* qui participent très peu ou jamais aux enchères doivent payer une taxe $T_i(t)$ en fonction de leur force afin d'éliminer rapidement ces classeurs parasites.

Le point faible de l'algorithme BB est de maintenir une longue chaîne de classeurs et donc il prend le temps pour renforcer tous les classeurs de la chaîne.

$$T_i(t) = k_1 \times F_i(t) \tag{2.5}$$

avec $T_i(t)$ la taxe prélevée à l'itération *t* sur la force $F_i(t)$ du classeur *i*,
k_1 le coefficient de taxe (généralement $0 \le k_1 < 1$),
$F_i(t)$ la force du classeur *i* à l'itération *t*.
La force d'un classeur *i* évoluera alors selon l'instanciation:

$$F_i(t) \leftarrow F_i(t) - E_i(t) + R(t) - T_i(t) \tag{2.6}$$

Etape 6. Le système augmente sa performance en appliquant périodiquement un AG à la population [P] des classeurs pendant la période d'apprentissage.
Le rôle de l'AG dans des problèmes d'optimisation classiques est de chercher la règle la plus adaptative, mais dans le LCS est de trouver un ensemble coopératif des différents types de règle pour résoudre un problème spécifique.

3.1.3. Les environnements de travail des LCS

Les environnements à partir desquels les LCS sont capables d'apprendre peuvent être divisés en deux classes. La première comprend les environnements pour lesquels à chaque

message entrant est fournie une action qui génère une récompense marquant la fin du problème à résoudre. On qualifie ces environnements par le terme *singlestep* (mono-itération). La seconde est constituée par les environnements qui requièrent plusieurs itérations avant de produire l'action qui permet de recevoir une récompense. Ces environnements seront dits *multistep* (multi-itérations).

Généralement l'apprentissage en environnement *single-step* est plus facile que dans le cas *multi-step* car la récompense s'applique directement, et uniquement, au classeur qui vient juste d'être activé. Les environnements *multi-step* requièrent la formation d'une chaîne de classeurs qui s'activent successivement avant que la récompense ne soit reçue et distribuée à l'aide de l'algorithme d'allocation des crédits.

Afin d'évaluer les performances des systèmes de classeurs dans ces deux classes d'environnement, deux problèmes types sont utilisés: le multiplexeur booléen pour les environnements *single-step* et les environnements Woods pour les environnements *multi-step*.

3.1.3.1. Multiplexeur booléen

Les fonctions décrites par les multiplexeurs booléens sont largement utilisées dans les systèmes d'apprentissage comme les systèmes de classeurs, les algorithmes génétiques ou les réseaux de neurones. Leur popularité s'explique par la facilité de modularité de leur degré de difficulté.

L'exemple suivant montre la fonction booléenne à 6 entrées F_6 :

$F_6 = x_0'.x_1'.x_2 + x_0'.x_1.x_3 + x_0.x_1'.x_4 + x_0.x_1.x_5$

Les variables x_i sont les entrées booléennes du multiplexeur. Les symboles ".", "+" et "'" signifient respectivement le "ET", le "OU", et le "NON", logique. Le multiplexeur est donc considéré comme une boîte noire qui réalise la fonction F_6 (figure 2.10).

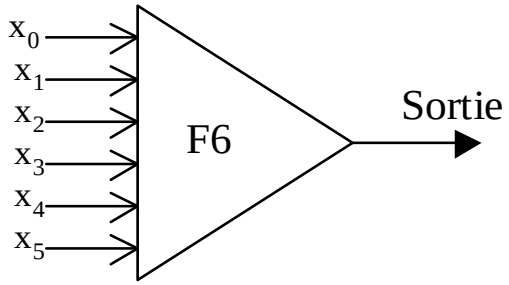


Figure 2.10 : Le Multiplexeur F_6

L'objectif du système est de calculer la sortie du multiplexeur (0 ou 1) en fonction des entrées. A chaque tour, le système reçoit un gain s'il trouve la bonne solution, autrement il ne reçoit rien. Ce type de problème es t souvent appelé problème a simple étape car le résultat a chaque cycle ne dépend que des entrées courantes. L'enchainement des entrées ne répond pas à une logique particulière. Ainsi, ce sont des environnements sans conséquence perceptive.

Wilson a été le premier à tester ce type de problème avec un LCS nommé Boole [Wil 87].

3.1.3.2. Environnement Woods

Les environnements woods ont été présentés par Wilson en 1985 [Wil85] dans le cadre de la simulation à l'aide d'un système de classeurs du comportement d'une créature virtuelle nommée Animat

Woods est un monde en 2 dimensions représenté par une grille rectiligne et composé de cases représentant des obstacles, arbres (A) et de la nourriture (N). Certaines cases sont vides (V) pour permettre à une créature (animât) d'explorer l'environnement à la recherche de son repas [Wil95]. Le but de l'Animat dans ces environnements est de se déplacer de case vide en case vide afin d'atteindre la nourriture en un minimum de déplacements.

Le système de perception de l'animât est limité aux cases qui l'entourent. Dans les problèmes abordés par les systèmes de classeurs [HM91], les états sont caractérisés par plusieurs attributs représentant autant de propriétés perceptibles de l'environnement. Ces états sont les situations que produisent l'environnement. Une situation est un vecteur de plusieurs valeurs discrètes, un pour chacun des attributs perçus.

L'environnent Woods se décline en plusieurs modèles de référence comme Woods1, Woods101 et bien d'autres. Le point commun entre tous ces mondes est leur constitution de base : ils sont définis par des motifs adjacents identiques se répétant à l'infini. Une simplification de ces environnements consiste à les limiter à un unique motif toroïdal (figure 2.11).

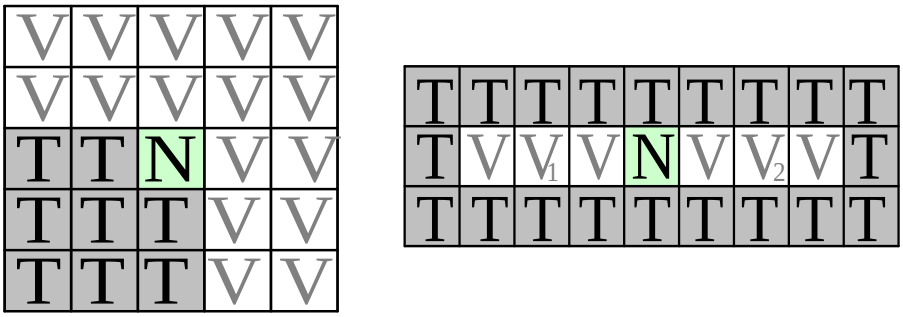


Figure 2.11 : Exemple d'environnements Woods1 et Woods100

Les mondes Woods sont classés en deux types : markovien et non-markovien. La transposition de la loi de Markov à Woods donne la propriété suivante : un monde Woods est markovien s'il n'existe pas deux conditions entrantes identiques (même schéma autour de l'Animat) nécessitant deux actions distinctes. Ainsi, Woods1 est markovien alors que Woods100 ne l'est pas car la propriété n'est pas vérifiée pour les cases V_1 et V_2. En effet, le déplacement optimum à partir de V_1 est dirigé vers la droite alors qu'il est orienté vers la gauche pour V_2.

S. Wilson a utilisé Woods7 pour tester les capacités d'apprentissage de son Animat. Woods7 est une grille toroïdale de 58 par 18. Elle contient 57 cases de nourriture reparties uniformément, chacune étant entourée aléatoirement de deux arbres. Le reste de la grille est vide.

Dans ses résultats, il montre que l'Animat apprend très rapidement jusqu'au $1000^{ème}$ essai. Ensuite, l'apprentissage est beaucoup plus lent. Au bout de 4000 essais, son système se stabilise et l'Animat arrive à retrouver sa nourriture en 5 déplacements, alors qu'un comportement aléatoire en nécessite 41 et que le minimum théorique est 2.2.

3.2. Versions évoluées des systèmes de classeurs

En marge des différentes variantes des LCS, des nouveaux standards comme les XCS se sont imposés. Ces systèmes méritent d'être décrits séparément car ils ont révolutionné les LCS en apportant de nouveaux concepts et en ouvrant de nouveaux horizons qui étendent considérablement les capacités des LCS. Les paragraphes suivants présentent donc les systèmes ZCS, XCS, ACS et YCS, ainsi que leurs différentes évolutions

3.2.1. Zeroth Level Classifier Systems (ZCS)

Bien que l'architecture des systèmes de classeurs consiste à généraliser la représentation de l'environnement, elle peut induire des problèmes de rapidité d'apprentissage. ZCS a été présenté par Wilson en 1994 [Wil94]. Il est considéré comme le LCS de niveau zéro, c'est à dire un LCS qui ne conserve que le strict minimum des LCSs classiques. Ainsi, il ne possède pas de liste de messages et utilise un format de classeur ayant un unique paramètre appelé force.

En effet, le ZCS suppose que les capteurs fournissent une information complète des situations de l'environnement, donc il ne nécessite plus d'utiliser le cycle d'inférences internes pour compléter une information incomplète ou imprécise. La suppression de ce cycle renforce la réactivité des systèmes de classeurs car il fait augmenter le temps de réponse d'un message entrant. La partie action n'est plus nécessairement de la même taille que la partie condition.

Le système permet l'utilisation d'autres représentations pour la partie action comme un type entier dont une valeur indique une action. Par contre, le ZCS n'est pas adapté aux environnements non-Markovien où les capteurs ne représentent qu'une information partielle de l'état de l'environnement. Le deuxième changement dans le ZCS est l'ajout d'un ensemble *A* (*action set*) pour regrouper des classeurs ayant le même comportement (c'est-à-dire déclenchant la même action) et donc partager également la récompense provenant de l'environnement.

3.2.1.1. Fonctionnement des ZCS

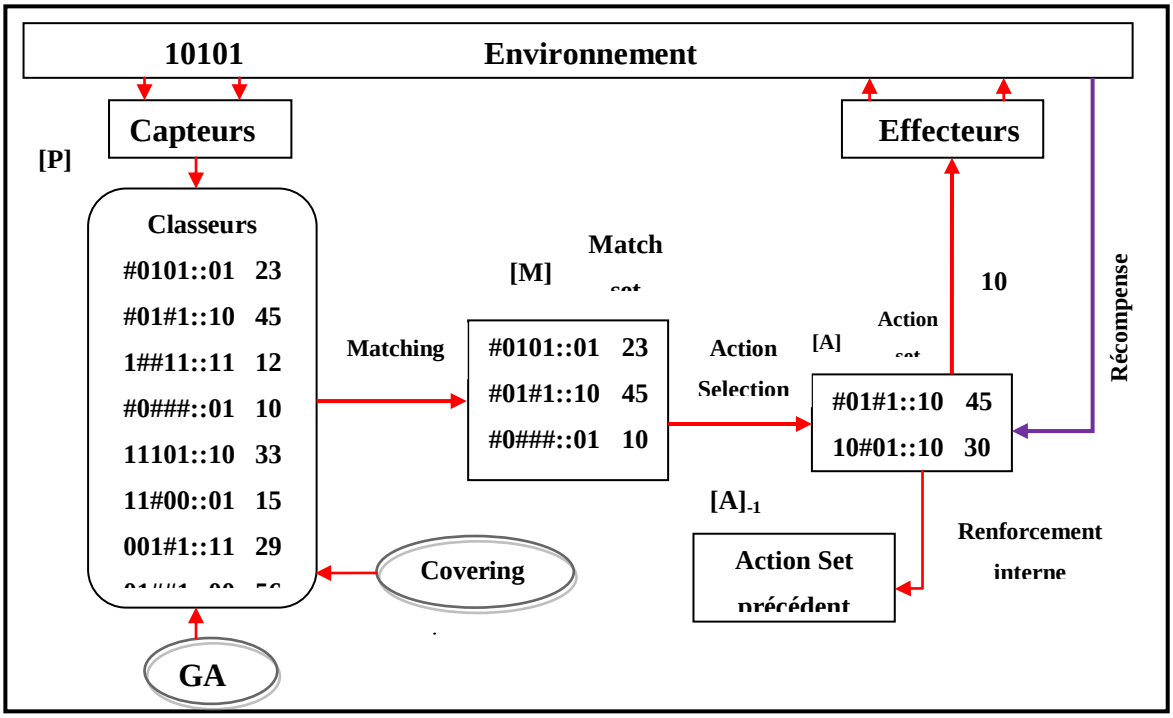


Figure 2.12: Schéma fonctionnel des ZCS

Le cycle de déroulement d'une itération dans un ZCS est globalement identique à celui des LCS de Holland mais il comporte toutefois des différences sensibles.

Le fonctionnement du ZCS est similaire au LCS de Holland. Etant donnée un message entrant par les capteurs, un ensemble *M* est constitué des classeurs pouvant répondre au message entrant. S'il n'existe pas de tel classeur, ou que leur nombre est insuffisant dans [M], alors un premier opérateur, appelé opérateur de recouvrement (*covering*) et permettant la création de nouvelles règles en fonction du message entrant, est appliqué. Ensuite, la sélection par roulette est appliquée à la liste des effecteurs de [M] pour déterminer celui qui va être activé. La probabilité donnée à chaque effecteur est proportionnelle à la somme des forces des classeurs qui utilisent cette action afin de créer l'ensemble [A] (*Action Set*) des classeurs de [M] activant l'action sélectionnée. Cette action est alors directement effectuée dans l'environnement à l'aide des effecteurs. La récompense provenant de l'environnement renforce les classeurs dans *A* pour améliorer la performance du système. L'application régulière d'un algorithme génétique

à la population *P* pour explorer des nouveaux classeurs potentiels pouvant résoudre le problème demandé. Le ZCS utilise un mécanisme de découverte supplémentaire, un opérateur covering, activant dans le cas d'une absence de classeurs en réponse au message courant.

3.2.1.2. Renforcement

Afin d'assurer le renforcement du système par la mise à jour des forces des classifieurs, S. Wilson propose deux algorithmes de distribution des crédits légèrement différents: le "*bucket brigade implicite*" adapté du "*bucket brigade*", et le "*Q-Bucket brigade*" (QBB) .

Algorithme IBB

Wilson a proposé un algorithme IBB (*implicit bucket brigade*) basé sur l'algorithme BB pour redistribuer la récompense entre l'ensemble *A* à l'instant *t* et l'ensemble *A*-1 à l'instant précédent *t*-1. Son intérêt est de développer la coordination séquentielle d'actions car les environnements *mutistep* demandent une séquence d'actions pour atteindre la cible à partir d'une situation.

Dans les environnements *singlestep* demandant une seule action pour atteindre la cible, le dernier terme de la formule 2.9 est ignoré. La réalisation de l'algorithme IBB est divisée en trois étapes. D'abord, une fraction de la fitness *Fi* de chaque classeur dans l'ensemble *A* est retirée et versée dans un seau commun initialement vide *B*(*t*) (2.7).

$$B(t) = \sum_{i \in [A]} \beta \times F_i(t) \tag{2.7}$$

Avec *B(t)* le seau commun issu de [A],
β une constante représentant l'intensité du renforcement des règles (le taux d'apprentissage du système, $0 < \beta \leq 1$),
$F_i(t)$ la force du classeur *i* à l'itération *t*.

Puis, si le système reçoit une récompense *R* provenant de l'environnement après avoir exécuté une action, les classeurs de *A* partagent *également* une fraction *r(t)* de *R*.

$$r(t) = \beta \times \frac{R(t)}{n_{[A]}(t)} \tag{2.8}$$

avec β le taux d'apprentissage du système ($0 < \beta \leq 1$),
$n_{[A]}(t)$ le nombre de classeurs de [A] à l'itération *t*.

Enfin, une fraction *b(t)* du contenu du seau *B*(*t*) est divisée *également* entre les classeurs de l'ensemble *A*-1 si ce dernier n'est pas vide. Le seau *B*(*t*) est alors vidé et *A* remplace *A*-1.

$$b(t) = \gamma \times \frac{B(t)}{n_{[A]_{-1}}(t)} \tag{2.9}$$

Avec γ ($0 < \gamma \leq 1$) une constante représentant un facteur de réduction du seau, $n_{[A]\text{-}1}(t)$ le nombre de classeurs de $[A]_{-1}$ à l'itération *t*.

La fitness *Fi* de chaque classeur de *A* à l'instant *t* est mise à jour par la formule 2.10 (l'ensemble *A*+1 représente l'ensemble *A* à instant *t*+1 et $n_{[A]}(t)$ est le nombre de classeurs dans *A)*

$$F_i(t) \leftarrow F_i(t) - \beta \times F_i(t) + \beta \frac{R(t)}{n_{[A]}(t)} + \gamma \times \frac{\sum_{j \in [A]_{+1}} \beta \times F_j(t+1)}{n_{[A]}(t)} \tag{2.10}$$

Avec β le taux d'apprentissage du système ($0 < \beta \leq 1$),
γ le facteur de réduction du seau commun ($0 < \gamma \leq 1$),
R(t) la récompense de l'environnement pour l'action produite à l'itération t,
$n_{[A]}(t)$ le nombre de classeurs de [A] à l'itération t,
$F_j(t+1)$ la force d'un classeur j de $[A]_{+1}$.

Algorithme QBB.

Wilson a proposé une autre formule de mise à jour des forces, appelée QBB (*Q-Bucket Brigade*). Elle est différente de l'algorithme IBB au niveau du dernier terme de la formule 2.10. En fait, l'algorithme QBB prend en compte la valeur maximale des fitness des classeurs des ensembles A_{+1}. Si on récrit la formule 2.11, la formule 2.12 est similaire à

celle de Watkins [Wat89] qui est utilisée dans l'apprentissage par renforcement. M_{+1} indique l'ensemble M en réponse au message entrant à l'instant t+1.

$$F_i(t) \leftarrow F_i(t) - \beta \times F_i(t) + \beta \frac{R(t)}{n_{[A]}(t)} + \gamma \times \frac{\max_{A_{+11} \subseteq M_{+1}} \sum_{Aj \in [A]_{+1}1} \beta F_j(t+1)}{n_{[A]}(t)} \tag{2.11}$$

$$F_i(t) \leftarrow F_i(t) + \beta \left(\frac{R(t)}{n_{[A]}(t)} + \gamma \times \frac{\max_{A_{+11} \subseteq M_{+1}} \sum_{Aj \in [A]_{+1}1} F_j(t+1)}{n_{[A]}(t)} - F_i(t) \right) \tag{2.12}$$

Taxation des classeurs inactifs

Quelque soit l'algorithme précédent appliqué, le cycle de renforcement présente toutefois une dernière opération. En effet, afin de permettre au système de converger plus rapidement vers un choix d'action définitif en accord avec les messages entrants, les classeurs de [M] qui ne sont pas présents dans l'ensemble [A] sont alors taxés. La mise à jour, pour l'itération *t*, des forces $F_k(t)$ des classeurs *k* de l'ensemble défini par [M]-[A] se fait par la formule (2.13) :

$$F_k(t) \leftarrow F_k(t) - \tau \times F_k(t) \tag{2.13}$$

avec τ une constante représentant le coefficient de taxe ($0 < \tau \leq 1$).

3.2.1.3. Mécanisme de découverte.

La découverte du ZCS utilise un AG panmictique (global) et un opérateur de recouvrement (*covering*) supplémentaire. Un AG est appliqué à la population [P] pendant l'exploration. L'AG est réalisé avec une probabilité *g* fixée par l'utilisateur afin d'éviter la perturbation de l'apprentissage du système à cause de l'exécution de l'AG trop souvent ou trop rarement. Quand l'AG est activé, deux classeurs parents sont sélectionnés par une roue de la fortune basée sur la fitness, leurs classeurs enfants sont générés via deux opérateurs génétiques : la mutation avec une probabilité μ et le croisement avec une probabilité χ, et enfin les classeurs enfants sont ajoutés à la population *P*. Leur fitness est héritée des fitness de leurs parents et/ou la moyenne des fitness de leurs parents si le croisement apparaît, ou

un pourcentage de la fitness moyenne des classeurs de P (comme 10%). Pour maintenir la population [P] à une taille constante, deux classeurs dans [P] sont sélectionnés en utilisant la roue de la fortune basée sur la fitness inverse. Ils sont supprimés pour laisser leur place aux classeurs enfants. L'AG est désactivé pendant l'exploitation pour arrêter l'évolution des classeurs mais ces derniers sont encore évalués par le processus de renforcement. L'opérateur *covering* est activé si une des ces conditions est satisfaite : aucun classeur existant dans la population [P] ne peut répondre au message courant (l'ensemble *M* est vide) ou la fitness totale des classeurs de *M* est inférieure à une fraction Φ de la fitness moyenne des classeurs de [P]. L'opérateur *covering* génère un nouveau classeur dont l'action est choisie aléatoirement. Sa condition est le message courant mais ses éléments sont remplacés avec une probabilité *p#* par des symboles # pour généraliser la condition de classeur Le nouveau classeur est ajouté dans [P] et classé dans *M*, et un classeur est supprimé comme dans l'AG, l'opérateur *covering* permet au système d'assurer la diversité génétique, d'explorer des actions différentes ainsi que de tester une relation condition-action représentée dans le nouveau classeur. Il est activé pendant l'exploration et l'exploitation, mais il ne fonctionne que lorsque cela est nécessaire car son appel permanent empêche l'évolution des classeurs par l'AG.

3.2.1.4. ZCS étendus

ZCSM: ZCS avec Mémoire

Afin d'imiter le rôle tenu par la liste de messages et, ainsi, d'améliorer les performances des ZCS dans les environnements non-Markoviens, S. Wilson a suggéré d'ajouter une mémoire temporaire (qui faisait défaut à son système) en ajoutant des bits supplémentaires (appelés registres) à ses classeurs [Wil95].

Le rôle de ce registre au niveau des classeurs est effectué par l'ajout des informations d'un capteur interne au niveau de la partie condition, et d'un effecteur capable de modifier la mémoire temporaire au niveau de la partie action [Wil95].

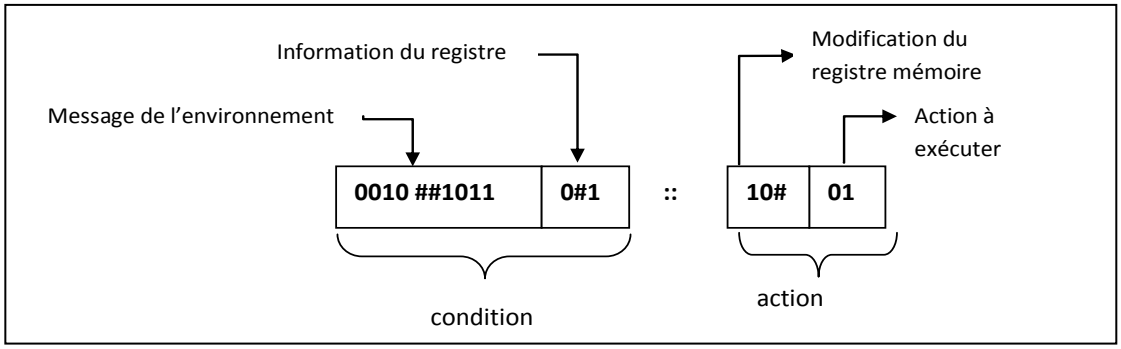


Figure 2.13: Classeur type d'un ZCSM [Wil95].

Cette idée a été implémentée par D. Cliff [CR95] en 1995 sous le nom de ZCSM. Grâce à cette méthode, il a montré qu'il était effectivement possible d'étendre les capacités des ZCS à des environnements non-markoviens tels Woods101.

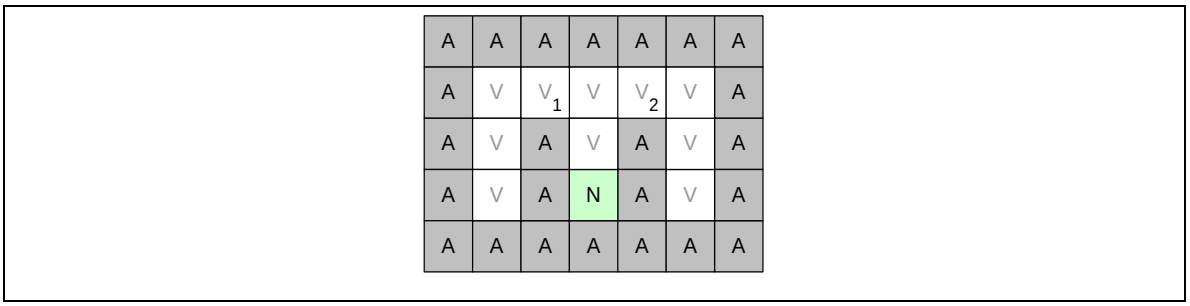


Figure 2.14: L'environnement Woods101[CR95].

Dans ses résultats, D. Cliff montre qu'un bit supplémentaire pour la mémoire interne ainsi que pour l'effecteur interne est suffisant pour que l'Animat arrive à retrouver sa nourriture. De plus, il a testé le ZCSM dans des environnements contenant plusieurs positions ambiguës, et a montré que, même s'il n'y a pas assez de bits pour représenter les états internes, le système converge vers une solution optimale.

CCS: Corporate ZCS

Tomlinson (A.) [TB99] a modifié le ZCS en ZCCS en y ajoutant le principe de la *corporation*. Il définit les liens entre classeurs grâce à des listes chaînées. Chaque règle possède deux paramètres : un lien vers le classeur précédent eu un lien vers le classeur suivant. Le couplage s'effectue aléatoirement avec des règles appartenant à d'autres

corporations, avec la probabilité 0.1. Si une règle possède déjà deux liens actifs, le couplage est appliqué en début ou fin de la chaîne formée par cette règle.
L'opérateur de croisement s'effectue entre corporations et permet la formation de nouvelles corporations ainsi que de nouvelles règles (figure 2.15).

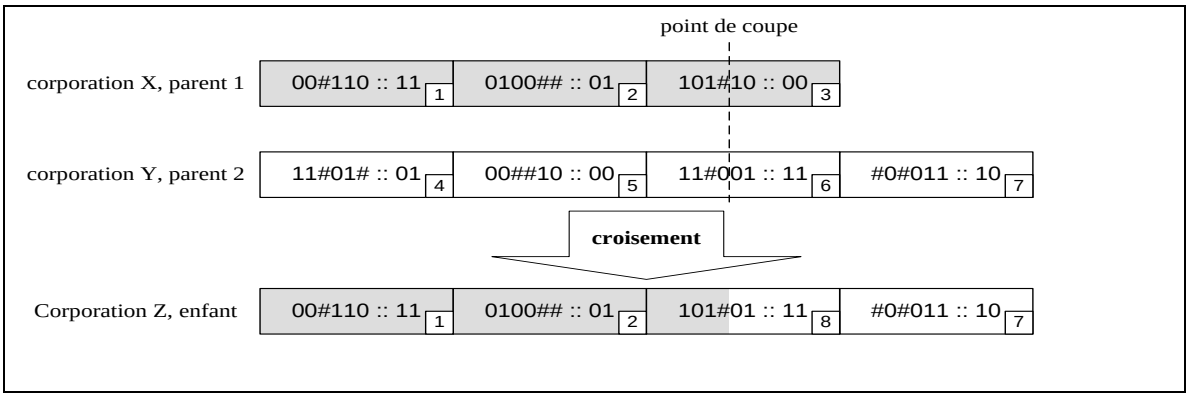


Figure 2.15: Le croisement corporatif. Les corporations parentes X et Y génèrent un enfant unique par croisement. A partir du point de coupe sélectionné aléatoirement, les règles 3 et 6 produisent la nouvelle règle 8. La conservation des liens sur le classeur précédent de 3 et suivant de 6 permettent la formation de la nouvelle corporation Z [TB99].

L'opérateur de mutation est appliqué de façon traditionnelle sur tous les classeurs de la base de règles.

Tomlinson a évalué le CCS dans l'environnement Woods1 et Woods7 avec 400 classeurs et il l'a comparé aux ZCS. Son approche s'est révélée légèrement plus efficace dans Woods1 (environnement markovien) mais moins efficace dans Woods7 (environnement non markovien). Il a ensuite testé les deux systèmes sur un problème de type multiplexeur et a remarqué que la corporation s'avérait avantageuse en début de simulation, mais que ses effets diminuaient une fois l'apprentissage de la tâche terminé.

3.2.2. eXtended Classifier System de Wilson (XCS)

Wilson a introduit le XCS [Wil 95] [Wil 98] une extension du ZCS, doté des mécanismes supplémentaires afin de pallier la prédominance des classeurs surgénéralisés et la difficulté

d'obtenir une solution optimale dans les environnements nécessitant une planification importante. Ses développements fondamentaux sont : la fitness de classeur est basée sur la précision de la prédiction et un maximum de classeurs généralisés et précis est obtenu au fil de l'évolution au lieu de découvrir des classeurs ayant la grande prédiction. Le XCS tend à construire une carte complète et précise de prédictions sous la forme *Condition* x *Action* => *Prédiction*. Il essaie d'éliminer des classeurs surgénéralisés.

L'application de l'AG à l'ensemble *A* au lieu de la population *P* limite la perte de la diversité génétique et augmente la vitesse de la convergence.

3.2.2.1. Fonctionnement des XCS

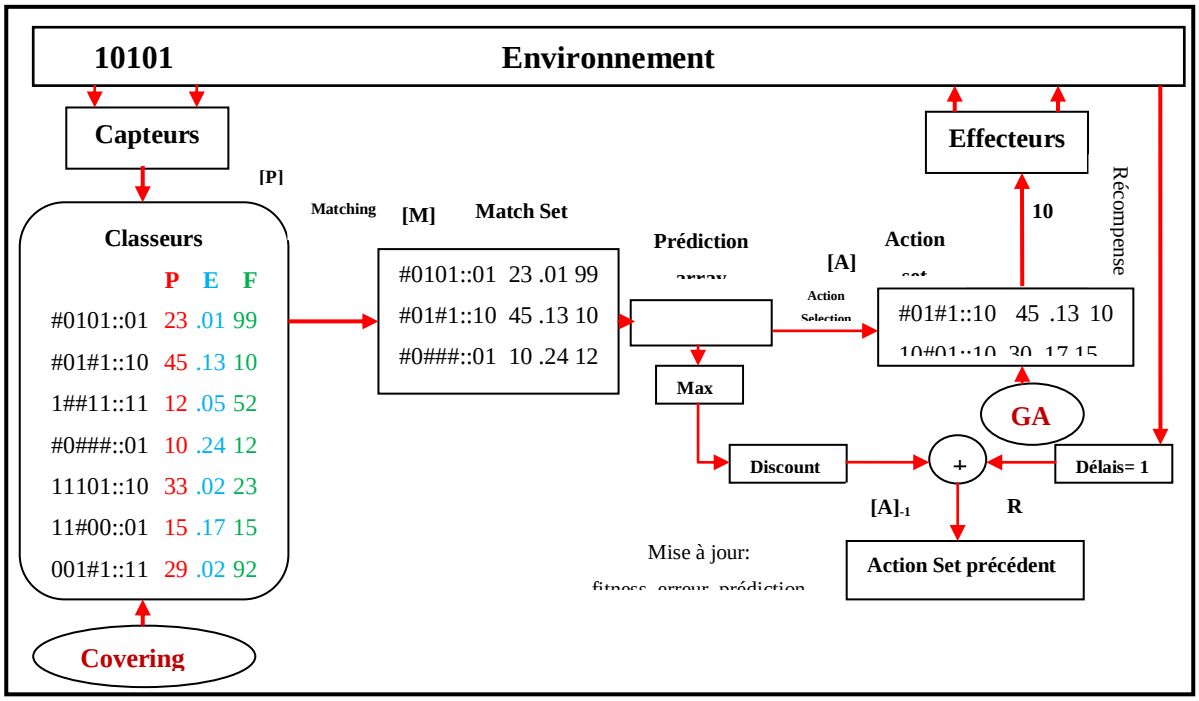


Figure 2.16 Schéma fonctionnel des XCS

Le fonctionnement du XCS est similaire au ZCS (figure 2.16). Etant donné un message entrant par les capteurs, Un ensemble M de classeurs est créé à partir de ceux qui satisfont le message provenant de l'environnement (condition unique).

Un *tableau de prédictions* est construit, dont chaque élément *p* (*ak*) représente une information associée pour une action possible *ak* dans *M*. L'action gagnante a_j est

sélectionnée à partir des prédictions $P(a_j)$ de diverses façons. Un ensemble *A* est construit de classeurs déclenchant la même action exécutée. La récompense provenant de l'environnement renforce les classeurs dans *A* pour améliorer la performance du système.

Un algorithme génétique est appliqué régulièrement à *A* pour explorer des nouveaux classeurs potentiels pouvant résoudre le problème demandé.

Un opérateur *covering* est activé dans le cas d'une absence de classeurs en réponse du message courant. Le renforcement interne dans l'ensemble précédent *A*-1 utilise l'algorithme QBB permettant de chaîner les comportements si l'environnement est *multistep.*

3.2.2.2. Structure des classeurs

Dans les systèmes de classeurs traditionnels (LCS et ZCS), la *force* représente à la fois la *prédiction* de la récompense à venir (par son rôle au niveau de la résolution de conflits dans le choix du classeur à activer) et la *fitness* indiquant les classeurs les mieux adaptés lors de l'application de l'algorithme génétique. Dans le XCS, la force est séparée et remplacée par trois nouveaux paramètres : la *prédiction p* estimant la récompense à venir, l'*erreur de prédiction* ε mesurant l'erreur commise par la prédiction en comparant avec la récompense effective, La *fitness* est fonction de l'inverse de l'erreur, elle représente donc l'exactitude de la prédiction du paiement. Elle est utilisée par les algorithmes génétiques comme critère de sélection. Ceci permet au système de répondre correctement aux situations dans lesquelles la règle la plus forte, celle qui prédit la récompense la plus importante, n'est pas nécessairement celle qui déclenche l'action la plus pertinente pour la résolution du problème posé. A ces trois paramètres sont de plus associées quatre autres valeurs de nature plus "stastistique": A ces trois paramètres sont de plus associées quatre autres valeurs de nature plus "stastistique": l'expérience x_i qui indique le nombre de fois qu'un classeur *i* a été activé (son nombre de présence dans l'ensemble [A]), le *tsi* enregistre la dernière itération à laquelle le classeur a participé dans l'ensemble *A*, ta_i la taille moyenne des ensembles [A] dont le classeur *i* a fait partie, et la numérosité *ni* indique le nombre de copies du classeur, que ce soit au niveau de l'initialisation, du recouvrement ou de l'algorithme génétique, s'il existe un classeur ayant la condition et l'action identiques dans la population *P*, Dans le cas contraire, le nouveau

classeur sera ajouté à la base de règles avec une numérosité initiale égale à 1. De cette manière, lorsque le système supprime un classeur, sa numérosité est décrémentée d'une unité et il n'est que retiré de la population si sa nouvelle numérosité est nulle.

Ainsi, un classeur de numérosité *n* est l'équivalent structurel de *n* classeurs "classiques". On parlera alors de macroclasseur. L'utilisation des macroclasseurs est un moyen permettant au système d'augmenter la vitesse de calcul du système (augmenter sa réactivité), Aussi, lorsqu'un macroclasseur doit être effacé de la base de règle,

La population de classeurs est traitée comme elle contenait *N* classeurs classiques quoi que le nombre actuel de macroclasseurs *M* puisse être inférieur à *N*. Cependant, le système doit tenir compte, s'il y a lieu, des numérosités. Par exemple, un macroclasseur de numérosité *n* est traité comme il est *n* classeurs séparés, ainsi il reçoit *n* fois la récompense qu'un classeur classique recevrait. Wilson a constaté que le système de macroclasseurs n'a pas un comportement différent du système de classeurs classiques. Pour plus de clarté, le terme *classeur* continue à être utilisé à la place du macroclasseur, et le terme *macroclasseur* est explicitement cité en cas de nécessité.

3.2.2.3. Résolution de conflit

Le XCS utilise un simple mécanisme pour équilibrer entre l'exploration de l'espace de recherche et l'exploitation de la connaissance obtenue dans le passé afin de tester la performance.

Pendant l'exploration, le système sélectionne aléatoirement une action à partir d'une liste d'actions possibles dans l'ensemble [M].

Pendant l'exploitation, la sélection de l'action se décompose en deux étapes. D'abord, le système construit un tableau de prédictions (*prediction array*) dont chaque élément représente la prédiction moyenne $p_a(t)$ pondérée par la fitness pour une action parmi les actions possibles dans [M]. La prédiction moyenne *p* (*ak*) est calculée de manière suivante (2.14).

$$p_a(t) = \frac{\sum_{i \in [M]_a} P_i(t) \times F_i(t)}{\sum_{i \in [M]_a} F_i(t)} \qquad (2.14)$$

Enfin, le système sélectionne toujours l'action *a* ayant la prédiction moyenne $p_a(t)$ la plus élevée dans le tableau de prédictions, permettant de maximiser la récompense. Le XCS pondère la prédiction par la fitness afin d'éviter la sélection des classeurs ayant la grande prédiction mais moins précise.

3.2.2.4. Renforcement

Dans les XCS, le cycle de renforcement consiste à mettre à jour les paramètres de prédiction P_i, d'erreur commise sur la prédiction E_i, et de fitness F_i des classeurs actifs *i* du système, soit les classeurs de l'ensemble [A] dont la partie *action* commune a été transmise à l'environnement. De plus, sont intégrés dans ce cycle les mises à jour des paramètres
"statistiques" des règles actives: la taille de l'ensemble [A] et l'expérience des classeurs.
La récompense *R* se compose de la récompense immédiate r_{imm} au retour de l'exécution de l'action dans l'environnement, et la prédiction maximale *max(p(a))* du tableau de prédictions de l'itération suivante réduite par un facteur γ si l'environnement est *multistep*. Le facteur γ ($0<\gamma\leq 1$) est utilisé pour réduire l'impact des récompenses des itérations suivantes. (M_{+1} est l'ensemble *M* formé correspondant au message de l'itération suivante).

$$R = r_{imm} + \gamma \cdot \max_{j \in M_{+1}}(p(a_j)) \tag{2.15}$$

Soit *x* le paramètre à estimer, *x* représente aussi sa valeur courante et *x'* est sa nouvelle valeur. La règle *delta rule* [WH60] met à jour le paramètre *x* dans la direction où l'erreur diminue (formule (2.16)). Le paramètre *x* s'approche lentement vers *x'* et la vitesse de correction est contrôlée par β. La règle fait converger *x* à sa valeur correcte.

$$x \leftarrow x + \beta \cdot (\underbrace{x' - x}_{erreur}) \tag{2.16}$$

Le processus de renforcement du XCS consiste à mettre à jour les paramètres des classeurs *i* dans *A* :

- L'expérience xp_i est simplement incrémentée d'une unité (formule (2.17))

- La taille moyenne des niches sze_i, l'erreur de prédiction ε_i et la prédiction p_i sont mises à jour par la règle *delta rule* avec le taux d'apprentissage β ($0<\beta\leq 1$) (formule (2.18)-(2.20)). La prédiction p_i est corrigée par la récompense R et l'erreur de prédiction ε_i est corrigée par la différence absolue $|R-p_i|$.

$$xp_i \leftarrow xp_i + 1 \tag{2.17}$$

$$sze_i \leftarrow sze_i + \beta \cdot (\sum_{j \in A} n_j - sze_i) \tag{2.18}$$

$$\varepsilon_i \leftarrow \varepsilon_i + \beta \cdot (\underbrace{|R - p_i(a)|}_{Erreur} - \varepsilon_i) \tag{2.19}$$

$$p_i \leftarrow p_i + \beta \cdot (R - p_i(a)) \tag{2.20}$$

- Enfin, la fitness F_i est mise à jour par la règle *delta rule* (formule (2.21)-(2.23)).

$$\kappa_i = \begin{cases} 1 & \text{si } \varepsilon_i < \varepsilon_0 \\ \alpha \left(\frac{\varepsilon_i}{\varepsilon_0} \right)^{-\nu} & \text{sinon} \end{cases} \tag{2.21}$$

$$\kappa'_i = \frac{\kappa_i \times n_i}{\sum_{j \in A} \kappa_j \times n_j} \tag{2.22}$$

$$F_i \leftarrow F_i + \beta \cdot (\kappa'_i - F_i) \tag{2.23}$$

Où n_i est la numérosité d'un classeur i. La constante ε_0 représente le seuil de tolérance pour l'erreur de prédiction ε_i. Si ε_i est inférieure à ε_0, l'erreur est acceptée et le classeur i est considéré comme précis ($\kappa_i=1$). Sinon, sa précision diminue rapidement et elle est contrôlée par une fonction de précision avec les paramètres α et ν. La précision relative κ'_i est calculée par rapport aux classeurs de [A].

La technique MAM (Moyenne Adaptative Modifiée) [Ven94] modifie la règle *delta rule* afin que les N premières valeurs de x' puissent influencer x. Elle fait la moyenne des N premières valeurs de x', où N est le plus grand entier inférieur ou égal à β^{-1}. Cette technique est utile si on ne connaît pas la valeur initiale de x. Après avoir utilisé ces N

premières valeurs, la technique MAM reprend la règle *delta rule* (formule (2.24)).

$$x \leftarrow x + \begin{cases} \dfrac{x' - x}{N} & \text{si } N < \beta^{-1} \\ \beta \cdot (x' - x) & \text{sinon} \end{cases} \tag{2.24}$$

Alors, la taille moyenne des niches sze_i, l'erreur de prédiction ε_i et la prédiction p_i sont mises à jour par la technique MAM avec le taux d'apprentissage β ($0<\beta\leq 1$) (formule (2.24)-(2.26)). La fitness F_i est toujours mise à jour par la règle *delta rule*.

$$sze_i \leftarrow sze_i + \begin{cases} \dfrac{(\sum_{j \in A} n_j - sze_i)}{xp_i} & \text{si } xp_i < \beta^{-1} \\ \beta \cdot (\sum_{j \in A} n_j - sze_i) & \text{sinon} \end{cases} \tag{2.25}$$

$$\varepsilon_i \leftarrow \varepsilon_i + \begin{cases} \dfrac{\left(\left|R - p_i(a)\right| - \varepsilon_i\right)}{xp_i} & \text{si } xp_i < \beta^{-1} \\ \beta \cdot (\underbrace{\left|R - p_i(a)\right|}_{Erreur} - \varepsilon_i) & \text{sinon} \end{cases} \tag{2.26}$$

$$p_i \leftarrow p_i + \begin{cases} \dfrac{\left(R - p_i(a)\right)}{xp_i} & \text{si } xp_i < \beta^{-1} \\ \beta \cdot \left(R - p_i(a)\right) & \text{sinon} \end{cases} \tag{2.27}$$

3.2.2.5. La découverte des nouveaux classeurs

La découverte des nouveaux classeurs est assurée dans les XCS, comme dans les ZCS, par un algorithme génétique couplé à un opérateur de recouvrement.

L'opérateur de recouvrement

L'opérateur de recouvrement (qui permet de créer, et d'insérer dans la population [P], des nouveaux classeurs en fonction du message entrant lorsque le système ne peut définir un ensemble [M] suffisant à partir des règles déjà existantes) est en tout point identique à celui utilisé pour les ZCS.

L'algorithme génétique (AG)

A la différence des ZCS, dans les XCS l'algorithme génétique n'est plus appliqué à la population [P] des classeurs mais à une niche de classeurs particulière constituée par l'ensemble [A] des règles qui activent une action unique en fonction du message entrant. Hormis le changement d'ensemble de classeurs sur lequel il est appliqué, le déroulement de l'algorithme génétique est identique pour les ZCS et les XCS. Cependant, l'algorithme génétique ne s'exerçant dorénavant que sur les ensembles [A], il est inutile d'opérer le croisement sur l'intégralité des classeurs car les parties *action* sont identiques. Ainsi, cet opérateur ne s'effectue (généralement en utilisant un croisement uniforme) que sur les parties *condition* des individus parents. La mutation, quant à elle, s'applique à la fois sur la *condition* et l'*action* afin de favoriser l'exploration de l'espace de recherche.

Enfin, pour ménager un nombre d'itérations suffisant au renforcement du système entre deux applications de l'algorithme génétique sur un ensemble [A] particulier, il n'est déclenché que si la durée moyenne de la période depuis sa dernière application, t_{GA}, est supérieure à un seuil fixé par l'utilisateur θ_{GA}.

$$t_{GA} = t - \frac{\sum_{i \in [A]} ts_i \times n_i}{\sum_{i \in [A]} n_i} \tag{2.28}$$

avec t l'itération courante,
ts_i la dernière itération à laquelle le classeur i de [A] a été soumis à l'AG,
n_i la numérosité du classeur i.

Une fois l'algorithme génétique exécuté, le paramètre correspondant à l'enregistrement ts_i de la dernière itération t d'application est mis à jour pour les classeurs i de [A]: $ts_i \leftarrow t$.

Effacement des classeurs

Lorsque la population [P] des classeurs a atteint sa taille maximale N, c'est-à-dire lorsque la somme des numérosités des classeurs la composant excède N, il est alors nécessaire de procéder à l'effacement de certaines règles afin de pouvoir insérer dans [P] les nouveaux individus issus de l'opérateur de recouvrement ou de l'algorithme génétique.

La suppression d'un classeur *i* utilise une roue de la fortune dont chaque secteur est proportionnel à une valeur *d*i définie par la formule 2.29. La valeur *di* du classeur *i* dépend de sa taille moyenne de niches *szei*, de son expérience $x_i(t)$ et de sa fitness $f_i(t)$ relative à la fitness moyenne de la population $\overline{F}(t)$ formule 2.30 .S'il a suffisamment d'expérience mais sa performance est faible, ou il a participé à des niches de taille importante, alors il risque d'être sélectionné à effacer. Le système garde des classeurs performants tout en essayant de faire l'égalité des ressources (des classeurs) entre des niches.

$$d_i(t) = \begin{cases} \dfrac{ta_i(t) \times n_i(t) \times \overline{F}(t)}{f_i(t)} & \text{si } x_i(t) > \theta_d \text{ et } f_i(t) < \delta \times \overline{F}(t) \\ ta_i(t) \times n_i(t) & \text{sinon} \end{cases} \tag{2.29}$$

$$\overline{F}(t) = \frac{\sum_{i \in [P]} F_i(t)}{\sum_{i \in [P]} n_i(t)} \tag{2.30}$$

$$f_i(t) = \frac{F_i(t)}{n_i(t)} \tag{2.31}$$

Où

fi(t) la fitness pour une unité de numérosité d'un macroclasseur i de [P]:

$\overline{F}(t)$ la fitness moyenne de la population [P] à l'itération t:

Fi(t) la fitness d'un classeur i de [P] à l'itération t,

θ_d un seuil d'expérience minimale requise fixé par l'utilisateur,

δ une constante ($0 < \delta \leq 1$).

3.2.2.6. Architecture de subsomption

L'opérateur de "subsumption" a été introduit par Wilson pour d'augmenter la capacité de généralisation du XCS et de réduire la taille de la population.

Un classeur subsume un autre classeur si sa partie condition est plus générale (elle contient plus de symbole '#") et si leurs parties *action* sont identiques. Le classeur subsumé est effacé de la base de règles et sa numérosité est ajoutée à celle du classeur qui le subsume.

Cependant, afin d'éviter qu'un classeur trop général subsume un classeur plus spécifique et efficace, le classeur subsumant doit avoir suffisamment d'expérience (son expérience supérieur à un seuil θ_{sub} défini par l'utilisateur) et être précis (son erreur de prédiction inférieur à ε_0).

3.2.2.7. XCS étendus

XCSS : XCS utilisant l'opérateur "Specify"

P. Lanzi propose le XCSS pour des environnements qui ne permettent que très peu de généralisation [Lan97]. Il introduit un nouvel opérateur "specify" qui permet de remplacer certains symboles # par des valeurs binaires. Ce mécanisme agit sur l'ensemble [A] et sélectionne un classeur qui va donner naissance à un fils. Ce fils est construit avec les gènes de son père dans lesquels chaque # est remplacé par les bits de la condition entrante avec une probabilité P_{Sp}.

Comme pour l'opérateur Mutespec [DS93], l'opérateur "specify" supprime les oscillations dues à une surgénéralisation. Ce type de modification est donc particulièrement bien adapté aux environnements complexes. Il a donc testé son système avec Maze4 (figure 2.17) ou le XCS échoue habituellement à cause de son instabilité. Ici, le XCSS atteint des performances optimales en 500 cycles d'exploitation. Il a ensuite utilisé l'environnement Woods2 qui est moins complexe que Maze4 et il a montré que le "specify" n'élimine pas la tendance à généraliser comme pour l'architecture originale mais ralentit plutôt le processus de généralisation. Malgré cela, avec des performances quasi-équivalentes, le XCSS utilise 15% de règles en moins.

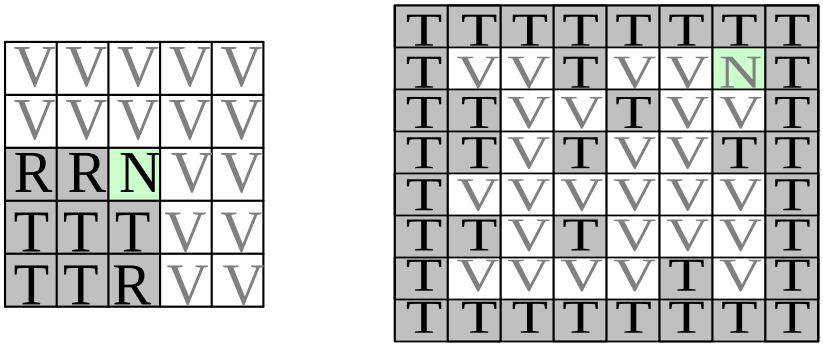


Figure 2.17 : Les environnements Woods2 et Maze4 (T et R représentent deux types d'obstacles)

XCSM : XCS avec Mémoire

De manière similaire à l'implémentation du ZCSM par Cliff, P. Lanzi a ajouté un registre de mémoire interne au XCS [Lan98]. Le nouveau système, baptisé XCSM, est capable d'atteindre des solutions optimales dans des environnements non-markoviens simples comme Woods101 en utilisant indifféremment 1, 2 ou 3 bits de mémoire interne. Avec l'environnement Woods102 (qui possède deux positions ambiguës), le XCSM ne converge que si l'opérateur *Specify* lui est ajouté et il atteint la nourriture de façon optimale en utilisant 1600 classeurs. Cependant, le même système n'obtient qu'une solution approximative dans l'environnement Maze7: l'Animat atteint parfois les mêmes cases ambiguës avec deux configurations différentes de la mémoire interne.

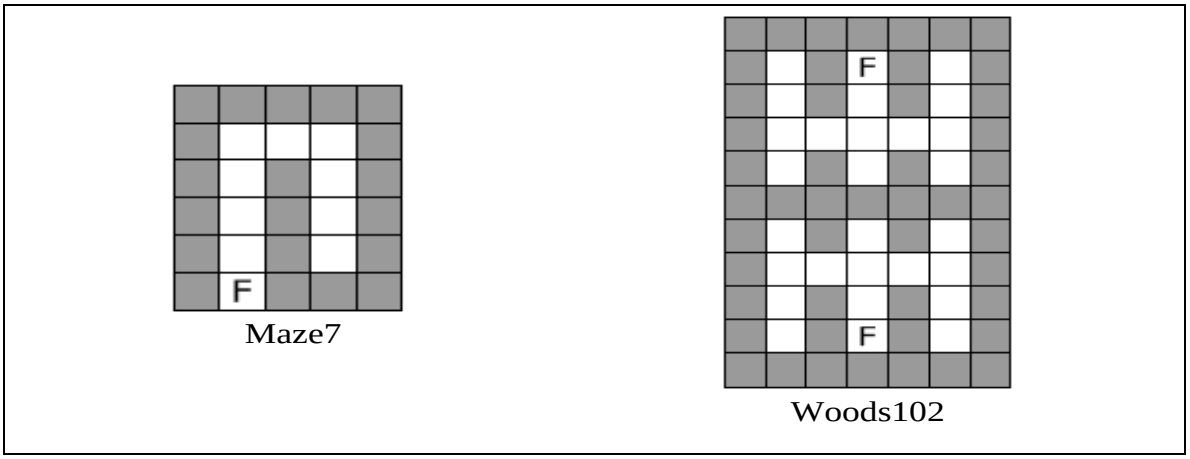


Figure 2.18: les environnements Woods102 et Maze7

Une solution optimale pour Maze7 sera obtenue par une extension des XCSM appelée XCSMH [Lan98]. Le XCSMH conserve la structure du XCSM de base mais diffère en deux points majeurs qui permettent une meilleure exploitation des registres internes dans les environnements plus complexes:

- les effecteurs internes n'agissent que dans le cas ou l'action externe correspondante modifie la perception de l'agent c'est-à-dire si l'action engagée génère un déplacement.
- il emploie une stratégie d'exploration hiérarchique : le système sélectionne une action interne de manière déterministe, puis il sélectionne une action externe parmi celles qui font appel à l'action interne grâce à une stratégie d'exploration aléatoire.

CXCS : XCS avec principe de Corporation

A. Tomlinson a poursuivi son étude des systèmes corporatifs en appliquant les principes et mécanismes mis en œuvre dans les CCS aux XCS [Tom99]. Le nouveau système, CXCS, est évalué sur deux types de problèmes:

- Une comparaison entre un XCS et un CXCS dans l'environnement markovien Woods2: le CXCS converge plus vite que le XCS vers des performances optimales.
- Un test en environnement non-markovien, un DRT M:N (Delayed Reward Task), pour démontrer l'efficacité de la corporation: avec différentes valeurs de M et N, un XCS n'arrive pas à converger vers une solution alors que le CXCS obtient des performances satisfaisantes avec DRT 2:2 et DRT 4:2. Par contre, le CXCS voit ses performances diminuer avec l'augmentation de N (N=3), tout en restant nettement supérieur au XCS.

3.2.3. ACS : Anticipatory Classifier System

3.2.3.1. Principe

Un ACS (Anticipatory Classifier System) est un type dérivé de système de classeurs classique assez original. Apparus en 1996 grâce aux travaux de W. Stolzmann [Sto96], les ACS sont basés sur un mécanisme d'anticipation du comportement introduit par J. Hoffmann en 1993 [Hof93].

J. Hoffmann [Hof93] a émis l'hypothèse que l'anticipation des conséquences induites par une action est un facteur décisif dans la génération de comportements. Ainsi, il a défini un modèle où le système apprend à déterminer l'état final E à partir d'une action R exécutée dans un état initial S (figure 2.19).

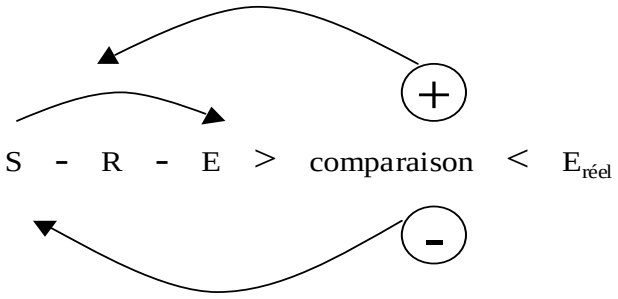


Figure 2.19 : Modèle d'anticipation du comportement.

Ce modèle évalue un triplet (E, R, S) de deux façons différentes. Une fonction de fitness classique permet de renforcer ou d'affaiblir le triplet en fonction des récompenses provenant de l'environnement. Ensuite, le modèle compare l'état réel $E_{réel}$ obtenu lors du déclenchement de l'effecteur R avec l'état anticipé E_{ant} de manière évaluer la pertinence de l'anticipation.

Les ACS utilisent ce principe d'anticipation pour produire des comportements sous forme de règles binaires. Nous allons présenter dans les paragraphes suivants le fonctionnement des ACS, les possibilités induites par cette architecture et quelques applications récentes.

3.2.3.2. Architecture

Les ACS sont constitués de 4 composants de base :

- une interface d'entrée avec des capteurs
- une interface de sortie avec des effecteurs
- une liste de messages envoyés par les capteurs
- une liste de classeurs qui consiste la base de règles

Chaque capteur de l'interface d'entrée génère une information booléenne sur un attribut de l'environnement. Il délivre donc une valeur 0 ou 1. L'ensemble de "d" capteurs du système sensoriel produit alors un message binaire de la forme $\{0,1\}^d$. Jusqu'à ce niveau, l'architecture d'un ACS est comparable à un système de classeurs de type ZCS.

La principale différence provient de la définition d'une règle. Un classeur est généralement constitué de deux parties : condition et action. Avec les ACS, une règle est composée de 3 parties correspondant au triplet (E, R, S) défini par Hoffmann.

La partie condition : spécifie les attributs de l'environnement permettant la sélection et le déclenchement de la règle (noté S),

La partie action : contient l'effecteur (noté R) et la partie 3 représente l'état induit par l'action de l'effecteur R (noté C), c'est à dire l'état anticipé par le classeur.

Les messages S et C sont de la forme $\{0, 1, \#\}^d$ tandis que les messages R sont de la forme $\{0, 1, \#\}^e$ (e étant le nombre d'effecteurs).

Une autre différence entre les ACS et les SC concerne la force d'un classeur. Dans un ACS, chaque règle possède deux forces. La première force (notée s_c) évalue la récompense de l'environnement. La deuxième (notée s_c^{ant}) évalue l'adéquation de l'état espéré avec l'état réellement obtenu.

Le fonctionnement d'un ACS est semblable à un SC classique, excepté les quelques modifications apportées sur la structure d'un classeur. La figure 2.20 représente un cycle complet d'évolution d'un ACS.

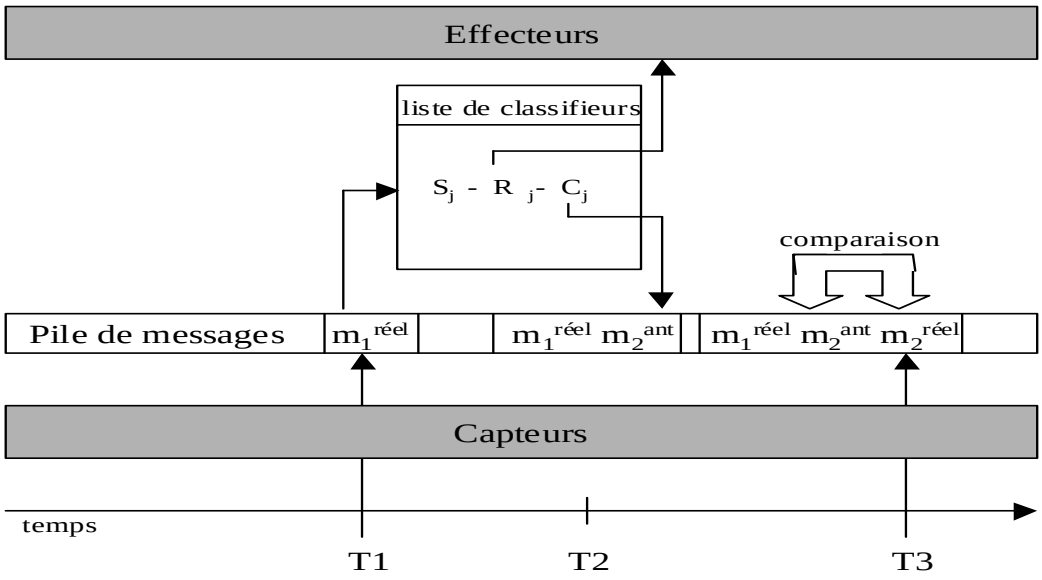


Figure 2.20 : Cycle d'évolution d'un ACS

Le nombre d'étapes d'un cycle d'évolution d'un ACS est un peu plus important que pour un SC classique. Ces différentes étapes sont les suivantes :

1 : A l'instant T1, les capteurs envoient un message m_1^{reel} qui décrit l'état actuel de l'environnement.

2 : Ce message est comparé avec les parties conditions de chaque classeur (partie S) pour déterminer ceux qui satisfont le message m_1^{reel} et qui vont entrer en compétition.

3 : Le ACS décide du gagnant en choisissant celui qui propose l'enchère b(t) la plus haute, avec $b(t)=s_c(t).s_c^{ant}(t).2^{spec(t)}$ et spec(t) : nombre de symboles 0 et 1 du message S.

4 : A l'instant T2, un classeur j devient actif et le message m_2^{ant} est ajouté à la pile de messages. La partie action du classeur j déclenche une action dans l'environnement. La force s_c du classeur est mise à jour grâce à une fonction de fitness classique.

5 : A l'instant T3, les capteurs envoient le nouveau message de l'environnement m_2^{reel} à la pile de messages.

6 : Le système compare m_2^{ant} et m_2^{reel}. En fonction de cette comparaison, le classeur est récompensé ou pénalisé.

7 : Les messages $m1^{reel}$ et m_2^{ant} sont supprimés de la pile de messages. Retour à l'étape 2.

Ce type de système permet donc de générer des classeurs permettant d'agir efficacement dans l'environnement et de prévoir l'état qui va apparaître après l'exécution de l'effecteur. Une conséquence de cette caractéristique est la possibilité d'optimiser le fonctionnement d'un ACS d'un point de vue temps de calcul.

3.2.3.3. Optimisation par séquence de comportements

Après quelques cycles d'utilisation, certains classeurs peuvent révéler une force d'anticipation s_{ant} assez élevée. Cela signifie que la probabilité pour que m_2^{ant} et m_2^{reel} soient identiques est très forte. De tels classeurs sont appelés "sûrs" [Sto96].

Ainsi, après avoir utilisé un classeur "sûr", il est possible d'utiliser m_2^{reel} comme message m_1^{reel} lors du cycle suivant. Le calcul de m_1^{reel} n'est donc plus nécessaire et, en itérant le processus, on diminue le nombre d'opérations au niveau de l'activation du système sensoriel, de la sélection et du calcul de l'adéquation entre m_2^{ant} et m_2^{reel}.
La séquence des comportements forme donc une chaîne représentée sur la figure 2.21.

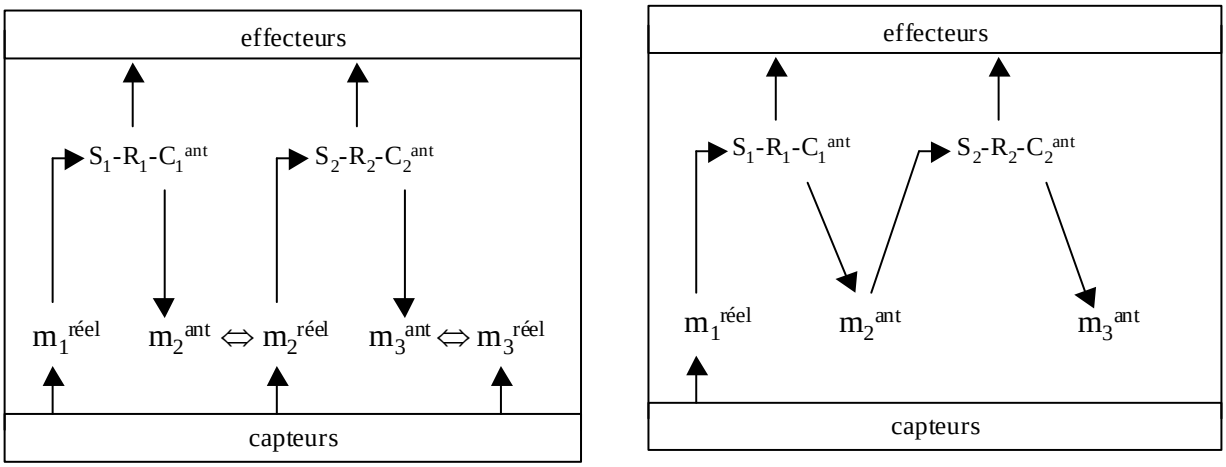


Figure 2.21 : Génération d'une chaîne de deux classeurs

3.2.4. YCS de Bull

Bull [Bul03] a présenté le YCS dont le but est de clarifier la compréhension de la fitness basée sur la précision du XCS [Wil95]. En effet, le YCS est basé sur le XCS mais il simplifie ce dernier. Il est différent au XCS par le fait qu'il n'applique pas l'AG à l'ensemble *A* et il n'utilise pas l'architecture de subsomption. Donc, il n'a pas de mécanismes pour obtenir une carte complète sur l'espace état-action. Mais Bull a montré dans les problèmes 6- et 11-multiplexeurs que ce ne sont pas de conditions préalables pour obtenir une performance optimale.

Le YCS se compose d'un ensemble de classeurs dont les parties conditions et actions sont définies comme dans le XCS. Chaque classeur a quatre paramètres principaux. Une prédiction p_j estimant une prédiction à venir est utilisée dans la sélection de l'action. Une erreur de prédiction ε_j indique une estimation de l'erreur de la prédiction du classeur. Une fitness F_j définissant par l'inverse de l'erreur de prédiction est utilisée dans la sélection de classeur lors de l'application d'un AG, et une taille de niches α_j estime la taille moyenne de niches (des ensembles *A*) auxquelles le classeur a participé. Ces paramètres sont initialisés arbitrairement à leur valeur prédéfinie. L'ensemble *M* est formé comme dans le XCS lors de la réception d'un message entrant. Le YCS peut utiliser différentes stratégies de sélection de l'action pour regrouper des classeurs ayant la même action gagnante dans un ensemble *A*, mais il utilise celle du XCS. En fait, dans un problème, une action est

sélectionnée aléatoirement et dans le problème suivant, une action ayant la prédiction la plus élevée dans un tableau de prédictions est sélectionnée de façon déterministe.

Lors de la réception d'une récompense immédiate *P*, provenant de l'environnement, les paramètres (erreur de prédiction, taille de niches et prédiction) de chaque classeur dans l'ensemble *A* sont mis à jour par la règle *delta rule* [WH60] avec le taux d'apprentissage β ($0<\beta\leq1$) comme dans le XCS sauf pour la fitness de classeur

$$F_j = \frac{1}{1+\varepsilon_j} \tag{2.32}$$

Le YCS utilise deux mécanismes de découverte : un AG panmictique avec une probabilité *g* et un opérateur *covering*. Quand l'AG est activé, une roue de la fortune basée sur la fitness est utilisée pour la sélection de classeur de reproduction et une roue de la fortune basée sur la taille de niches est utilisée pour la suppression de classeur de la population. Rien n'a changé au sujet du fonctionnement de l'opérateur *covering*.

Le YCS peut obtenir des performances optimales avec les 6- et 11-multiplexeurs, mais ses performances ne sont plus optimales avec le 20-multiplexeur en raison d'une part de l'utilisation de l'AG globale qui limite la vitesse d'apprentissage et la généralisation sur l'espace du problème. Pour favoriser la généralisation, [BSB+05] a alterné le YCS pour que l'AG opère sur les niches. D'autre part, la fonction de fitness est aussi modifiée. L'exposant *v* de la fonction de fitness modifiée facilite la séparation des classeurs dans la sélection des classeurs performants (pour se reproduire) qui est proportionnelle à la fitness.

$$F_j = \frac{1}{1+\varepsilon_j^v} \tag{2.33}$$

L'approche de l'ensemble de machines [HHS94] utilise un ensemble de modèles dont chacun propose une solution pour le problème demandé. Comme aucun modèle ne domine les autres modèles, l'objectif de cette approche est ainsi de combiner la sortie des différents modèles pour trouver une solution globale. [BSB+05] utilise un ensemble de systèmes de classeurs YCS dont chacun explore parallèlement l'espace de recherche. Etant donné un message entrant, chaque YCS propose une action en concurrence et l'action gagnante est celle ayant la majorité de votes. La figure 2.22 montre la performance d'un ensemble de 10 YCSs contre un seul YCS sur le 20-multiplexeur. Ce premier converge plus vite que ce

dernier. Cependant, l'ensemble de 10 YCSs nécessite plus de temps de calcul car il fait évoluer à la fois 10 YCSs.

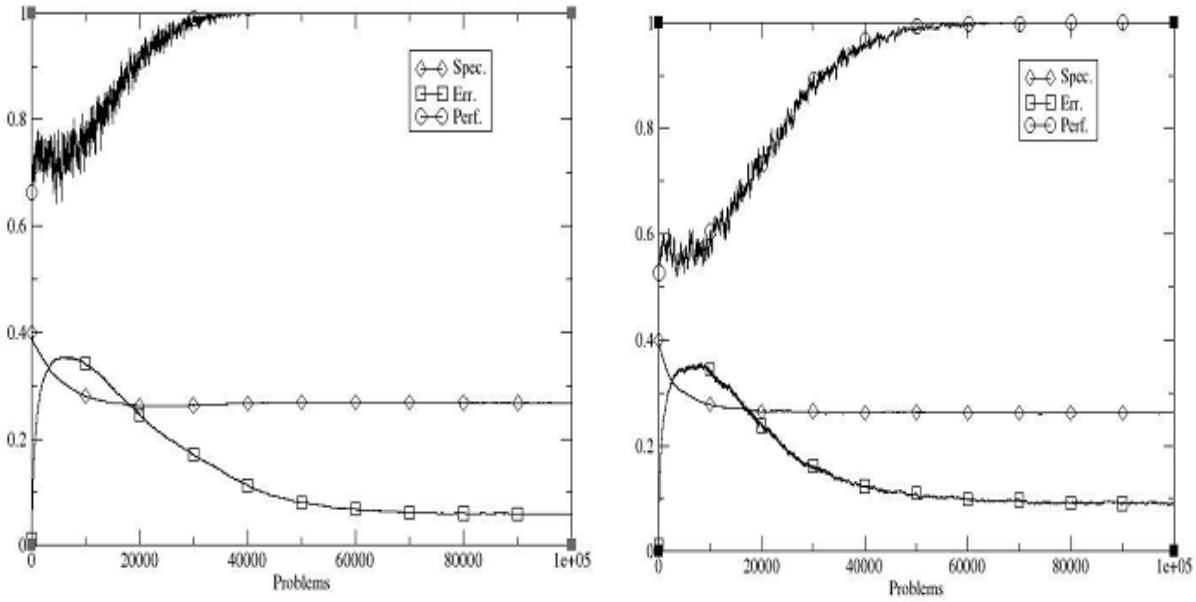


Figure 2. 22: Performance d'un ensemble de 10 YCS (à gauche) et d'un seul YCS (à droite) sur le 20-multiplexeur [BSB+05]

3.2.5. Tableau de synthèse des principaux systèmes de classeurs

Sigle	LCS	ZCS	XCS	ACS	YCS
Nom	Learning Classifier System	Zeroth level Clasifier System	Extended Classifier System	Anticipatory Classifier System	Yet Clasifier System
Auteur	Holland 1976	Wilson 1994	Wilson 1995	Stolzman 1996	Bull 2003
Codage des informations	Chaîne de bits	Chaîne de bits	Chaîne de bits	Chaîne de bits	Chaîne de bits
Génération des nouvelles règles	• Algorithme génétique sur [p]	• Algorithme génétique sur [p]. • Mécanisme de Covering.	• AG sur [A] • mécanisme de covering	• AG sur [A] • mécanisme de covering	• AG sur [p] • mécanisme de covering
Particularités	• Partie action et condition de même taille. • Boucle interne.	• Basé sur LCS. • Pas de boucle interne. • Création d'un Action Set.	• asé sur ZCS. • G appliqué à A • ise à jour retardée de [A]-1 en fonction de la récompense R-1 et de la valeur maximale du tableau de prédictions Ajout de paramètre de contrôle d'erreur. • itness basée sur la performance d'un classeur.	• Précision de la prédiction d'un classeur, utilisée lors de la sélection de l'action, de l'effacement et de la subsumption • Mise à jour retardée de [A]-1 en fonction de la récompense R-1 et de la valeur maximale du tableau de prédictions • Enregistrement de la dernière itération d'application de l'AG	• Modèle simple du XCS • AG appliqué à la population P • Pas d'architecture de subsomption
Points forts	• ase de tous les classeurs Actuels. • nchaînement des classeurs grâce à la boucle interne.	• Plus rapide que les LCS. • Répercute les récompenses sur les différents classeurs qui ont amenés à la solution.	• ermet d'obtenir une carte de rétributions. • ermet d'obtenir des classeurs pertinents (erreur minimisée). • Permet d'obtenir des classeurs généralisés.	• valuation en fonction de l'état obtenu par rapport à ce qui était prédit. Enchaînement des comportements. Planification de tâches	• Obtention de la performance optimale dans certains environnements : 6- et 11-multiplexeurs malgré le modèle simple du XCS
Points faibles	• éveloppement des classeurs surgénéralisés. • rédominance des classeurs les plus forts.	• es problèmes des classeurs surgénéralisés sont réduits mais subsistent.	• Grande difficulté face aux problèmes non-Markovien (même avec mémoire) • Paramétrage complexe = nombre élevé de tests.	• Structure très lourde si on a beaucoup de paramètres. • roblèmes lorsque l'on se trouve dans un environnement markovien. • Surcoût de calcul.	• roblème de performance dans le 20-multiplexeur

4. Conclusion

Dans ce chapitre, nous avons présenté les systèmes évolutionnaires, y compris les algorithmes génétiques. En particulier, nous avons fais un tour d'horizon tout en clarifiant leurs principaux modèles, leurs notions de bases, et les différentes variantes de chacun d'eux, sous l'ombre de leur efficacités et pertinences dans la génération des comportements adaptatifs. L'accent a été mis sur le système de classeur de type YCS, car il sera le type qu'on adopte dans notre travail.

Dans le chapitre suivant, nous allons présenter notre architecture comportementale combinant différents mécanismes afin d'obtenir des comportements adaptatifs des agents autonomes et d'autres détails pertinents pour notre architecture.

Chapitre III : Le Modèle Proposé

Chapitre3

Simulation comportementale par Systèmes de classeurs

There is nothing wrong with a good illusion as long as one does not claim it is reality. " , Mark Twain.

L'objectif de notre simulation est de présenter et d'évaluer une architecture afin de simuler les comportements des agents pour réaliser une stratégie coopérative et de modéliser les interactions qu'entretiennent les entités entre elles ou avec l'environnement. Au niveau de l'individu, l'entité va essayer d'apprendre, d'évoluer et de s'adapter pour satisfaire leurs objectifs. Au niveau de groupe, les entités vont coopérer afin d'accomplir une tache collective.

Pour contourner cette difficulté, nous nous intéressons à des approches issues de la vie artificielle comme les systèmes de classeurs. L'originalité de cette méthode, par rapport aux autres systèmes d'apprentissage, est la capacité de générer des comportements adaptatifs.

Concevoir un agent qui apprend de son environnent pour mieux réagir afin d'atteindre un objectif est déjà une tâche difficile mais sa complexité augmente quand l'agent doit s'adapter pour collaborer avec d'autres agents.

Alors, notre objectif est de générer des comportements ayant des capacités d'adaptation aux intensions d'autres agents lorsque les agents réalisent une tache collective dans un environnement virtuel.

Avant de décrire notre architecture mise en place, nous allons présenter rapidement les systèmes comportementaux évolutionnaire de référence.

1. systèmes comportementaux de référence

1.1. Approche basée théorie des motor schemas de R. Arkin

Les *motor schemas* [Arkin92, Arkin98], dérivés de la méthode de champs de potentiel, sont des systèmes réactifs pour la navigation des robots mobiles. Il décompose le comportement en modules indépendants ; les motor schemas. Le motor schema est défini comme le comportement de base par exemple : *aller-à-cible, éviter-obstacles, avancer*, etc. (Figure 3.1). Ces derniers s'expriment comme champs vectoriels attractifs ou répulsifs. Les champs de potentiel ont été développés tout d'abord comme une approche d'évitement de collision d'obstacles en temps réel et présenté par Latombe en 1991 [Lat91]. La superposition de plusieurs champs vectoriels permet de déterminer une trajectoire d'un agent ou plusieurs. Le but est d'atteindre une cible sans heurter les obstacles qui peuvent être dans l'environnement.

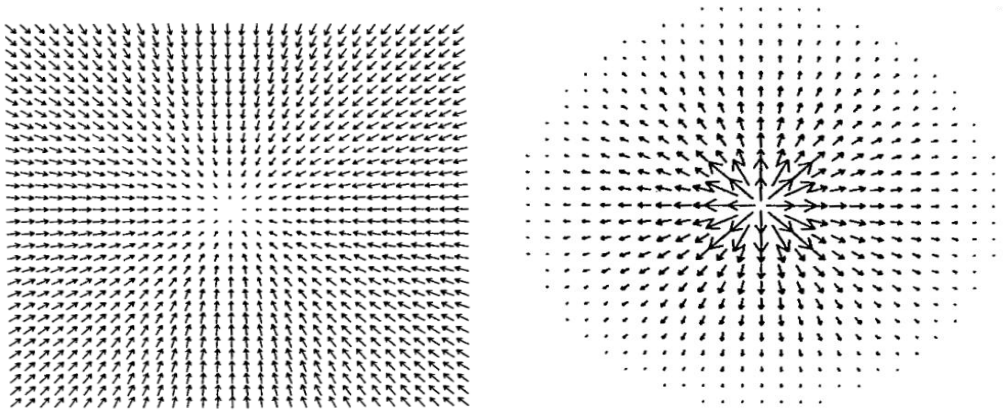


Figure 3.1 : Deux *motor schemas* : *aller-à-cible* (gauche) et *éviter-obstacles* (droite). Etant donné une position du robot, un seul vecteur pour chaque *motor schema* est calculé en réponse à sa perception [Ark92].

Arkin [Ark92] propose son modèle pour la navigation de robots autonomes. La particularité de cette approche est la représentation des motor schemas selon un unique format, un vecteur d'action. Ce vecteur est généré selon des méthodes à base de champs potentiels considérant les cibles comme des attracteurs et les obstacles comme des

répulseurs. La coordination entre les motor schemas se fait alors simplement en additionnant l'ensemble des vecteurs calculés par les motor schemas actifs, après multiplication de chaque vecteur par un poids dynamique associé à chaque motor schema. Il en résulte un unique vecteur d'actions correspondant à la direction que le robot doit suivre en réponse à sa perception. Ainsi chaque comportement de base participe à l'émergence du comportement global. Cette approche a prouvé son efficacité pour la génération de comportements surtout pour la coordination de robots au moment de l'exploration de l'environnement.

La théorie des *motor schemas* est aussi utilisée pour implémenter des formations pour un petit groupe de robots hétérogènes (quatre robots) [BA98]. Les robots naviguent dans un environnement pour atteindre une cible en évitant les obstacles et les autres robots tout en maintenant leur formation géométrique (*ligne*, *colonne*, *diamant* et *canard*) (Figure 3.2). Chaque robot doit calculer sa propre position dans la formation, appelée *point de référence*. Trois techniques pour la détermination du *point de référence* sont *centre*, *leader* et *voisin* . Balch a montré qu'en fonction de l'environnement et du déplacement qui est imposé, les formations géométriques ont des performances qui ne sont pas équivalentes. Par exemple, si les robots doivent traverser un champ d'obstacles, la formation *colonne* est la plus efficace si les techniques *centre* et *leader* sont utilisées, car elle minimise l'erreur de position (qui mesure la moyenne des déplacements par rapport à la position correcte dans la formation) et aussi le temps passé hors de la formation (car les robots peuvent quitter temporairement leur position face à la présence d'un obstacle). Par contre, pour un déplacement à 90°, c'est la formation *diamant* qui est la meilleure si la technique *centre* est utilisée, ou les formations *ligne* et *canard* sont les meilleures si la technique *leader* est utilisée.

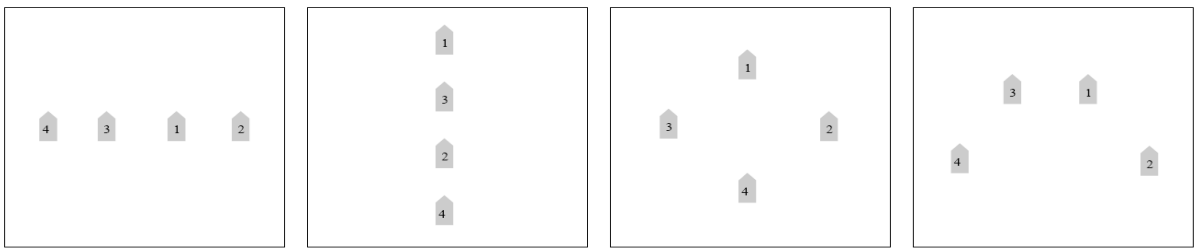


Figure 3.2 : Formations pour quatre robots numérotés 1, 2, 3 et 4 (de gauche à droite : *ligne*, *colonne*, *diamant* et *canard*) [BA98]

1.2. Systèmes comportementaux évolutionnaires

Développer une méthode de génération de comportements requiert en principe la capacité de se repérer dans l'environnement. Il est démontré [Tan04] que l'efficacité d'apprentissage se base sur la perception dans l'environnement et une communication directe ou indirecte entre les autres agents, de cette manière les contrôles comportementaux sont autonomes.

Grâce à l'apprentissage, les agents évoluent dans l'environnement pour enrichir sa base de connaissances. Il est vrai que d'autres systèmes ont été expérimentés et proposent des tâches plus complexes aux agents, où l'apprentissage est réalisé de manière incrémentale [AOF01].

Les systèmes de classeurs (SC) fournissent une réponse performante aux problèmes d'apprentissage : ils couplent une base de règles qui permet d'obtenir une réponse à un état de l'environnement, avec un mécanisme de rétribution favorisant les règles les plus productives, et un mécanisme d'évolution pour renouveler les règles de la base.

Plusieurs applications utilisent les différents types du système de classeur en simulation comportementale à fin de l'émergence des comportements individuel ou collectif ou bien la combinaison les approches évolutionnaire avec d'autre approche par exemple (les approches réactives ou /cognitives).

Sigaud et Gérard 2001 [SG01] ont développé un système de classeurs qui s'appliquent aux problèmes de contrôle des animaux grégaires. Dans le problème qu'ils proposent, un groupe d'agents (bergers) doit guider un autre groupe d'agents (moutons) vers un endroit spécifique dans l'environnement.

Chaque entité berger est dotée d'un système de classeurs, celui-ci se déplace de manière continue dans l'environnement. Les agents berger ne communiquent pas entre eux, donc la réussite est basée sur le bon choix de l'action dans la liste d'actions possibles. Leur travail démontre la fiabilité de l'apprentissage par renforcement pour la création de comportements. Une des difficultés dans les systèmes de classeurs est le contrôle des états non markoviens, ce qui revient à la discrétisation de l'espace où interviennent les agents. Afin d'automatiser la conception de contrôleurs utilisant une représentation plus générale des espaces des entrées et des sorties.

Dans le but de fournir un système comportemental adaptatif, Sanza [San 01] s'est appuyé

sur un système de classeurs, αCS, pour modéliser le comportement de ses entités.

Il se base donc sur une base de règles pour définir le comportement de ses entités, la sélection de la règle, ou classeur, à appliquer s'effectue en deux phases : La première étape présélectionne les classeurs dont la partie condition correspond à l'état de l'environnement. La seconde permet de choisir parmi tous les candidats potentiels celui qui devrait fournir la meilleure réponse à l'état courant. Ainsi il a pondéré la force de chacune des règles avec sa spécificité (c'est-à-dire la précision avec laquelle la partie condition du classeur décrit son environnement) et il a ajouté un bruit on obtient une valeur représentative de la règle. Une sélection stochastique permet alors de déterminer la règle choisie. Il ne reste plus qu'à activer l'effecteur déterminé par la partie action du classeur.

En plus de la mécanique de sélection de l'action, ce modèle rajoute un élément de coordination entre entités : La communication et par conséquent l'échange de règles permet de profiter de l'expérience de plusieurs entités simultanément afin d'obtenir un système plus robuste. Il ajoute également la possibilité de créer un dialogue Entité/utilisateur au moyen d'un générateur syntaxique.

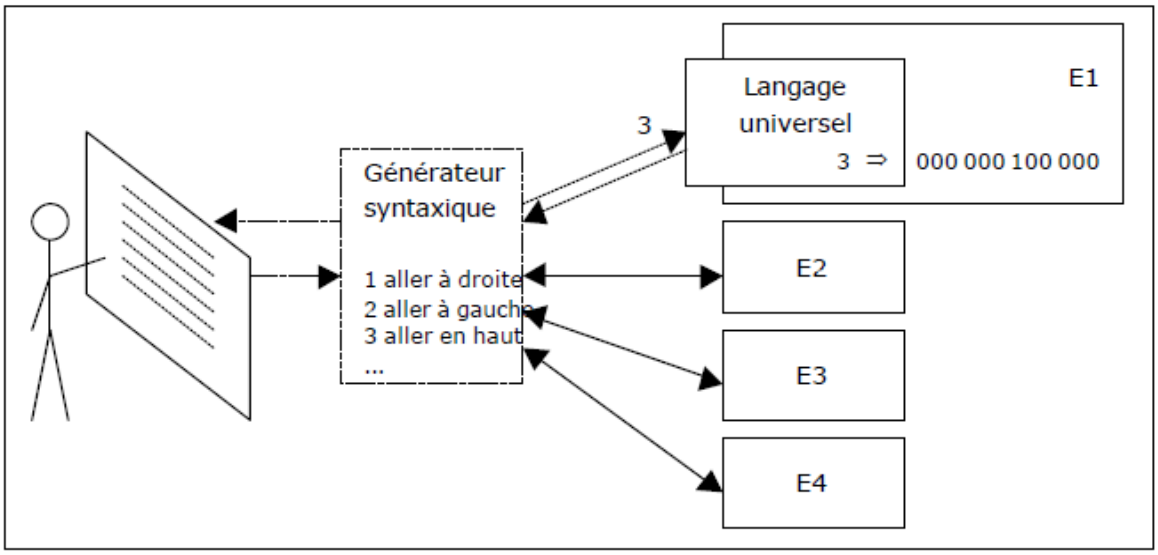


Figure 3.3 : Dialogue utilisateur/entité

L'aspect adaptatif et collectif du modèle a permis la réalisation d'une simulation de football dans laquelle chaque entité est contrôlée par ce système. Les résultats sont pertinents et accompagné d'une forte émergence d comportements collectifs.

Cependant ce système présente le désavantage de n'activer qu'un effecteur à la fois limitant ainsi les applications du modèle. De plus il ne permet pas de pouvoir intégrer d'autres méthodes de contrôle des entités dans le but de pouvoir comparer les résultats face à des situations prédéterminées.

Heguy 2003 s'appuit sur le classeur XCS [Heg03] pour faire une généralisation GCS qui permet de construire les parties condition et action de chaque classeur à partir de différents types d'entrées ou de sorties « binaires, réelles, entières et vectorielles ». Cela lui permet notamment, après avoir sélectionné arbitrairement un certain nombre de points pertinents dans l'environnement, de générer des actions de déplacement sous la forme de combinaisons de champs de forces, chaque force attirant l'agent vers un des points pertinents. Chaque agent est doté de son propre système de classeurs et agit sans communiquer avec les autres agents. Bien que le modèle permette une représentation continue de l'environnement, l'espace de simulation a été discrétisé.

Ramos [Ram07**]** a présenté un modèle comportemental pour produire une structure modulaire plus facile à utiliser. Il a donc créé une architecture hybride rajoute un facteur d'apprentissage et donc d'adaptation entre la couche réactive et la couche cognitive.

Le modèle a été évalué dans une plateforme de simulation utilisant plusieurs techniques de la vie artificielle : l'utilisation de vecteurs pour gérer la perception des entités et contrôler leur déplacement, des techniques pour la détection d'obstacles, les systèmes de classeurs pour la partie apprentissage afin de simuler des comportements plus complexes. L'émergence de comportements anticipatifs résulte de la coordination entre la couche réactive et de la couche cognitive. Le système produit des comportements individuels et collectifs en temps réel.

Tran [Hau07] a crée une architecture CREA dans le cadre de la simulation de comportements adaptative d'entités. Il a utilisé les caractéristiques de réactivité des agents pour assurer une exécution rapide des actions dans l'environnement sans passer par la planification. Cependant. L'architecture comportementale CREA a été validée dans un problème de type proie - prédateur. Les systèmes de classeurs s'appuient sur des actions *discrètes*. Ils ne sont pas et il a proposé d'étendre le XCSF à des actions continues qui sont calculées directement en fonction du message entrant. Malgré le manque de feedback d'action, leur système de classeurs XCSFCA a réussi avec succès à résoudre ces problèmes

où l'espace d'actions est continu. Le XCSFCA a été validé dans les problèmes de référence : *frog*, *double integrator* et *pendulum swing up*.

2. Architecture globale du système

Nous avons remarqué dans le chapitre1 plusieurs façons pour représenter les agents pour l'émergence de comportements autonomes dans les environnements virtuels :

Les approches cognitives exigent une phase de planification avant d'agir. Cela entraîne souvent un temps de réponse élevé ainsi qu'une représentation interne de l'environnement qui doit être mise à jour régulièrement, et ne permettent pas de produire un comportement efficace face à des situations imprévues.

Les approches réactives permettent aux agents d'agir en temps réel. L'information des capteurs est directement envoyée aux comportements de base. Les agents sélectionnent une action instantanément pour répondre à cette information sans passer par la planification d'actions pour satisfaire les objectifs à long terme. L'émergence du comportement au niveau de l'observation apparaît de la fusion des comportements de base actifs. Des stratégies de fusion sont la combinaison pondérée des comportements de base actifs par les poids adéquats ou la sélection du comportement ayant la priorité la plus élevée. Dans l'approche réactive, le calcule de comportements de base est rapide et ne requiert pas de représentation interne de l'environnement, parce qu'elle doit être mise à jour régulièrement lorsque les changements dans l'environnement apparaissent, cela ralentit le temps de réponse.

Lorsque les poids sont établis par un concepteur humain, ces agents réactifs ne peuvent s'adapter aux changements de l'environnement.

Les approches évolutionnaires issus de la vie artificielle (VA), elle sert à garantir le processus d'apprentissage, qui permet d'ajuster automatiquement tous les paramètres (poids) en réponse à des situations différentes.

Les recherches actuelles essaient de combiner ces aspects pour créer des architectures hybrides capables de coupler ces approches.

Nous allons donc proposer une architecture comportementale pour la simulation de comportements d'agents dans un environnement virtuel, elle combine deux approches l'une réactive et l'autre évolutionnaire afin de créer des agents qui sont capables d'une part de répondre instantanément aux informations reçues de l'environnement et d'autre part d'adapter leur comportement aux intentions d'autres agents lorsque les agents réalisent une tâche collective. En effet, le mécanisme de déplacement basé sur la théorie de *motor schemas* permet d'obtenir des comportements réactifs, et merci aux systèmes de classeurs pour la partie apprentissage qui permettent d'obtenir des comportements adaptatifs.

Pour cela, on va créer deux types d'agents : agent programmé qu'utilise l'approche réactive (motor schemas) avec des poids ajustés par le concepteur et un agent apprenti a la même architecture de l'agent programmé mais les poids sont générés par une approche évolutionnaire, donc l'architecture de l'agent apprenti est un assemblage de deux approches l'une réactive et l'autre évolutionnaire.

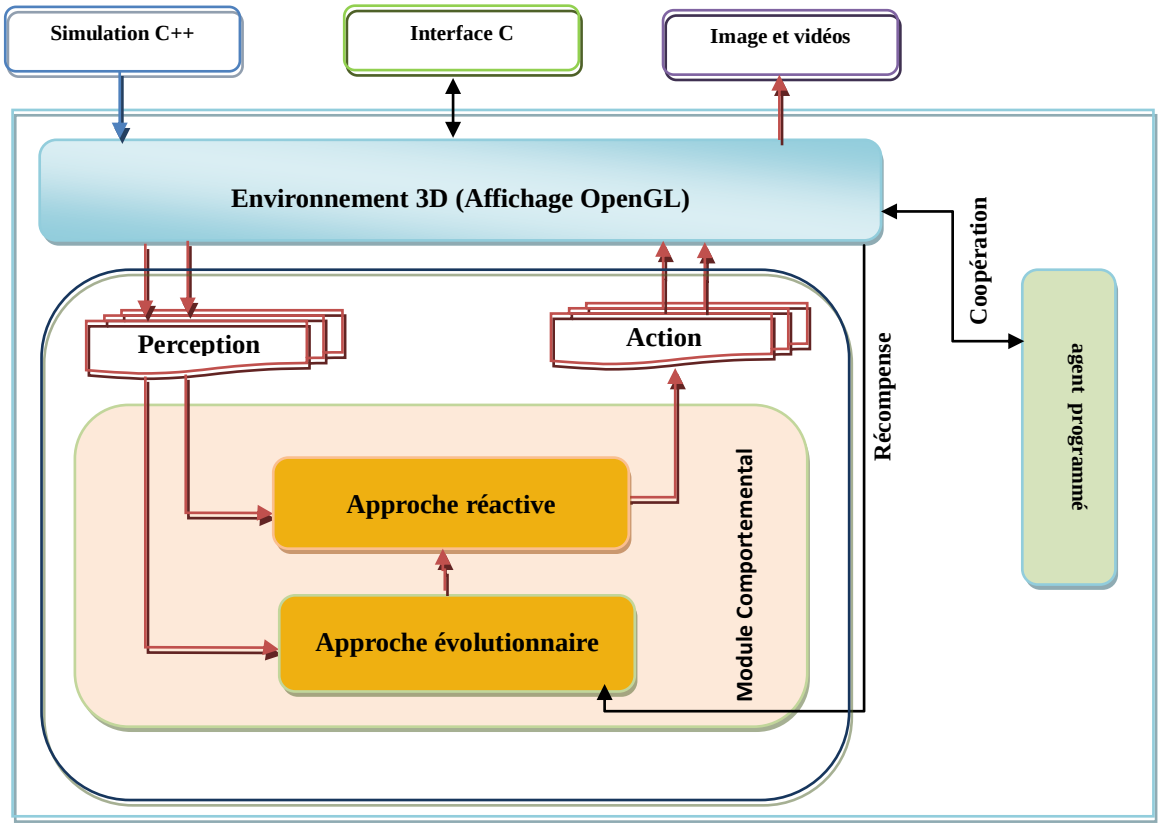


Figure 3.4 : L'architecture globale du système.

La figure ci-dessus (figure 3.4) nous illustre comment se réalise la coopération entre l'agent programmé et l'agent apprenti, pour résoudre cette problématique notre architecture est constituée des parties suivantes :

- Un module de perception (capteurs)
- Un module d'action (effecteurs)
- Un module comportemental (système de décision).

2.1. L'environnement

L'environnement représente l'ensemble des données pertinentes qui contrôlent la simulation. Les agents agissent dans leur environnement en évoluant dans le monde. L'environnement produit des récompenses pour les actions qui augmentent le degré de satisfaction chez les agents. Il stocke également l'ensemble des données physiques relatives aux entités comme leur position et leur orientation afin de pouvoir valider l'intégrité des lois qui régissent ce monde.

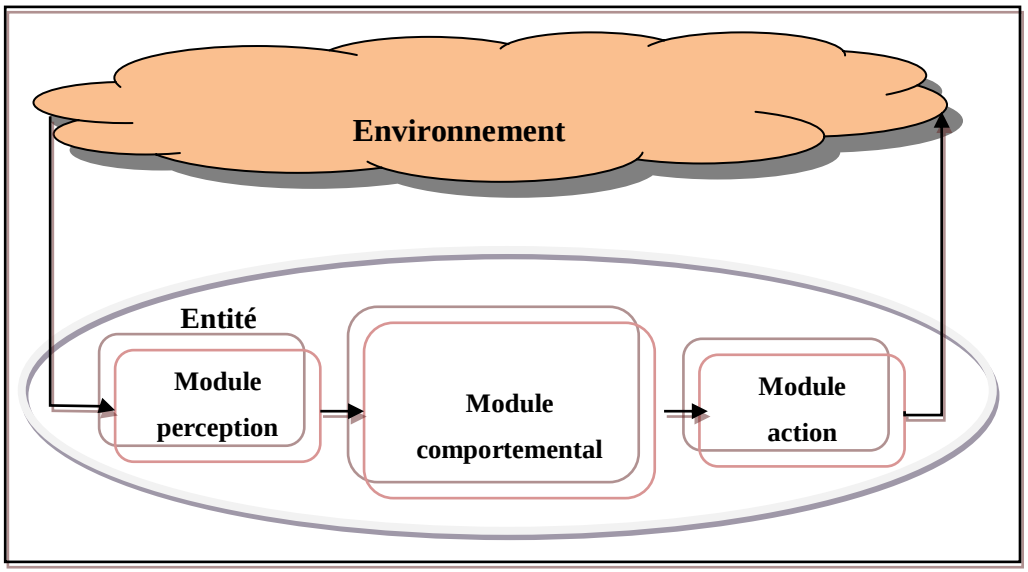


Figure 3.5: Relation environnement / entité

Terzopoulos a défini un modèle comportemental qui est repris depuis dans la plupart des simulations de comportements [Ter94]. Ce modèle s'appuie sur une représentation de la forme : *Perception - Module comportemental - Action.*

Un des points importants du rôle de l'environnement est de prendre en compte l'aspect temporel de la simulation. Contrairement au monde réel, les mondes virtuels ne travaillent pas suivant un temps continu. Il faut donc gérer des états dans l'environnement pour éviter des incohérences. De façon générale, à chaque pas de temps, tous les objets de la scène mettent à jour les sauf pour les objets statiques. Parmi ces objets statiques on retrouve les obstacles (les murs, la végétation ou plus généralement le décor de la simulation).

Enfin les agents vont être capables d'agir dans leur environnement, sans aide extérieure, grâce à la pression qu'exerce l'environnement (apprentissage) et leurs motivations.

Boucle d'animation

Faire

Incrémenter le pas de simulation
Pour toutes les entités
Perception
Pour toutes les entités et objets dynamiques
Détermination du comportement
Pour toutes les entités et objets dynamiques
Exécution des actions

Tant que la simulation n'est pas terminée

La durée d'un pas de simulation est déterminante. En effet, si le pas de temps est trop important par rapport à vitesse de déplacement d'un agent, celui-ci risque de se retrouver de l'autre coté d'un obstacle sans avoir perçu de collisions.
Il faut donc veiller à restreindre le plus possible ce pas d'animation.
En synthèse d'image, l'objectif est d'avoir au moins 25 images par seconde. Il faut donc que chaque pas de simulation ait une durée maximum de 0,04 secondes.

La représentation graphique de l'environnement et des entités animés a été réalisé grâce à la librairie graphique OpenGL.

2.1.1. Librairies graphiques.

OpenGL (Open Graphic Library)

OpenGL est une API (Application Programming Interface) multi plate-forme pour le développement d'applications gérant des images 2D et 3D. Elle a été développée en 1989

Le modèle proposé par Sillicon Graphics, puis portée sur d'autres architectures en 1993. Depuis 1992, sa spécification est surveillée par l'OpenGL Architecture Review Board (ARB).

Son interface regroupe environ 150 fonctions qui peuvent être utilisées pour afficher des scènes tridimensionnelles complexes en décrivant des objets (caméras, lampes, modèles 3D…) et les opérations que l'on peut effectuer pour les manipuler.

Elle à été implémentée sur la plupart des plates-formes informatiques actuelles (Linux et Unix,

Windows, MacOs, BeOS...) et est disponible pour de nombreux langages de programmation (C, C++, Java, Fortran, Ada...).

Du fait de son ouverture, de sa souplesse d'utilisation et de sa disponibilité sur toutes ces plates-formes, elle est utilisée par la majorité des applications scientifiques, industrielles ou artistiques 3D et certaines applications 2D vectorielles.

2.2. Le module de perception (capteurs)

La perception est une partie très importante chez les agents, il traite l'information provenant de l'environnement. La difficulté est de traiter l'information de façon correcte pour permettre aux agents une meilleure adaptation. Cependant il y a deux problèmes : le premier est de reconnaître les informations pertinentes et le deuxième est la génération de comportements cohérents par rapport à l'environnement et les objectifs globaux.

La perception est basée sur des capteurs qui envoient l'information perçue de l'environnement aux modules comportementaux.

L'information provenant de l'environnement étant volumineuse, pour la traiter efficacement il est nécessaire d'appliquer des filtres à l'entrée des capteurs de l'agent.

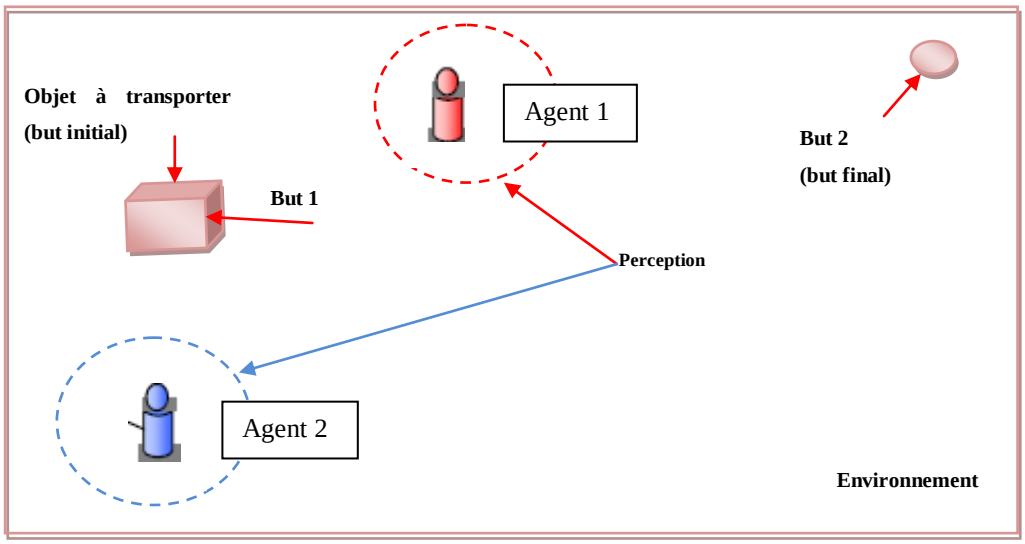


Figure 3.6 : Perception des agents dans l'environnement.

Bien que les capteurs soient dotés de filtres pour traiter l'information, il est nécessaire de stocker l'information pour que le module comportemental puisse l'utiliser. Pour cela nous avons mis en place une mémoire basée sur les systèmes perceptifs d'Arkin [Ark98]. Le but est de fournir différents sous-systèmes perceptifs spécialisés qui extraient l'information pertinente pour chacun des comportements actifs. La perception est ainsi guidée par les besoins des comportements, c'est-à-dire par les actions à réaliser. Cette approche sélective de la perception se base sur le principe que le monde peut être vu de différentes façons, en fonction notamment des intentions d'un individu et des tâches qu'il doit réaliser.

2.3. Le module action

Une action est la modification de l'environnement. Les actions doivent respecter la cohérence de l'environnement, une action ne peut pas être exécutée si celle-ci ne respecte pas les règles, par exemple se déplacer au travers d'un obstacle. Le résultat de l'action pourra néanmoins être connu par l'intermédiaire de la perception. Les éléments de transition entre module comportemental et module d'action sont des ordres contenant à la fois l'effecteur à activer et une valeur d'évaluation.

Dans notre architecture le module d'action génère des vecteurs déplacements appropriés qui guident les agents grâces aux changements des poids de l'approche motor schemas.

2.4. Le module comportemental (système décisionnelle)

La génération de comportement s'appuie souvent sur la manipulation de connaissances symboliques pour atteindre ses objectifs [DM99]. Bien que ce système puisse être très performant dans un certain contexte comme les algorithmes de pathfinding, il se trouve restreint à un faible panel de simulations.

De plus, la plupart des problèmes résolus à l'aide de plans ou de séquences d'actions à réaliser sont dépendants d'une certaine représentation de l'environnement et d'un objectif unique.

Mais cette représentation interne doit être mise à jour régulièrement lorsque les changements dans l'environnement apparaissent, cela ralentit le temps de réponse.

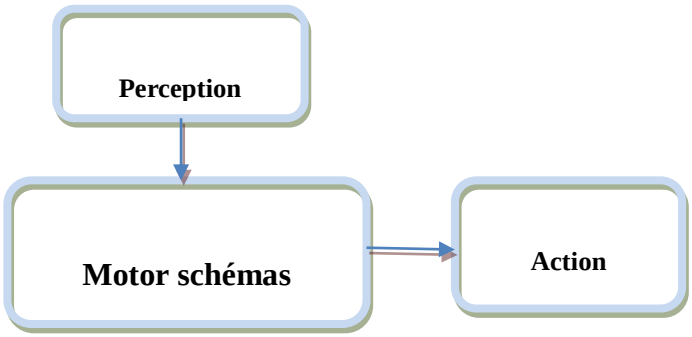


Figure 3.7 : Architecture du module comportemental de l'agent programmé

L'approche réactive propose des modèles reliant fortement la perception et l'action pour assurer une exécution rapide de cette dernière dans l'environnement sans passer par la planification.

Notre agent programmé utilise l'approche réactive qui est le *motor schema* pour garantir son déplacement [Ark98, Ark92] (voir section 1.1). Un *motor schema* correspond à un comportement de base.

L'agent programmé calcule trois *motor schemas* (i.e 3 comportements de bases) :

1. diriger -vers -cible: lui permet d'approcher de sa cible,

2. garder-formation: lui permet de prendre la meilleure position autour de l'objet,

3. déplacer-aléatoire: fait un déplacement aléatoire.

Les sorties des *motor schemas diriger-vers-cible*, *garder-formation*, *déplacer-aléatoire* sont respectivement les vecteurs f_1, f_2, f_3 qui sont ensuite additionnés et normalisés pour générer un vecteur global f_3 guidant le déplacement f d'agent. Ou bien, la combinaison pondérée des vecteurs de deux premiers schemas donne le déplacement f d'agent. Les paramètres de schemas (poids) w_1, w_2 correspondent respectivement aux *motor schemas diriger-vers-cible* et *garder-formation*.

La combinaison des *motor schemas* actifs par les poids adéquats génère des comportements émergents et complexes. Classiquement, les poids qui contrôlent cette combinaison sont donnés par le concepteur pour l'agent programmé. Cependant, la capture de la cible est réalisée par le changeant dynamiquement les valeurs des poids en fonction des situations prédéfinies.

La figure 3.8 décrit l'architecture de comportement de l'agent réactif. Tout l'art de cette approche se trouve dans le module de combinaison Σ des *motor schemas*.

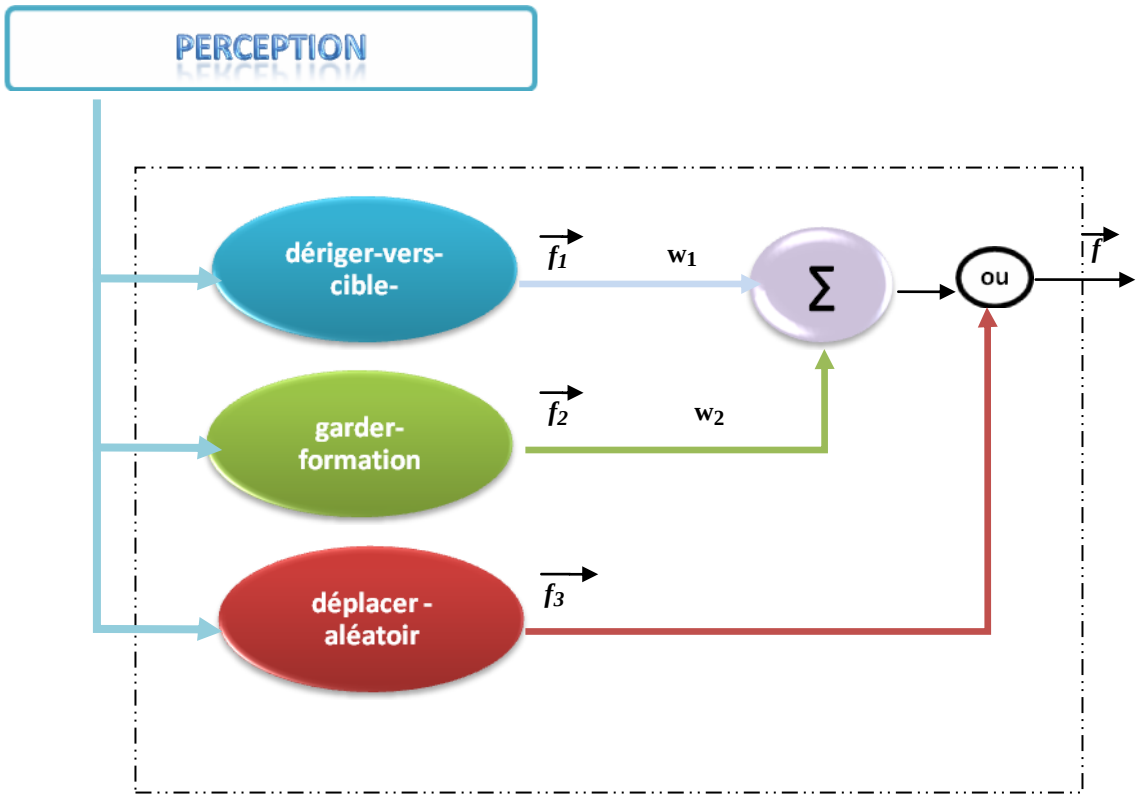


Figure 3.8 : comportements d'un agent programmé

L'agent programmé est un agent réactif ne peut pas s'adapter aux changements de l'environnement car le concepteur humain établis les poids de l'approche motor schema, pour cela nous avons intégré dans le module comportementale un modèle d'apprentissage permettant d'ajuster automatiquement tous les paramètres en repense à des situations différentes pour que ce nouveau type d'agent coopère avec l'agent programmé pour garantir la stratégie coopérative.

2.4.1. L'intégration de l'apprentissage

Agre et Arkin ont démontré les limitations des comportements réactifs [AC90, Ark98]: Un système réactif est robuste, mais manque d'adaptabilité puisqu'il ne s'appuie pas sur des capacités d'apprentissage qui lui permettraient de pouvoir étendre ses capacités.

Nous proposons d'intégrer au modèle d'agent réactif un modèle d'apprentissage pour prendre en charge le travail du concepteur et permettre d'ajuster automatiquement tous les paramètres en réponse à des situations différentes, ce type d'agent nommé *agent apprenti*.

L'utilisation d'un système d'apprentissage, surtout s'il est adaptatif et dynamique comme le sont les systèmes de classeurs, permet d'alléger le travail du concepteur en lui permettant de se focaliser uniquement sur la nature de la tâche à résoudre, les éléments à prendre en compte et les situations clés permettent de récompenser l'agent lors de son apprentissage.

L'enchaînement des différentes actions (le mécanisme de résolution de la tâche collective) sera alors déduit par l'agent lui-même en fonction de sa perception et de ses récompenses.

Dans le cadre de la génération des comportements adaptatifs d'une entité autonome, les systèmes de classeurs représentent une alternative pertinente tout d'abord le système de classeurs a été employé pour notre agent en tant qu'outil efficace pour l'adaptation, et qui mémorise des situations, fait évaluer des paires situation/action appelées règles, génère et fait évoluer une population de règles afin d'obtenir une meilleure coopération dans le groupe d'agents.

Leur structure et leur mode de fonctionnement sont proches des mécanismes des systèmes comportementaux classiques :

- **I**ls sont composés de capteurs produisant, sous la forme d'un message.
- **U**ne base de règles de production activables par les messages permet de sélectionner une action pertinente en fonction de la situation de l'agent.
- **L**'action sélectionnée est transmise à des effecteurs permettant sa réalisation.

Ensuite, leur faculté d'apprentissage, associée à leur nature évolutionniste, en fait une méthode de génération robuste capable d'explorer différentes possibilités de résolution et de réaliser des compromis si nécessaire. De plus, la génération automatique des règles de production en fonction des situations de l'environnement (par l'opérateur de *couverture*) leur permet d'adapter dynamiquement le comportement produit. Les règles produites par cette adaptation sont ensuite validées et renforcées en fonction des récompenses issues de l'environnement afin de générer les comportements désirés.

Enfin, l'utilisation par les *systèmes de classeurs*, ainsi que des opérateurs génétiques adaptés, permet de simplifier l'implémentation des problèmes à résoudre.

Cela permet de la réduire aux choix de la représentation des règles et messages et des situations générant des récompenses ou des pénalités.

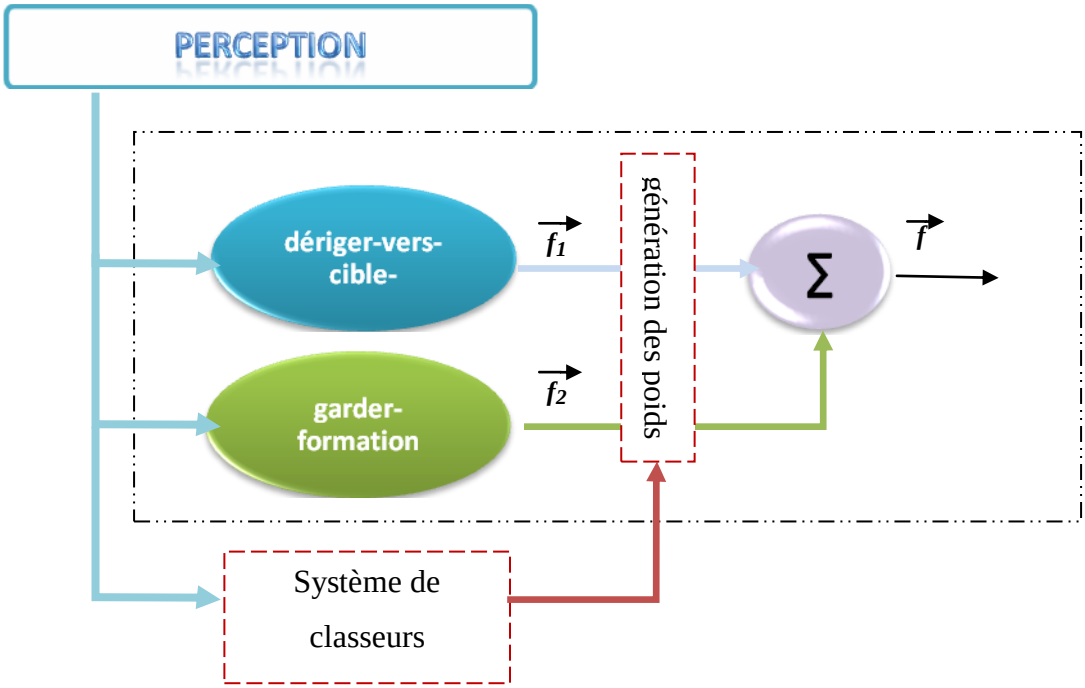


Figure 3.9 : comportement d'an agent apprenti

La figure 3.9 montre que l'agent apprenti connaît seulement deux *motor schemas diriger-vers-cible* et *garder-formation*. Il n'est pas équipé du *motor schema déplacer-aléatoire*. Ses paramètres de schemas w_1, w_2 sont ajustés par le système de classeurs et sont donc différents de ceux d'agent programmé. Pour mieux comprendre comment le YCS génère ces paramètres, il faux reconnaitre leur structure, et leur fonctionnement.

2.4.1.1. Méthodologie d'apprentissage

L'implémentation d'un problème spécifique devant être résolu par apprentissage à l'aide d'un système de classeurs de type *YCS* nécessite essentiellement la définition:

- de la forme des messages et des règles de production (les classeurs),
- de la politique d'attribution des récompenses.

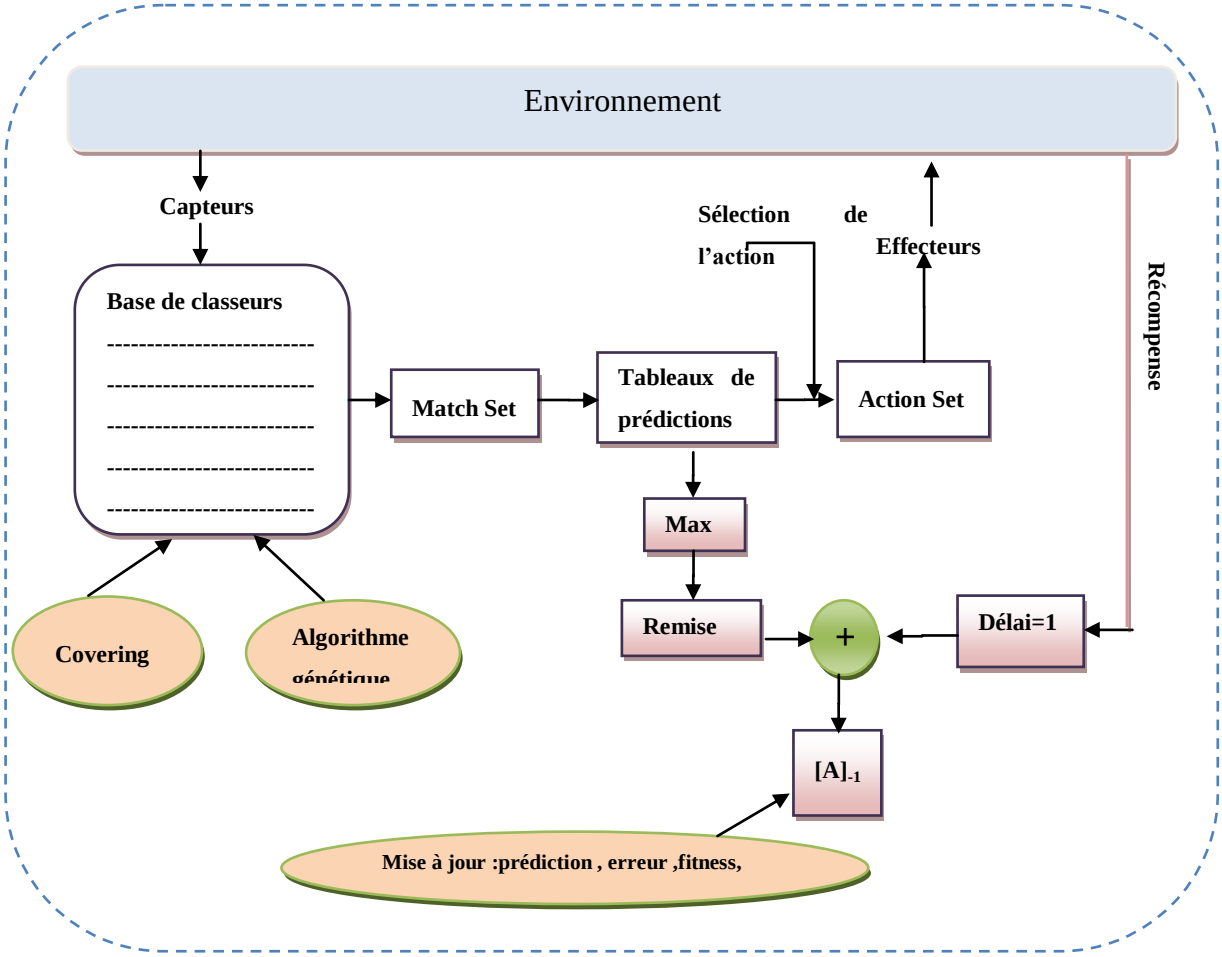


Figure 3.10: système de classeurs utilisé par l'agent apprenti.

2.4.1.2. Représentation d'une règle

Structure

Une règle (classeur) est composé de trois parties distinctes la partie condition et la partie action et les attributs :

La partie condition : est l'information provenant de l'environnement pour qu'elle soit interprétée par son système de classeurs, cette information est convertie en un message encodant les angles entre les agents.

Le nombre de bits de la partie "condition" (NbBitsC) correspond au nombre de capteurs (NbCap) : NbBitsC = NbCap.

Un système de classeurs scrute l'environnement grâce à son propre système sensoriel : les capteurs. Dans notre modèle, nous avons donc imposé des capteurs sous forme binaire. Ils constituent l'interface d'entrée entre l'environnement et le système de classeurs. Ainsi, la perception du monde extérieur est traduite en une série de valeurs codées respectivement par "1" et "0". A chaque acquisition de données, ces capteurs vont être utilisés simultanément par le module de codage pour composer le *message entrant*. Les bits sont concaténés les uns à la suite des autres pour former une chaîne binaire. La figure 3.11 présente un exemple, nous divisons le cercle en 8 secteurs (numérotés de 0 à 7) qui utilise 8 capteurs dans la construction du *message entrant*, le centre du cercle est la position de notre cible et la présence d'un symbole 'a' dans un secteur indique un agent, avec le code 1 ; les secteurs vides ont le code 0. Un message encodé a la forme : (*i2i1i0*) (*a0a1a2a3a4a5a6a7*). Nous employons 3 bits *i2i1i0* pour indiquer dans quel secteur l'apprenti est (secteur n°3 dans la figure 3.11). Le bit *a0* est toujours le code du premier secteur (secteur n°0 dans la figure 3.11) et les autres (*a1-> a7*) correspondent aux codes des secteurs n°1 ->7, on note A.A (Agent Apprenti)et A.P(Agent Programmé).

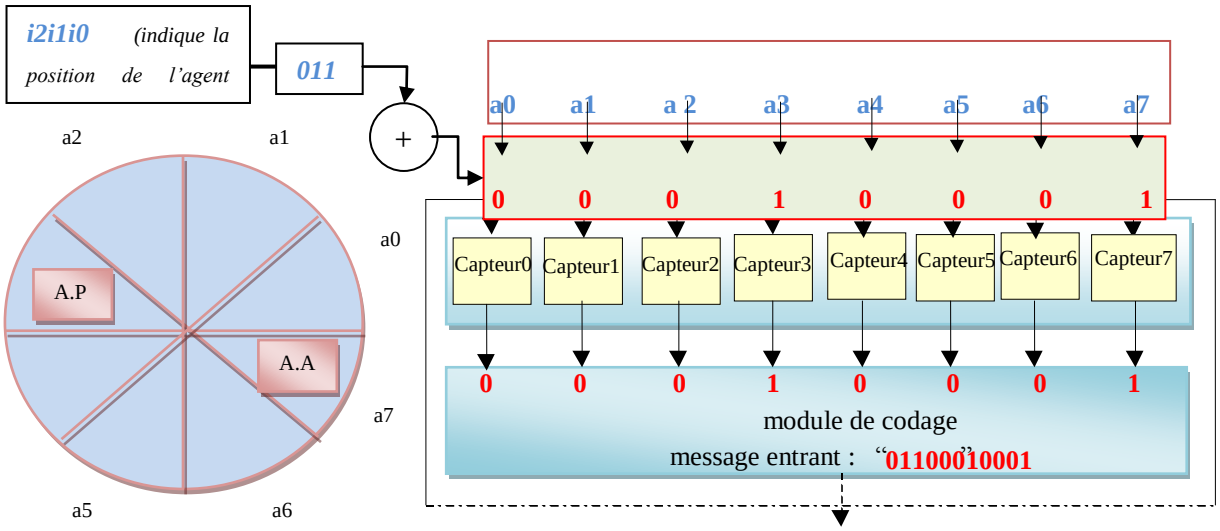


Figure 3.11 : Construction du message entrant

partie action : Le nombre de bits de la partie "action" (NbBitsA) correspond au nombre d'effecteurs (NbEff) : NbBitsA = NbEff.

Nous encodons la partie action du système de classeurs sous la forme : (*a3a2*)(*a1a0*) dont les 2 bits *a3a2* et les 2 bits *a1a0* indiquent respectivement les poids w*1*, w*2*. L'agent

apprenti ne possède pas des actions prédéfinies dans la modification des poids, mais son système de classeurs cherchera automatiquement des actions appropriées pour le faire. Nous avons utilisé l'approche *motor schema* pour aider l'agent à déterminer son déplacement. L'avantage de cette approche est de produire des comportements de haut niveau à partir d'un ensemble de comportements de base (*schemas*) en déterminant les poids de chaque schema dans le comportement final. Donc, la capacité d'adaptation de l'apprenti est améliorée en changeant automatiquement les poids en fonction des situations à l'aide du système de classeurs.

Le nombre de capteur et d'effecteurs est déterminé par le programmeur lors de l'intégration d'un système de classeurs dans une application. Pour garantir l'efficacité et la rapidité du système, il est préférable d'utiliser un nombre minimum de capteurs et d'effecteurs puisque leur nombre a une incidence directe sur la longueur d'une règle. Le choix de capteurs et d'effecteurs pertinents constitue donc la principale difficulté et contribue à la qualité du résultat final.

Attributs

Chaque classeur a quatre attributs principaux.

Une prédiction p_j estimant une prédiction à venir est utilisée dans la sélection de l'action.

Une erreur de prédiction ε_j indique une estimation de l'erreur de la prédiction du classeur.

Une fitness F_j définissant par l'inverse de l'erreur de prédiction est utilisée dans la sélection de classeur lors de l'application d'un AG.

Une taille de niches α_j estime la taille moyenne de niches (des ensembles A) auxquelles le classeur a participé.

Les attributs de chaque règle sont initialisés aléatoirement.

2.4.1.3. Initialisation de la base de règles

Dans notre système, le nombre de classeurs (NbClas) reste fixe tout au long d'une simulation. L'initialisation de la base de règles peut s'effectuer de deux manières différentes :

à partir d'un fichier de données contenant des règles sous forme binaire ou aléatoirement.

La première méthode est très intéressante car elle permet d'utiliser des règles créées lors de simulations précédentes. En supposant que ces règles soient efficaces, la

confrontation dans un autre contexte permet alors de tester non pas l'apprentissage mais l'adaptation à de nouveaux paramètres.

A l'opposé, l'initialisation aléatoire induit un apprentissage sans connaissance de base. Dans ce cas, le système va utiliser toutes ses potentialités pour faire émerger les comportements.

Et dans notre simulation on utilise la méthode la plus efficace pour émerger les comportements plus rapide à partir d'un fichier de données contenant les règles sous forme binaire et les attributs en réel.

2.4.1.4. Sélection d'un classeur

Le module de sélection permet de choisir les règles qui vont être activées parmi le nombre de classeurs pour construire l'ensemble [M] à partir de ceux qui satisfont le message provenant de l'environnement (condition unique), donc cette étape permet de réduire considérablement le nombre de candidats. Cette phase se décompose en deux parties : une présélection qui utilise l'adéquation de la règle avec le message entrant Ainsi, à partir d'un critère très simple, le premier niveau permet de réduire considérablement le nombre de candidats pour la deuxième étape.

Présélection

La présélection agit comme un filtre en éliminant une grande partie des règles présentes dans la base. A ce niveau, le module vérifie la concordance de la partie condition de chaque règle avec le message entrant en comparant les bits occupant la même position dans les deux chaînes pour former une base [M].

Sélection

A partir de l'ensemble [M] le système doit choisir "au mieux" l'action qui permettra la meilleure récompense. Pour cela, on constitue un tableau de prédictions (*prediction array*) contenant la prédiction $p_a(t)$ pour chaque action *a* présente dans les parties action des classeurs de [M]. La valeur de la prédiction $p_a(t)$ d'une action *a* est la moyenne pondérée par la *fitness* des *prédictions* des classeurs *i* de [M] qui déclenchent *a*:

$$p_a(t) = \frac{\sum_{i \in [M]_a} P_i(t) \times F_i(t)}{\sum_{i \in [M]_a} F_i(t)}$$

L'action ayant la prédiction la plus élevée dans un tableau de prédictions est sélectionnée de façon déterministe.

2.4.1.5. Système de rétribution

Nous avons utilisé notre système de classeurs afin de favoriser la coopération du comportement de nos agents. De manière à conserver la structure de base de notre modèle, nous avons choisi d'orienter la *coopération* sur le système de rétribution.

La récompense locale *LR* est appliquée à l'agent apprenti pour renforcer ses actions alors que celle qui est globale *GR* est appliquée à l'apprenti pour favoriser la coopération. A la fin de chaque itération, la récompense globale est $+GR$ si la cible est atteinte et $-GR$ dans le cas contraire. L'agent apprenti compare sa décision (f_{system} dans (**3.1**)) avec des signaux de coordination [Dej 99] f_{signal} reçus de son partenaire (l'agent programmé). Nous définissons un signal comme un vecteur qui guide le déplacement d'un agent. La différence entre un signal de coordination reçu et la décision de l'agent apprenti est interprétée comme la récompense locale pour mettre à jour les classeurs dans l'ensemble *A* (*Action Set*) déclenchant la dernière action. En outre, nous prenons en compte le lien direct entre deux actions exécutées consécutives. Donc, les classeurs dans κ ensembles précédents A_{-k}, …, A_{-2}, A_{-1} partagent la même récompense avec ceux dans A. L'algorithme IBB [Wil94] est utilisé pour faire ce renforcement.

$$LR = \cos(\vec{v}_{signal}, \vec{v}_{system}) \qquad 3.1$$

L'apprenti reçoit un signal de son voisin. Comme l'agent programmé utilise le même algorithme pour calculer des signaux à envoyer, l'agent apprenti prend en compte ce signal afin de la détermination de sa récompense. Les paramètres (erreur de prédiction, taille de niches et prédiction) de chaque classeur dans l'ensemble *A* sont mis à jour par la règle *delta rule* [WH60] avec le taux d'apprentissage β.

($0<\beta\leq 1$) comme dans le XCS sauf pour la fitness de classeur.

$$F_j = \frac{1}{1+\varepsilon_j}$$

2.4.1.6. Phases d'apprentissage et d'évolution

Nous distinguons deux phases dans l'élaboration des règles effectivement utilisées par le système.

Phase d'apprentissage : correspond à la recherche d'un classeur lorsque le système est confronté à un nouveau message entrant. Cette phase n'apparaît pas obligatoirement en début de simulation car l'environnement peut évoluer et générer de nouveaux contextes.

Phase d'évolution : Elle entraîne l'évolution d'une règle, dans le cas où de mauvaises rétributions l'amènent à être modifiée par l'algorithme génétique ou à être éliminée pour donner naissance à un nouveau classeur satisfaisant le même message entrant.

2.4.1.7. L'algorithme génétique

L'algorithme génétique (noté AG) est utilisé comme méthode d'optimisation pour l'exploration du nombre infiniment grand des combinaisons possibles de classeurs. Les opérateurs génétiques classiques de type croisement et mutation sont utilisés pour créer de nouvelles règles à partir des éléments de la population courante. Dans un cycle d'appel au système de classeurs, cette étape constitue l'unique moyen de renouvellement la base de règles. L'AG représente une part très importante dans le fonctionnement du système.

La représentation binaire des classeurs facilite grandement le codage des opérateurs génétiques. Malgré tout, l'AG reste très gourmand en temps de calcul. Par conséquent, l'accent a été mis sur les techniques d'optimisation du temps d'exécution, notamment au niveau de la sélection et, plus globalement à la fréquence d'appel à l'AG.

La sélection

Pour déterminer quelles règles sont plus enclines à obtenir les meilleurs résultats, une sélection est opérée. Ce processus est analogue à un processus de sélection naturelle, les règles les plus adaptées gagnent la compétition de la reproduction tandis que les moins adaptés meurent avant la reproduction, ce qui améliore globalement l'adaptation.
Le nombre de règle est fixé par le programmeur et il reste constant pendant toute la durée d'utilisation du système de classeurs.
La méthode de sélection que nous avons choisie est la méthode de *la roue de la fortune* (appelée aussi roulette pipée). Celle-ci permet d'introduire un peu de hasard en permettant la sélection des règles totalement différentes.

Le croisement

La procédure de croisement peut être appliquée de plusieurs manières. La version la plus simple est le croisement en un seul point (single-point) pour laquelle un seul point de croisement est choisi, on recombine des parties conditions des classeurs vers des règles potentiellement plus performantes, comme illustré dans figure ci-dessous.

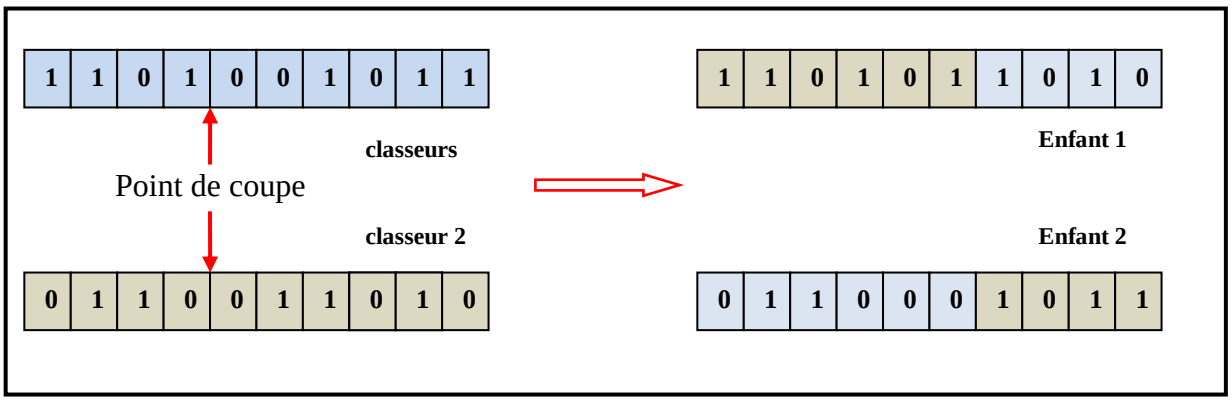


Figure 3.12 : croisement uni-point

La mutation

La mutation agit sur un bit du classeur à un point de mutation. Elle change de 0 à 1 ou l'inverse de 1 à 0 (figure 3.13). Le taux de mutation est faible et reste constante pendant l'évolution de l'AG. Le rôle de la mutation est secondaire par rapport au croisement mais elle est indispensable pour produire de nouveaux classeurs pour explorer toutes les situations possibles.

.

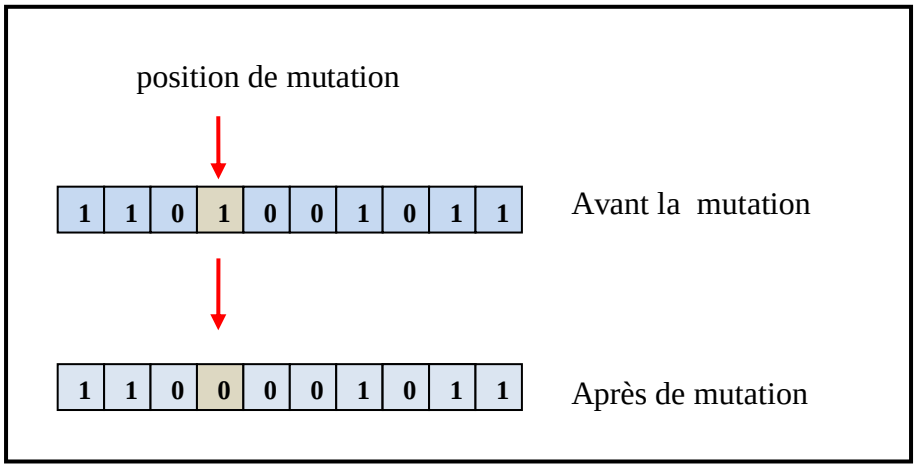


Figure 3.13 : Opérateur de mutation

2.4.1.8. Fréquence d'appel à l'algorithme génétique

L'analyse détaillée du fonctionnement d'un système de classeurs a fait apparaître les points suivants. L'algorithme génétique doit être très actif pendant la ***phase d'évolution***, puisque le système a besoin d'un maximum de diversité pour trouver rapidement une règle efficace. A l'opposé, la ***phase d'apprentissage*** ne nécessite pas d'autant de renouvellement. On utilise les algorithmes génétiques avec la même fréquence d'appel.

Ainsi, la fréquence d'appel à l'algorithme génétique doit être corrélée avec l'activité du système. Cette mise en correspondance est effectuée en créant un lien avec le module de sélection et le système de rétribution. L'adaptation de la fréquence se fait donc à ce niveau.

Lorsque le module de sélection déclenche automatiquement un appel dans le cas où le système n'aurait pas trouvé de candidats pour la présélection lorsque le covering n'est pas utilisé. Cette situation se produit particulièrement lorsque le nombre de règles est relativement faible.

2.4.1.9. Le covering (Recouvrement)

L'opérateur de recouvrement, permettant la création de nouveaux classeurs en fonction du message entrant, est activé lorsqu'aucun classeur ne correspond au message. Or, si le classeur ne produit pas les résultats espérés, un autre appel au covering va être nécessaire. Ainsi, nous avons jugé nécessaire de générer plusieurs règles lors de l'utilisation de cet opérateur.

Alors la création d'un (ou plusieurs) classeur (s) dont la condition correspond au message entrant et contient un nombre aléatoire de symboles '#'. L'action du nouveau classeur est quant à elle choisit aléatoirement. Les attributs du nouveau classeur sont initialisés de façons aléatoires, et il est introduit dans [P] selon le même principe que pour les enfants issus de l'application de l'algorithme génétique. Une fois l'opérateur de recouvrement appliqué, le système reprend son cycle avec la constitution de l'ensemble [M].

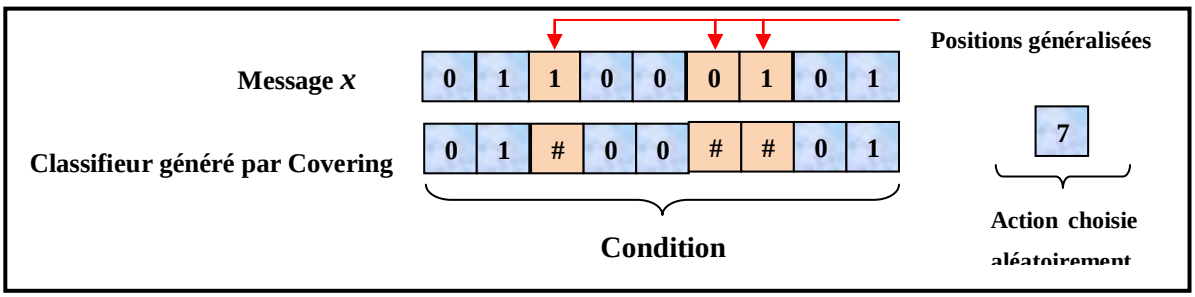


Figure 3.14 : Un classeur généré par l'opérateur Covering.

3. Récapitulatif

Tous au long de ce chapitre, nous avons proposé une architecture hybride pour générer des comportements autonomes des agents dans un environnement virtuel, combine des approches issues de domaine de la vie artificielle. Ainsi, nous avons, dans un premier temps, choisi des agents 3D dont la structure est préfixée et qui sont simulés grâce à l'exploitation de la librairie OPENGL.

Nous avons, par la suite, exploité un système de classeurs intégré dans notre architecture afin de générer des poids qui seront utilisées par l'approche *motor schemas* afin d'alléger le travail du concepteur (les poids sont initialisés par le concepteur le cas de l'agent programmé), et enfin créer un nouveau type d'agent nommé agent apprenti.

En effet, les systèmes de classeurs après avoir effectué leur évolution grâce à l'appel de façon périodique d'un algorithme génétique et après des centaines de générations, permet, à l'agent, d'apprendre de réaliser la coopération, la coordination des ses actions et de s'adapter avec son partenaire (l'agent programmé) et l'environnement.

4. Conclusion

Tous au long de ce chapitre, nous avons proposé une stratégie coopérative d'un groupe d'agents, (programmé et apprenti), afin d'atteindre leurs cibles. L'apprenti n'a aucune connaissance a priori et adapte ses actions en fonction de celles de ses partenaires afin de contribuer à la résolution de la tâche coopérative. Pour cela, l'agent apprenti est doté du système de classeurs qui génère les actions appropriées et modifie les poids de l'approche réactive (*motor schemas)*. L'ensemble d'actions possibles sont automatiquement déterminé pendant l'apprentissage.

Alors, l'aspect comportemental propose une architecture qui permet de combiner deux caractéristiques : une approche réactive basé sur la théorie de *motor schemas* permettant de fournir une architecture robuste et favorisante l'émergence de comportements collectifs et une approche d'apprentissage pour la génération des comportements adaptatifs, l'approche issues de la vie artificielle (le système de classeurs) intégré dans notre architecture a permis de mettre en évidence la capacité d'adaptation et la réactivité de notre agent apprenti.

Chapitre IV :
Implémentation et Résultats

Chapitre4

Implémentation et Résultats

"Il ne savait pas que c'était impossible, alors ils l'ont fait" Mark Twain

Ce chapitre a pour but de valider expérimentalement, à travers une simulation pour transporter un objet par une coopération de deux agents; un agent programmé et un autre apprenti. Les différentes notions théoriques abordées tous au long du chapitre précédent et à mettre en œuvre les divers détails tels qu'ils sont aboutis. Pour ce faire, nous allons adopter un langage de programmation que l'on voit commode pour notre type de système.

Nous verrons, dans la section 4 l'environnement de simulation, les agents qui y évoluent, leurs comportements et les différentes techniques implémentées et utilisées afin de les contrôler.

Nous mettons, également, en exergue dans ce chapitre les résultats de notre expérimentation que nous avons réalisée

Le chapitre est clôturé par un bilan discutant les capacités de notre architecture et son potentiel pour répondre aux objectifs envisagés.

1. Langage de programmation

Le langage de programmation est l'un des outils qui permettent au système d'être mis-en place. C'est pourquoi le choix du langage peut influencer sur la réalisation du système d'où

mais surtout pour sa compatibilité et ses capacités d'interfaçage avec le moteur physique « ODE Techniquement, le modèle permettant de faire évoluer un agent qui est réalisé par le biais d'une application implémentée dans le langage de programmation C++.

Nous avons choisi l'environnement de programmation Visuel C++ 2008 version9 pour, d'abord la qualité offertes par le C++, telle que la programmation orientée objet et pour assurer l'aspect graphique utilisant la bibliothèque OPENGL qui offre la possibilité de créer des objets graphiques (une scène, des agents virtuels, des objets,… .).

Pour montrer notre travail en reflétant le fonctionnement des YCS sur cet environnement graphique.

2. L'architecture globale de l'implémentation

Le schéma ci-dessous montre les relations entre les différentes techniques faisant partie du système global implémenté afin d'accomplir le rôle recherché dans ce travail, ce rôle étant la réalisation d'une stratégie coopérative des différents agents pour transporter un objet depuis un endroit vers un autre dans un environnement virtuel.

Comme nous pouvons le remarquer dans le schéma, le système de contrôle de l'agent apprenti réside dans la capacité de coopérer avec l'agent programmé, alors ce système est un assemblage de deux approches l'une évolutionnaire issus de domaine de la vie artificielle et l'autre réactive utilise la théorie de *motor schemas,* cette dernière est utilisée pour aider les agents de déterminer leurs déplacements.

Donc, le but de ce couplage est de générer des poids (paramètres de schemas) par le système de classeurs qui sont utilisés par le *motor schemas* pour garantir la stratégie coopérative.

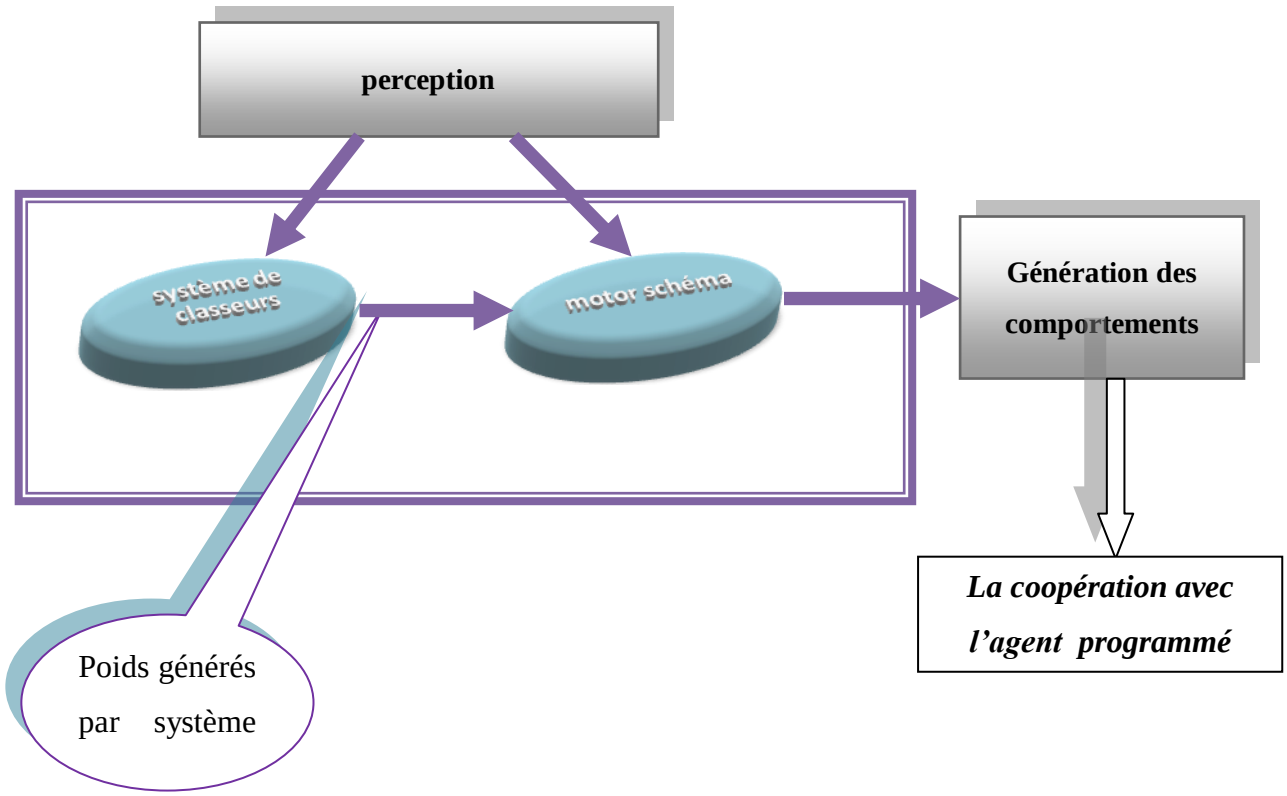


Figure 4.1: l'architecture globale du système implémenté

Nous remarquons, ainsi, que le système de contrôle de l'agent apprenti est le module principal qui nous garantit les déplacements et enfin de réalisé son objectif associé sans intervention de l'extérieur dans son environnement.

Après avoir simulé notre agent dans la classe « *Agents* », des informations sont prises en considération pour coder la perception afin de faire évoluer la base de classeurs par l'intermédiaire d'un système de classeurs qui est lui aussi implémenté dans une classe qui est la classe *«Classifier Systemes»*, qui nous permet de générer les actions(paramètres de l'approche motor schemas) recherchées.

De cette manière, notre application est implémentée grâce à un ensemble de classes, qui font appels aux différentes fonctions telles que la fonction de la dynamique des agents et celle du système de classeurs.

3. Les algorithmes utilisés

L'application réalisée dans le cadre de notre projet, est basée sur deux concepts très importants. Le premier étant la simulation graphique tridimensionnelle de nos entités dynamiques et de l'environnement (i.e. utilisation bibliothèque OpenGl) alors que le second concept est celui de l'utilisation de deux approches l'une évolutionnaire faisant partie du domaine de la vie artificielle (*système de classeurs*) et l'autre réactive (*motor schéma*).

Le système de classeurs, que nous avons intégré dans cette application, est un système de classeurs de type *YCS* qui fonctionne de pair avec le *motor schemas*, les actions générées par le système de classeurs représentent les paramètres de schemas (poids) .

Et comme mentionné ci-dessus le système de classeurs, et la dynamique des agents sont des fonctions exploitées par les fonctions 1 et 2 respectivement (voir fonction 1 et fonction 2).

3.1. Dynamique d'un système de classeurs

Les différents mécanismes impliqués dans la dynamique d'un système de classeurs (YCS), s'enchainent cycliquement selon l'ordre : Perception /Comparaison /Sélection /Action. Ce cycle est répété et peut être soumis à un critère de terminaison (temps maximal de résolution du problème).

Fonction1 : Dynamique d'un système de classeurs

```
Void : classifier systems
{
Int t=0 ; //COMPTEUR D'ITERATIONS
InitClassifierPopulation (P(t)) ; //POPULATION DE CLASSEURS INITIAL
While not do do   //CRITERE DE TERMINAISON (TEMPS...)
{
Ie(t)=ReadDedectors(t) //PERCEPTION : CODAGE DE LA PERCEPTION
M(t)=matchClassifiers (Ie(t),P(t),Comparaison)  //COMPTEUR D'ITERATION
If (M.size<criterion0 || criterion1)then
   M(t)=cover(Ie(t),P(t),Covering)
   A(t)=SelectClassifeirs (M(t), Selection) Io(t)=SendEffectors (A(t))
```

```
R=receivePayoff(t) ;
P(t)=distrbuteCredit(r,P(t),P(t-1),Rétribution) ;
If (criterion2)then           //GENERATION(AG):EVENTUELLEMENT (SELON T) UN
P(t+1) :=reviseRules (P(t), A_G)   ALGORITHME GENETIQUE EST UTILUSE SUR [P]
 }
```

3.2. Fonction d'animation d'agents

Fonction2: La fonction d'animation d'agents

```
void Environment::EvolveAgents()
{
int bObjectsInside=0;  //booléen pour tester si l'objet est dans la scène?
   if (fonction de test si l'objet dans la scène)
     bObjectsInside=1;
//DEUX BOOLEENS L'UN POUR TESTER SI LES AGENTS SONT DANS LA SCENE ET
   L'AUTRE COMPTER LE NOMBRE D'AGENTS
  int count_group=0, bAgentsInside=0;
  for( int i=0; i<nombre d'agents; i++) // NOMBRE D'AGENTS =2
  {
    If (fonction de test si les agents dans la scène)
       bAgentsInside++;
    float dist=fonction de calcul la distance entre l'objet et les agents
    if(dist<=CAPTURE_GROUP)
       count_group++;
  }
   int bObjectsCaptured=0;
   if (count_group==number d'agents)
        bObjectsCaptured=1;
   if ( bObjectsInside && bAgentsInside==1 && !bObjectsCaptured)
      {
         Fonction de perception des agents ;
         Fonction d'animation d'agents ;
         Fonction d'action d'agents ;
```

```
          Fonction de renforcement de l'agent apprenti ;
          Fonction d'algorithme génétique
          }
      else
            fonction de renforcement globale appliquée à l'agent apprenti
            pour favoriser la coopération entre les deux agents.
}
```

4. Mise en œuvre de la simulation

Les expérimentations reposent sur une simulation pour transporter un objet par des agents, un agent réactif munit d'une approche réactive (motor schemas) qui coopère avec un agent munit d'une combinaison de deux modèles (système de classeur couplé avec le motor schémas).

La simulation met en scène un ensemble des agents virtuels évoluant dans un environnement en 3D.

L'implémentation de l'environnement de simulation permet de créer différentes mises en scène. Le schéma en figure 4.2 présente grossièrement son organisation.
La simulation (*Simulation*) se déroule dans un environnement (*Environnement*). Elle accueillir un nombre illimité d'agents. Touts les agents de la simulation héritent de la classe (*Agent*).

L'environnement contient de deux types d'agents et un *objet statique* désirants de transporter. Chacun possède une position et une orientation relatives à l'environnement.
Les déplacements sont gérés de manière autonome mais la technique utilisée pour modifier la position et l'orientation (ie : pour réaliser la coopération entre les deux agents) entraîne la distinction de deux classes :

Le comportement de l'agent apprenti est évidemment géré par un contrôleur, qui lui est propre, qui sert à généré des paramètres par un système de classeurs, et possède aussi un certain nombre de capteurs et d'effecteurs.

L'agent programmé possède des comportements programmés (schemas) générés grâce à des poids (paramètres de motor schemas) initialisés par le concepteur.

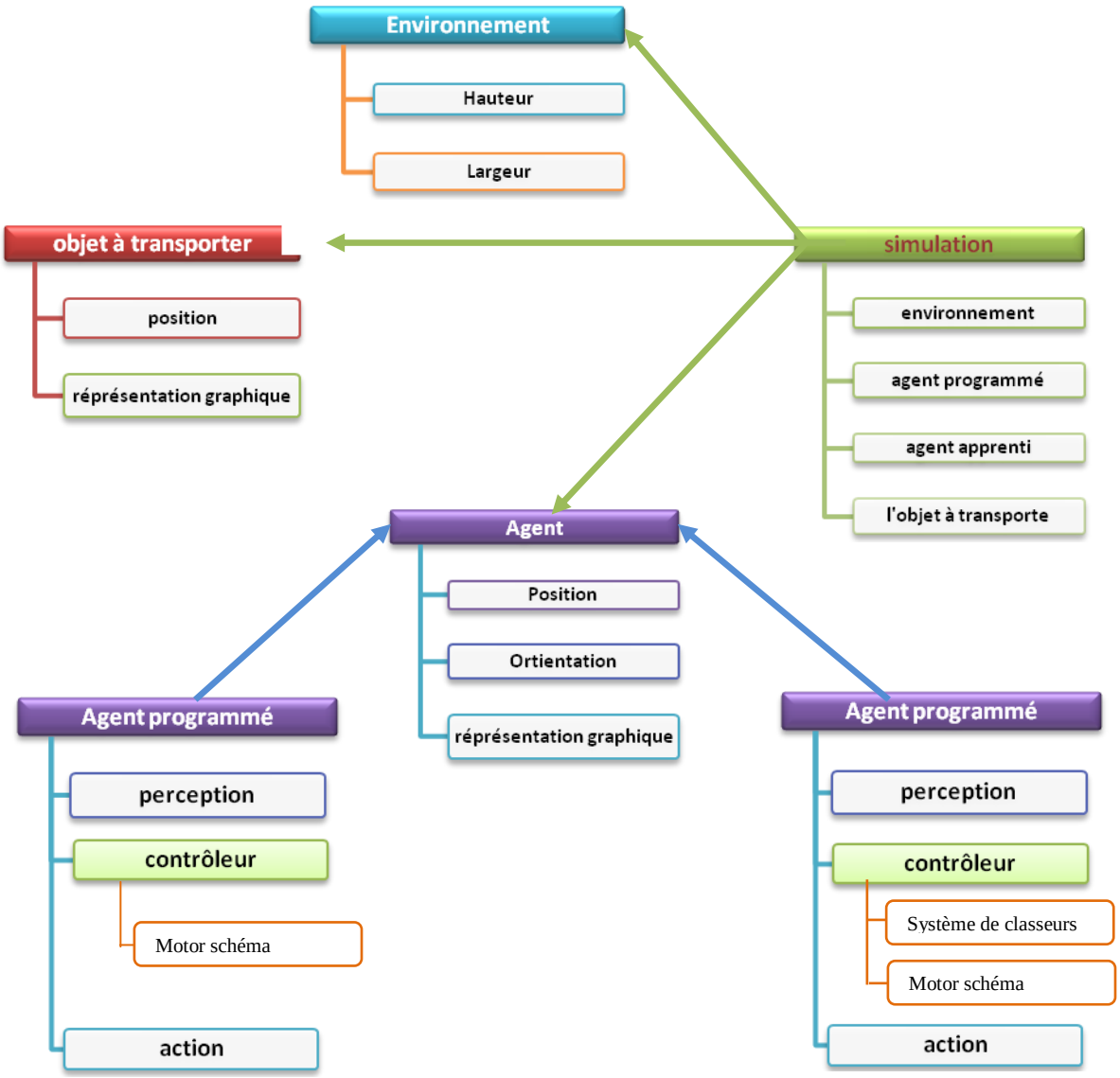


Figure 4.2 : Schéma global de l'implémentation de la simulation. Les flèches bleues représentent les relations d'héritage.

4.1. L'environnement virtuel

On définir Un environnement virtuel comme *« un environnement généré par ordinateur, composé d'objets, dans lequel des agents autonomes évoluent ».*

Alors, l'environnement virtuel de notre simulation est un environnement simple car il est plat et ne possède aucune courbure. Il s'agit donc d'une grande facette plate (i.e le sol) et quelques mures (i.e. des cubes), en plus nous avons à l'intérieur de cet environnement une sphère qui est l'objet à transporter et enfin les deux cubes représentant les agents virtuels.

4.2. Les agents virtuels

Du moment que l'environnement simulé possède les caractéristiques géométriques nécessaires à la procédure de simulation, les agents artificielles devant s'y mouvoir doit les posséder également.

Les agents simulés sont les éléments essentiels dans notre simulation, ils représentent les agents virtuels. Ces agents comme mentionné dans le chapitre précédent, possèdent des comportements pour réaliser leurs objectifs demandés. Alors, la complexité de la simulation provient de l'interaction entre ces agents.

Fondamentalement, l'environnement compte deux classes d'agents : l'agent apprenti et l'agent programmé, touts se déplacent grâce à des translations et des rotations figure (4.3).

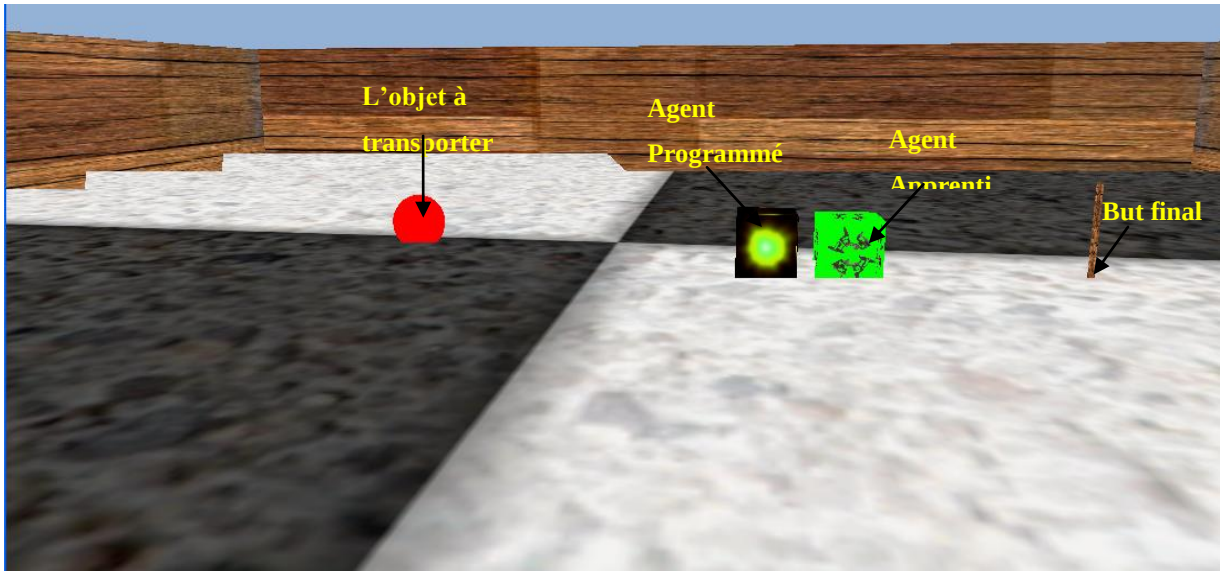


Figure 4.3 : La simulation met en scène divers entités : l'objet à transporter, l'agent apprenti et l'agent programmé

4.2.1. L'agent programmé

L'agent programmé se comporte à l'aide de la théorie *de motor schemas* qui fonctionne grâces à des poids qui sont initialisés par le concepteur. L'objectif de cet agent est de coopérer avec l'agent apprenti pour transporter un objet de sa position initiale vers le but final définit.

4.2.2. L'agent apprenti

L'agent apprenti est doté d'un système de classeurs qui lui permettre d'établir et de construire ses propres paramètres par émergence afin de garantir la coopération, cette dernière leur permet de mieux accomplir la tâche collective dans un environnement 3D**.** Ici, le système de classeurs va devoir utiliser toutes leurs ressources et capacités pour résoudre ce genre de problème. L'enchaînement d'apprentissage et d'évolution simultanés favorise le dynamisme et le renouvellement des comportements jusqu'à la fin de la simulation. De ce fait, une convergence de l'ensemble des classeurs semble improbable dans cet environnement.

De plus, la base de règles de systèmes de classeurs est initialisée à partir d'un fichier de format text. Nous avons utilisé le principe d'apprentissage par renforcement opérant pendant la simulation. L'apprentissage devra donc être très rapide et robuste face aux situations changeantes de cet environnement interactif.

L'interface entre l'architecture comportementale et l'environnement est définie par les moyens de perception (capteurs) et l'action (effecteurs).

4.2.2.1. Interface d'entrée-Sortie

Les Capteurs

Le système sensoriel constitue une partie délicate car il est à l'origine du cycle d'un système de classeurs. En effet, la sélection, l'action et la rétribution dépendent de l'information perçue Une attention particulière et de nombreux tests ont été nécessaires pour choisir les capteurs les plus intéressants.

L'agent apprenti perçoit la même information que l'agent programmé. Cependant il n'a aucune connaissance de la signification de cette information. Ainsi, il l'interprète comme des signaux pour évaluer ses actions et tenter de maximiser ces évaluations, figure (4.4).

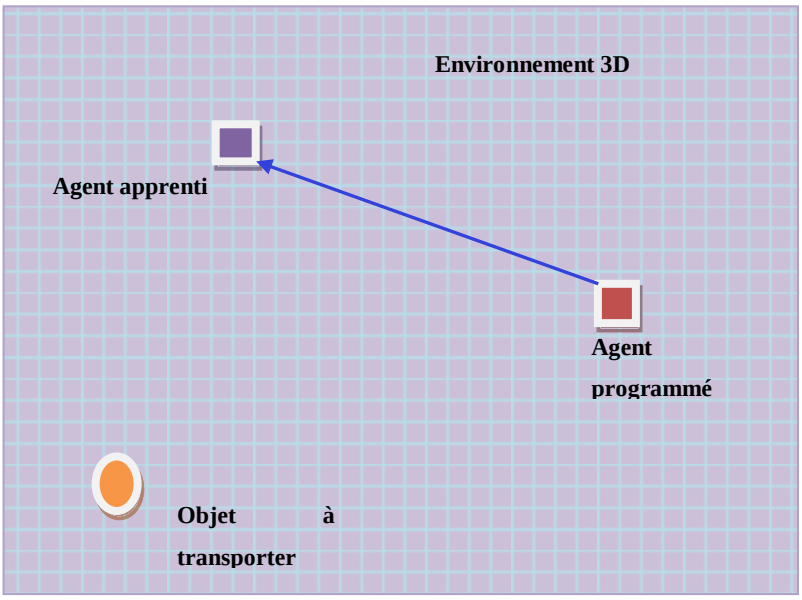


Figure 4.4: L'agent programmé s'envoie des signaux de coordination à l'agent apprenti

Les Effecteurs

L'agent apprenti détermine ses actions à l'aide de son système de classeurs. Comme l'agent programmé utilise le même algorithme pour calculer les signaux à envoyer, l'agent apprenti prend en compte ces signaux dans la détermination de sa récompense, ce qui nous aide à réduire le temps de calcul.

Les actions générés par le système de classeurs sont interprétées comme des poids via les effecteurs afin d'émerger des comportements au niveau de l'observation qui sont résulté par la combinaison pondérée de comportements de base avec les poids appropriés.

L'utilisation de la théorie des *motor schemas* sert à aider l'agent à déterminer son déplacement et à coopérer avec agent programmé.

L'avantage de cette approche est de produire des comportements de haut niveau à partir d'un ensemble de comportements de base (*schemas*) en déterminant les poids de chaque schéma dans le comportement final. Donc, la capacité d'adaptation de l'agent apprenti est améliorée

en changeant automatiquement les poids en fonction des situations à l'aide du système de classeurs.

5. Les conditions initiales de l'application

L'application est réalisée sous certaines conditions qui sont nécessaires au fonctionnement de notre simulation. Le domaine de la vie artificielle ne nécessite pas d'intervention de la part de l'utilisateur car toutes les tâches qui ont été réalisées par le programmeur ou ajustées par l'utilisateur sont désormais à la charge et sous la responsabilité du programme (système de classeurs). Ainsi, nous avons définis tous les paramètres ainsi que les valeurs initiales avant le commencement de la simulation.

La simulation est composée de deux agents l'un est programmé et l'autre appelé agent apprenti. Les positions initiales des agents sont générées de façon aléatoire depuis un fichier qui contient toutes les cordonnés des agents (les positions).

Le système de classeurs utilise :

- 11 capteurs (dont le 3 bits pour indiquer la position de l'agent apprenti dans le cercle autour de l'objet),
- 16 effecteurs encodés par 4 bits,
- Une récompense globale GR=1000,
- $r_{objet\text{-}agent}$=100 unités,
- le plan de la simulation de 1500 × 1500 unités,
- une probabilité de mutation μ=0.094,
- croisement χ=0.5, $p_{\#}$=0.35.
- L'AG est appelé toutes 25 générations pendant la phase d'évolution (exploration).
- Le nombre d'agents (apprenti et programmé) demandés pour la tâche collective est 2 (*nombre d'agents*=2).

6. Les résultats obtenus

Dans cette partie nous présentons les résultats de notre expérimentation, relativement simple, dont le but est de réaliser une tache coopérative pour transporter un objet depuis un endroit

vers un autre par un groupe d'agents, nous nous sommes attelé à évaluer notre architecture comportementale par rapport aux buts que nous nous sommes assignés.

Alors notre simulation s'articule autour des systèmes de classeurs présentés dans le chapitre précédent. Nous donnons une certaine indépendance aux agents et en particulier à notre agent apprenti pour qu'il puisse s'adapter à l'environnement et accomplir le but fixé.

6.1. Analyses des résultats

Notre système est consuls pour simuler des agents dans un environnement 3D, ce système est implémenté sur une machine Intel Pentium IV, de 2.7 Ghz, et avec l'environnement de programmation visuel C++ 2008 version 9 et l'environnement graphique OpenGL

Les résultats obtenus, à l'issue des premières itérations, étaient loin d'être une tache coopérative car l'agent apprenti avait comme premier but d'apprendre de découvrir les paramètres de schémas utilisés par l'agent programmé, afin qu'il produit le comportement le plus satisfaisant.

Le déroulement de notre simulation est réalisé en deux étapes :

La première étape consiste à rechercher le premier but qui est l'objet désirant de transporter, alors l'agent programmé connait ses paramètres et exécute les trois fonctions *perception, décision* et *action* pour que l'agent apprenti essaye d'apprendre de générer ses paramètres à l'aide le système de classeurs et grâce à la coopération avec l'agent programmé (l'envoi de signaux depuis l'agent programmé vers l'agent apprenti (ie : la coordination des actions)).

Après plusieurs itérations l'agent programmé et l'agent apprenti prennent leur place autour de l'objet pour le transporter et tous les états et les paramètres sont sauvegardés dans des fichiers de format text (figure 4.5).

La deuxième étape dans ce cas deux notions s'apparaissent, la notion de maître et d'esclave. On utilise deux booléens dans le programme sert à tester si l'état de collaboration est vrais et le nombre d'agents égale à le nombre total d'agents (nombre d'agents =2), alors l'agent apprenti joue le rôle de maître et prends le rôle du conducteur, sa position initiale de la deuxième étape dépendra de la position qui a été sauvegardée, puis il exécute les trois

fonctions précédentes (i.e perception, décision et action), pour que le collaborateur prend cette position dans le pas suivant. Dans ce cas le collaborateur ne fait que suivre son maître (figure 4.6). L'agent maître teste si le but final est atteint, s'il n'est pas le cas il exécute les trois fonctions de base (figure 4.7).

Les figures suivantes nous montrent comment l'agent programmé et l'agent apprenti coopèrent et prennent l'objet depuis à un endroit x vers un autre.

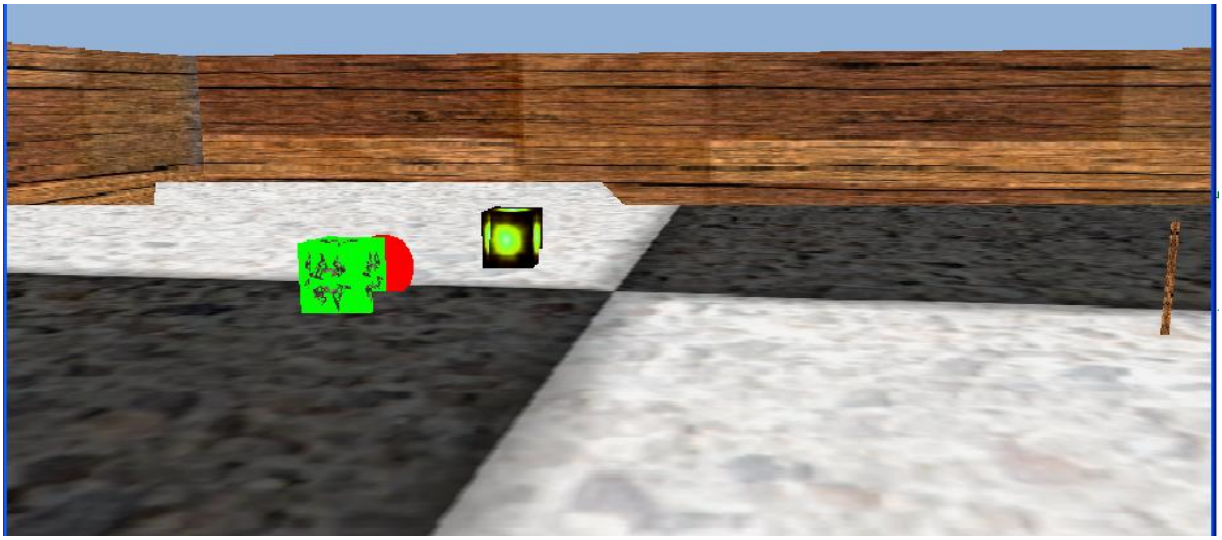


Figure 4.5 : l'agent apprenti et l'agent programmé s'approchent de l'objet

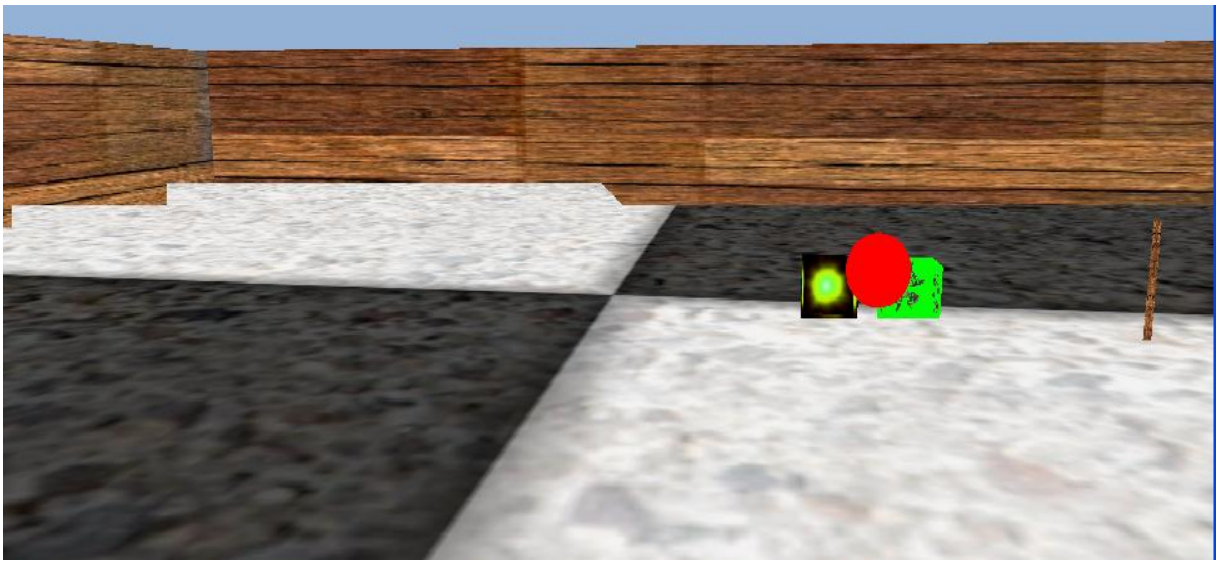


Figure 4.6 : Les agents prennent l'objet

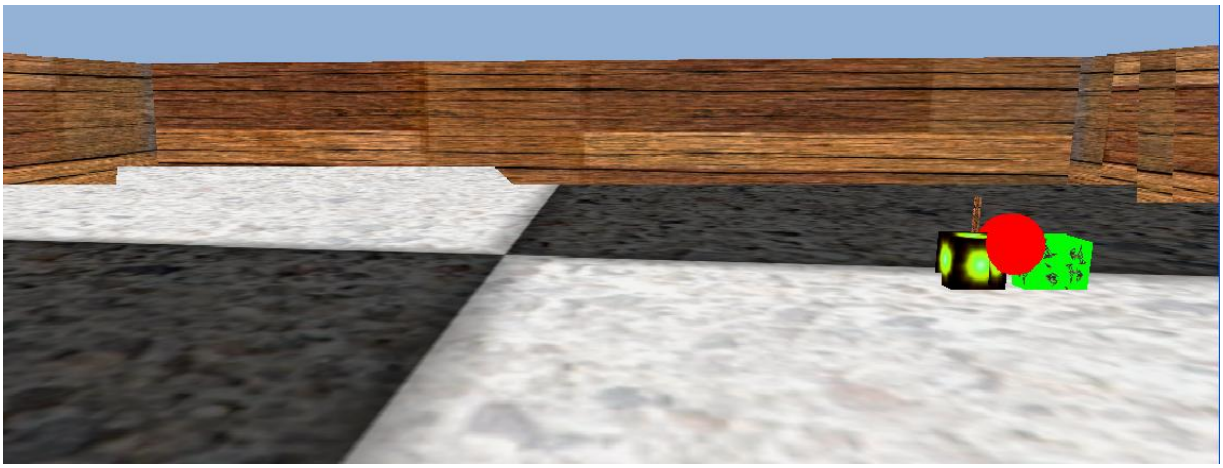


Figure 4.7 : les agents trouvent le but final

7. La discussion des résultats

L'agent apprenti peut corriger ses paramètres par rapport à l'environnement et à l'agent programmé. Trouver le comportement de l'agent est un processus difficile car après avoir trouver l'objet à transporter, il va devoir se positionner de manière pertinente dans l'environnement afin d'arriver au but final.

Nous découvrons 16 actions (voir tableau 4.1) qui déterminent les valeurs des paramètres de schemas *w1, w2*.

Action1	Action2	Action3	Action4	Action5	Action6	Action7	Action8	Action9	Action10	Action11	Action12	Action13	Action14	Action15	Action16
0101	0001	0100	1101	1001	0000	0111	1100	1011	1111	0010	1000	0110	1010	1110	0011

Tableau 4.1 : les 16 actions de système de classeurs

Parmi de ces actions, l'action *1* est celle qui est la plus exécutée. En outre, les autres actions sont aussi sélectionnées en fonction de chaque situation pendant la simulation car le

système de classeurs possède d'autres règles qui coopèrent pour résoudre un problème spécifique. Les résultats prouvent que l'agent apprenti peut déclencher des actions appropriées pour coopérer avec l'agent programmé. Le rôle de quelques actions est décrit comme suit :

- *action1(0101)* : l'agent apprenti découvre les paramètres de schémas utilisés par l'agent programmé et optimiser la coordination. En effet, avec l'utilisation de cette action, l'apprenti produit le comportement le plus satisfaisant.

- *action2(0001)* : L'agent apprenti se tient au milieu de ses voisins.

- *action3(0100)* : L'agent apprenti s'approche de l'objet à transporter.

- *action4(1101), action5(1001)* : Comme l'agent apprenti n'est pas doté du *motor schema déplacer-aléatoire*, il évalue des actions qui peuvent produire un tel comportement similaire et les utilise s'il en a besoin.

- *action6(0000)* : L'agent apprenti s'immobilise.

Dans ce tableau (tableau 4.2) certains classeurs enregistrés sont listés comme suit :

Condition	Action a_j	Prédiction p_j	Erreur ε_j	Fitness f_j
0 1 1 0 # # 0 # 0 # #	0000	263.860291	295.250671	0.337552
0 0 0 0 0 # 1 0 1 0 0	0000	276.259003	659.671265	0.151361
1 0 1 # # # 1 # 1 # #	1101	443.523743	363.879150	0.203510
1 1 1 # 0 1 # # 0 0 #	0100	147.716187	265.446991	0.375309
1 # 1 1 # # 0 1 # 0 #	0100	651.173218	358.955505	0.277812
# 0 0 0 # # 0 0 1 # 1	0010	955.562256	151.577377	0.655405
# 1 # 0 0 0 # 1 1 1 #	0100	974.602180	74.600255	1.335642
# 0 1 0 0 # 1 # 1 # 0	0001	969.800952	6.912056	12.558126
# 0 0 1 # # 0 0 0 01	0001	998.101746	1.526162	35.142109
0 0 1 0 # 0 0 1 # # 0	0001	998.501543	1.252309	43.732541
# 0 0 0 # # 1 0 1 # 0	1001	999.471258	0.984825	50.382286
# 0 0 0 0 0 1 # 1 # #	1001	999.828491	0.113834	85.712030
0 # 1 # 0 0 1 # 1 1 #	0101	999.973816	0.040427	96.114426
# 0 # 0 0 # # 0 1 0 1	0101	999.941345	0.019539	98.083542
0 # 1 # # # 0 # 1 1 #	0101	999.951904	0.005286	99.902779
1 1 0 # 0 0 1 # 1 # #	0101	999.986328	0.000973	99.999978

Tableau 4.2 : Certains classeurs enregistrés

Les résultats pour cette simulation démontrent, à nouveau, les capacités du système de classeurs de faire évoluer les solutions d'un problème donné vers des solutions robustes et intelligentes. Nous avons démontré la capacité des systèmes de classeurs de générer des actions adaptées à un certain problème, dans des temps raisonnables, pour une tâche coopérative.

Dans ce paragraphe on montre la convergence du système d'apprentissage pour générer les actions.

La figure 4.8 montre l'évolution de la récompense globale pour favoriser l'agent apprenti de coopérer avec l'agent programmé.

Au début de la simulation, la tache désirée est loin d'être une tache coopérative parce que la récompense globale pour favoriser la coopération est presque égal à -1000 et après des certaines itérations l'agent apprenti essaye d'apprendre de générer ses paramètres à l'aide le système de classeurs et grâce à la coordination avec l'agent programmé par l'envoi de la récompense globale va augmenter au fur à mesure jusqu'elle stabilise environ 1000 comme illustré dans la courbe.

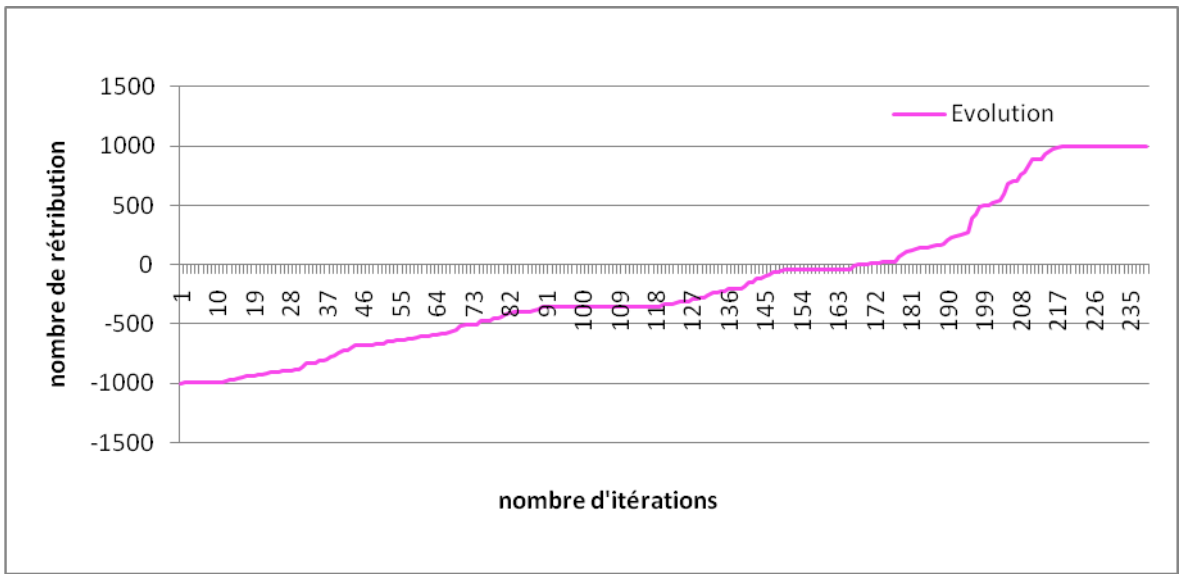


Figure 4.8 : Evolution de la récompense globale.

8. Bilan

Le but de ce travail est de démontrer un cadre pour faire munir, par apprentissage et évolution grâce au système de classeurs des comportements adaptatifs des agents dans un environnement virtuel.

Les résultats des expériences entamées prouvent que la génération des comportements autonomes et adaptatifs dans un environnement virtuel complexe contraint par les systèmes de classeurs est faisable. L'agent simulé a démontré leur capacité d'apprendre, de coopérer et d'interagir par l'intermédiaire d'un modèle d'apprentissage et d'évolution

L'apprenti apprend également à coordonner ses actions à l'aide de son système de classeurs. Le signal de coordination consécutive de son voisin (agent programmé) est considéré comme récompense et utilisée pour mettre à jour les attributs de son système de classeurs tout ça pour produire et émerger des comportements efficaces, et de cette manière, il peut apprendre à s'adapter à son environnement. En outre, il peut faire évoluer son comportement à l'effet de reproduire des comportements efficaces sans une quelconque intervention externe.

Les facultés d'apprentissage, couplées à des capacités d'évolution, font des systèmes de classeurs est une méthode robuste pour la génération de comportements, et une méthode adaptative face à toute éventualité comme une modification dans le déroulement de la simulation.

La majorité des travaux réalisés par les chercheurs travaillant dans le même domaine se sont focalisés, d'une manière générale, sur comment générer des comportements autonomes des entités dans des environnements dynamiques et complexes.

Nous concluons, qu'en plus des expérimentations réalisées, il serait plus intéressant d'aborder de nouvelles expérimentations dans le but de parfaire de générer des comportements plus complexes tels que l'évitement d'objets mobiles et statiques.
Une des principales difficultés est de gérer les différentes représentations du monde et de coordonner des modèles évoluant sur des échelles de temps différentes. pour cela on essaye de combiner l'aspect cognitif dans notre agent apprenti pour apprendre à utiliser sa représentation de l'environnement pour mieux s'adapter et utiliser les caractéristiques anticipatives chez les agents pour une meilleure adaptation dans l'environnement pour

accomplir une tâche donnée cette architecture est dupliquée à toutes les entités pour réaliser une tâche collective.

9. Conclusion

Dans ce chapitre, nous avons évalué notre architecture pour la simulation de comportements des agents dans un environnement virtuel par une expérimentation dans le but de produire des comportements adaptatifs.

L'objectif de cette simulation est de réaliser une stratégie coopérative de deux agents (programmé et apprenti), afin de transporter un objet depuis un endroit vers un autre. L'agent apprenti n'a aucune connaissance a priori et adapte ses actions en fonction de celles de ses partenaires afin de contribuer à la résolution de la tâche coopérative. Pour cela, l'agent apprenti est doté d'un système de classeurs de type *YCS* qui exécute les actions appropriées et modifie les paramètres de *motor schemas*. L'ensemble d'actions possibles est automatiquement déterminé pendant l'apprentissage. Les performances de notre système ont été évaluées.

Conclusion et Perspectives

L'objectif principal de ce mémoire est de proposer une architecture hybride réactive et évolutionnaire, permettant de générer des comportements cohérents. Dans l'optique d'élaborer cette architecture permettant de répondre à la problématique donnée, le premier pas de ce mémoire consistait donc à identifier les éléments nécessaires pour la compréhension puis la formalisation d'un outil de recherche approprié, nous avons concentré notre attention sur les différents types d'agents et la simulation comportementale. Puis on détaille les architectures comportementales qui génèrent des comportements dans des environnements de simulations.

En effet, l'entité virtuelle doit être capable non seulement de se mouvoir dans un environnement mais aussi d'interagir et de coopérer avec les autres entités afin de réaliser des taches collectives. De plus, on recherche dans ces interactions plus d'autonomie afin que le concepteur n'aurais pas la tâche de définir de manière intégrale le comportement à chaque instant de chaque agent pour chaque interaction possible mais plutôt que l'agent soit apte à décider en temps réel de sa propre séquence d'actions , pour cela on s'intéresse à des approches évolutionnaires qui sert à générer des comportements adaptatifs issus de domaine de la vie artificielle comme les systèmes de classeurs qui sont un outil d'apprentissage. Alors Nous avons détaillé ces systèmes du LCS de Holland, les ZCS (Zeroth Level Classifier System) et XCS (eXtended Calssifier System) de Wilson, les ACS (Anticipatory Classifier System), YCS de Bull confinés au départ dans des domaines comme la simulation de créatures virtuelles ou la robotique, leur utilisation s'étend désormais à une multitude de disciplines comme l'économie, la médecine, … Le nombre croissant de publications sur les systèmes de classeurs montre que ce domaine de recherche est en pleine expansion, aussi bien au niveau théorique que pratique.

Notre travail s'est articulé autour de deux axes principaux : la conception et la réalisation d'une architecture comportementale et l'intégration de l'apprentissage dans cette architecture de manière à produire des agents adaptatifs et coopératifs tout en simplifiant le travail de concepteur. Alors, nous avons présenté une architecture hybride s'appuyant sur les modèles existant pour produire une structure plus facile à utiliser. Notre architecture se compose de deux couches; la première est basée sur la théorie des *motor schemas* d'Arkin, elle utilise les caractéristiques de réactivité des agents pour assurer une exécution rapide des actions dans l'environnement sans passer par la planification. Cependant, le concepteur humain doit analyser le problème posé pour trouver des paramètres appropriés qui contrôlent la génération du comportement. Ainsi, nous avons proposé d'intégrer à cette architecture une couche supplémentaire qui prend en charge ces travaux de concepteur. Donc les outils issus de la vie artificielle et plus précisément les systèmes de classeurs sont des procédés performants pour étendre les capacités d'apprentissage d'une entité, et apprennent et explorent l'espace de recherche des paramètres pour déterminer les plus appropriés. Au lieu d'établir manuellement ces paramètres, on fournit les objectifs à atteindre et on laisse le système de classeurs évoluer.
C'est pourquoi nous avons commencé par fournir des systèmes de classeurs pouvant s'intégrer dans cette simulation.
Cette architecture est ensuite évaluée à travers d'une simulation d'une tâche collective qui sert à transporter un objet à l'aide d'une coopération entre deux agents un agent apprenti et l'autre programmé.
Afin de répondre à notre objectif, générer des comportements adaptatifs. Nous avons montré que les systèmes de classeurs sont bien adaptés pour la réalisation d'un tel objectif.

Les réflexions menées au cours de ce travail ouvrent quelques perspectives sur la génération des comportements complexe d'agents autonomes dans des environnements complexes et dynamiques.

Le système de classeurs

Nous envisageons étendre notre architecture en utilisant d'autres extensions des systèmes de classeurs : les ACS afin de simuler l'anticipation chez les agents avec l'utilisation de trois couches (réactif, cognitif et évolutionnaire) pour générer des comportements de plus haut

niveau puisque l'émergence de comportements s'appuie souvent sur la manipulation de connaissances mais pas seulement des informations perçues de l'environnement tel que nous l'avons fait. Dans certains cas l'utilisation de connaissances acquises dans le passé est d'une grande utilité pour atteindre des objectifs. Alors l'utilisation de la mémoire permet de sauvegarder les règles (conditions / actions) qui ont produit une action permettant de se retrouver dans un état désirable (mémoire à long terme), il est important de garder une trace des meilleures règles pour réduire le temps de réponse de l'agent. La mémoire cognitive permet donc de générer des comportements plus complexes qui pourront être utilisés dans le futur afin d'augmenter les capacités d'un agent.

Les comportements

On essayera dans la théorie de motor schémas d'ajouter un troisième comportement de base (*schema*) qui est l'évitement d'obstacles, donc notre nouveau système de classeur essaye de générer trois poids (*w1, w2, w3*), au lieu de générer deux paramètres (*w3* pour *éviter-obstacle*), cette théorie tous jours utilisée pou favoriser les taches collectives.

Les agents

Concernant nos agents utilisés on peut comme perspective remplacer successivement chaque agent programmé par un agent apprenti, muni de cette nouvelle architecture. Nous pouvons ainsi observer l'émergence de nouvelles stratégies coopératives.

Bibliographie

[AC90] Agre (P.E.) et Chapman (D.) -*What are plans for?- Robotic and Autonomous Systems*-, Vol. 6(1-2), pp 17-34, 1990

[AOF01] Alain (D.), Olivier (B.), et François (C.), -*Multi-Agent Systems by Incremental Gradient Reinforcement Learning*-. 17th International Joint Conference on Artificial Intelligence- Seattle, WA, USA: Loria, 2001.

[Ark 92] Arkin (R.C.), -*Behavior-Based Robot Navigation for Extended Domains.*- Adaptive Behavior, 1(2):pp201–225, 1992.

[Ark98] Arkin (R. C.), -*Social Behavior. In Behavior-Based Robotics*-, Chapter 9. MIT Press, Cambridge MA., 1998.

[Bec98] Becheiraz. (P.) -*Un Modèle comportemental et émotionnel pour l'animation d'acteurs virtuels*- Thèse de doctorat de L'EPLF de Lausanne, 1998

[BA98] Balch, T., and Arkin, R.C., -*Behavior-based Formation Control for Multi-robot Teams*.- *IEEE Transactions on Robotics and Automation*, 14(6): 926-939, 1998.

[Bro85] Brooks (R.A.) *A robust layered control system for a mobile robot*-. Rapport technique de "Journal of Robotics and Automation", pp 14-23, 1985.

[Bro85] Brooks (R.A.) *A robust layered control system for a mobile robot*-. Rapport technique de "Journal of Robotics and Automation", pp 14-23, 1985.

[Bro86] Brooks (R.A.) -*A robust layered control system for a mobile robot*-. IEEE Journal of Robotics and Automation pp. 14–23, 1986.

[Bro89] Brooks (R.A.) -*How to build complete creatures rather than isolated cognitive simulators*-. in K. VanLehn (éd.), Architectures for Intelligence, pp. 225–239. Lawrence Erlbaum Associates, 1989.

[BSB+05] Bull (L.), Studley (M.), Bagnall, (A.) et Whittley (I.), -*On the Use of Rule Sharing in Learning Classifier System Ensembles.*- *CEC'2005, Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 12-617, 2005.

[BST+04] Bull, (L.), Sha'Aban,(J.), Tomlinson, (A.), Addison, (P.), et Heydecker, (B.), -*Towards Distributed Adaptive Control for Road Traffic Junction Signals using Learning Classifier Systems*-. *Applications of Learning Classifier Systems*, L. Bull (Ed.), Springer, pp. 276-299, 2004.

[Bul03] Bull (L.), -*A Simple Accuracy-based Learning Classifier System*-. Technical Report UWELCSG03-005, 2003.

[CG95] Cliff (D) et Geoffrey (F. M). -*Tracking the red queen: Measurements of adaptive progress in co-evolutionary simulations*-. In Berlin Springer-Verlag (ed.), Proceedings of the Third European Conference on Artificial Life, pp. 200–218 .1995.

[CG96] Cliff (D) et Geoffrey (F. M). -*Co-evolution of pursuit and evasion*- ii : Simulation methods and results. In USA MIT Press, Cambridge (ed.), From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior, pp. 506–514 1996.

[Coe95] Coen (M.) -SodaBot*: -A Brief Introduction-*. Rapport technique. Institut MIT.1995.

[CR95] Cliff (D.) et Ross (S.) -*Adding Temporary Memory to ZCS.*-. Technical Report CSRP347, School of Cognitive and Computing Sciences, University of Sussex, 1995.

[CT05] Conde (T.) et Thalmann (D.) -*Autonomous virtual agents learning a cognitive model and evolving, intelligent virtual agents.*- pp:88–98 ,2005.

[DAM00] Durand (N.), Alliot (J.M.), Medioni (F,)"*Neural Nets trained by genetic algorithms for collision avoidance"*, Applied Artificial Intelligence, Vol13, 2000.

[Dej75] De Jong (K.A). -*An analysis of the behaviour of a class of genetic adaptative systems*- Thèse de doctorat, Department of Computer and Communication Sciences, University of Michigan. 1975.

[Dej 99] De Jong, E.D., Coordination Developed by Learning from Evaluations.J.A. Padget (ed.) *Collaboration between Human and Artificial Societies*, pp. 234-245, 1999.

[DM99] Drogoul (A.)et Meyer (J.A.,)"Intelligence artificielle située : cerveau, corps et environnement", Hermès 1999.

[DS93] Dorigo (M.) et Schnepf (U.) -*Genetics-based Machine Learning and Behaviour Based Robotics*-: A New Synthesis". IEEE Transactions on Systems, Man and Cybernetics, 23(1), pp141-154, 1993.

[Fer95] Ferber (J.) -*Vers une intelligence collective*-. Inter Editions, 1995.

[Fer97] Ferber (J.) -*Les systèmes multi-agents: vers une intelligence collective*-,

[1997.

FG96] Franklin (S.) et Grasser (A.) -*Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents*-. In Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag. 1996.

[FS05] Fabien (F.) & Sigaud (O.) -*GACS, an evolutionary approach to the spatia coordination of agents*-. In Proceedings AAMAS 2005, pp. 1109–1110, Utrecht, The Netherlands. ACM Press 2005.

[FTT99] Funge (J.), Tu (X.), et Terzopoulos (D).Cognitive modeling : -*Knowledge, reasoning and planning for intelligent characters*-. In Proceedings of the 26th annual conference on Computer graphics and interactive techniques, pages 29, 38. ACM Press/Addison-Wesley Publishing Co. New York, NY, USA. 1999.

[Gol89] Goldberg (D.) *"Genetic Algorithms in Search, Optimization, and Machine Learning"*. Addison-Wesley, Reading, Mass., 1989.

[GRS04] Guessoum (Z.), Rejeb(L.), et Sigaud(O.),-*Using XCS to build Adaptive Agents*-. *Fourth Symposium on Adaptive Agents and Multi-Agent Systems (AAMAS-4)*, AISB convention, pp. 101-106, Leeds, UK, 2004.

[Hau07] Hau (T.T).-*Approches évolutionnaires pour le comportement adaptatif d'entités autonomes*-. Thèse de Doctorat. Université de Toulouse 3 Paul Sabatier. 2007.

[Heg03] Heguy (O.) -*Architecture comportementale pour l'émergence d'activités coopératives en environnement virtuel.*- Thèse de Doctorat. Université Paul Sabatier de Toulouse, 2003

[HHN+86] Holland (J.H.), Holyoak (K.J.), Nisbett (R.I.), et Thagard (P.R.).

"*Induction: Processes of Inference, Learning, and Discovery"*. MIT Press, Cambridge, 1986.

[HHS94] Ho (T.), Hull, (J. J.) et Srihari (S. N.), -*Decision Combination in Multiple Classifier Systems.*- *IEEE* Trans on *PAMI* 16(1). pp.66-75, 1994.

[HLT+03] Heigeas (L.), Luciani (A.), Thollot (J.) et Castagné (N.), -*A Physically-Based Particle Model of Emergent Crowd Behaviors*-, Graphicon 2003.

[HM91] Holland (J. H.) et Miller (J. H.), -*Artificial Adaptive Agents Theory*,- American Economic Review, pp. 365-370, 1991.

[Hol75] Holland (J.H.) -*Adaptation in Natural and Artificial Systems*.- University of Michigan Press ,1975.

[HR78] Holland (J. H.) et Reitman (J.) -*Cognitive Systems based on Adaptive Algorithms*-, Pattern Directed Inference Systems pp .313-329, New York : Academic Press, 1978.

[Inn 92] Inn (A. F) -*Touring machines: Autonomous agents with attitudes.* - EEE Computer 25(5) .pp.51–55, 1992.

[Kha03] Khamassi (M.) -*Un modèle d'apprentissage par renforcement dans une architecture de contrôle de la sélection de l'action chez le rat artificiel*- Psikharpax. Spécialité Sciences Cognitives ,2003.

[Kov97] Kovacs (T.) - *XCS Classifier System Reliably Evolves Accurate, Complete, and Minimal Representations for Boolean Functions. WSC2-:* Second Online World Conference on Soft Computing in Engineering Design and Manufacturing, London, UK, Juin 1997.

[Koz92] Koza (J. R.) -*Genetic Programming*-, MIT Press, 1992.

[KY03] Katagami (D.), et Yamada (S.), -*Active Teaching for an Interactive Learning Robot*-. *12th IEEE Workshop Robot and Human Interactive Communication (ROMAN2003)*, pp. 181-186, San Francisco, USA, 2003.

[Lan97] Lanzi (P.L.) -*A study of the generalization capabilities of XCS*- in T. Baeck, editor, "Proceeding of the Seventh Intrenational Conference on Genetic Algorithm, pp 418-425, San Francisco, California; 1997

[Lan98] Lanzi P. L., -*An Analysis of the Memory Mechanism of XCSM*-, GP98, Third Genetic Programming Conference, San Francisco, USA, 1998.

[Lat91] Latombe. J. C., -*Robot Motion Planing.-:* Kluwer Academic Publisher, 1991.

[Mae89] Maes (P.), -*How to do the right thing*.- Connection Science Journal, Vol 1(3),pp. 291-323, 1989.

[Mae 94] Maes (P.) Social Interface Agents: -*Acquiring Competence by Learning from Users and Other Agents*-. *Rapport technique*, pp. 71-78, AAAI Press, Mars 1994.

[MG91] Jean-Arcady Meyer & Agnes Guillot. -*Simulation of adaptive behavior in animats*-: review and prospect. In J.-A. Meyer et S. W. Wilson (ed.), In From animals to animats : Proceedings of the First International Conference on the Simulation of Adaptive Behavior, Cambridge, MA. The MIT Press/Bradford Books, 1991.

[MSD+03] Meyer (J-A.), Stéphane (D.), David (F.), et Agnès (G.). -*Evolutionary approaches to neural control of rolling, walking, swimming and flying animats or robots*-. Physica-Verlag GmbH, Heidelberg, Germany, Germany 2003.

[NS76] Newell (A.) et Simon (H.A.) -*Computer Science as Empirical Enquiry*-, Communications of the ACM, Vol. 19, pp. 113-126, 1976.

[ODD+09] Ouannes (N.), Djedi (N), Duthen (Y) et Luga (H.), -*Evolution du contrôleur d'un robot humanoïde par un algorithme génétique dans un environnement complexe*-, Symposium IMAGE'2009, Algérie, 2009

[OMS+07] Ontanon (S.), Mishra (K.), Sugandh(N.), et Ram,(A.) - *Case based planning and execution for real-time strategy games*- 2007.

[Pan03] David Panzoli.- *Simulation comportementale par réseau de neurones et apprentissage par algorithme génétique*-. Thèse de Master, IRIT, Toulouse.2003.

[Pan08] Panzoli David thèse de Doctorat « *Simulation d'entités artificielles dans les environnements dynamiques virtuels* ». 2008

[RA98] Richard (S.S) et Andrew (G.B). -*Reinforcement Learning: An Introduction*-. MIT Press, Cambridge, MA 1998.

[Ram07] Ramos (M. A.) -*Etude et proposition d'un système comportementale autonome anticipatif*.- Thèse de doctorat, Université Toulouse ,2007.

[Rey87] Reynolds (C.W.). - *Flocks, herds and schools: a distributed behavioral model – Dans:* SIGGRAPH'87, Vol. 21(4) of Computer Graphics, pp 25-34, ACM Press, Anaheim (USA), 1987.

[Rey99] Reynolds (C.W.). - *Steering Behaviors For Autonomous Characters*. – *GDC 99*. pp. 763-782 A. Yu (Ed.) Miller Freeman, San Fransisco, 1999.

[Ric 01] Richard (N.) -*Description de comportement d'agent autonomes évoluant dans des mondes virtuels*-. Thèse de doctorat, l'École Nationale Supérieure des Télécommunications (ENST). Spécialité :

Informatique et Réseaux. 2001.

[RN95] Russell (S.J.) et Norvig (P.) -*Artificial Intelligence: A Modern Approach*-, Rapport technique. pp 912. 1995.

[San01] Sanza (C.) - *Evolution d'entités virtuelles coopératives par systèmes de classifieurs*-. Thèse de Doctorat, Université Paul Sabatier, Toulouse, 2001.

[San04] Sanchez (S.) -*Une architecture générique intégrant des mécanismes évolutionnistes pour la simulation comportementale d'acteurs virtuels*-. Thèse de Doctorat. Université des sciences sociales, 2004.

[Sch93] Schwartz (A.) -*A reinforcement learning method for maximizing undiscounted reward*- in Machine Learning Proceeding of the Tenth International Worshop. San Mateo, CA, Morgan Kaufmann. 1993

[SDD99] Sanza (C.) ,Destruel (C.) et Duthen (Y.) -*Evolution of autonomous actors in an interactive real-time environment.*- 1999.

[SG01] Sigaud (O.) et Gérard (P.), -*The use of roles in a multi-agent adaptive simulation.* - AnimatLab PARIS, 2001.

[Sim91] Sims (K.) -*Artificial evolution for computer graphics.*- In SIGGRAPH'91 Proceedings, pp. 319–328 ,1991.

[Sim94a] Sims (K.) -*Evolving 3d morphology and behavior by competition.*- In Brooks & Maes (eds.), Artificial Life IV Proceedings, pp. 28–39 ,1994.

[Sim94b] Sims (K.) -*Evolving virtual creatures.*- In SIGGRAPH'94 Proceedings, pp. 15–22,1994.

[Ste94] Steels (L.) -*The artificial life robots of artificial intelligence*-. in Artificial Life Journal, Vol. 1(1), pp. 75–110. MIT Press, 1994.

[Sto96] Stolzmann (W.) *"Learning Classifier Systems using the Cognitive Mechanism of Anticipatory Behavioral Control"*, First European Workshop on Cognitive Modeling, pp.14-16 . 1996.

[SLD+04] Sanchez (S.), Luga (H.), Duthen (Y.), et Balet (O.). -*Bringing autonomy to virtual characters.*- In Fourth IEEE International Symposium and School on Advance Distributed Systems. Published in Lecture Notes in Computer Science, Springer, vol. 3061, Mexico,2004.

[SLD08] Sylvain (C.), Hervé (L) et Yves (D.) -*Développement de créatures artificielles primitives possédant un métabolisme et une morphologie*-, Toulouse IRIT Presse ,AFIG, 2008.

[Syl09] Sylvain (C.), Creatures Artificielles : -*Développement d'Organismes à partir d'une Cellule Unique-thèse de doctorat 2009*

[Tan04] Tan (K. C.), -*Recent advances in simulated evolution and learning.*- Singapore; River Edge, NJ: World Scientific, 2004.

[TB99] Tomlinson (A.) et Bull (L.)-*A Zeroth Level Corporate Classifier System*- in A.S. Wu, editor, Proceedings of the Genetic and Evolutionary Computation Conference Workshop Program. GECCO-99, pp 306-313. 1999

[Tom99] Tomlinson (A.) -*Corporate Classifier Systems*-. Thèse de doctorat, University of the West of England, 1999.

[TT94] Tu (X.) et Terzopoulos (D.)- *Artificial Fishes: physics, locomotion, perception, behaviour*. – Dans: SIGGRAPH 94 Conference Proceedings, pp 43-50, Orlando, FL, USA, 1994.

[TTG94] Terzopulos (D.), Tu (X.) et Grzeszczuk (R.). -*Artificial Fishes with*

Autonomous Locomotion, Perception, and Learning in a Simulated Physical World-. Artificial Life IV, R. Brooks et P. Maes Editors, MIT Press, pp 17-27, 1994

[Tyr93] Tyrrell (T.) -*Computational mechanisms for action selection.*- Thèse de doctorat, University of Edinburgh, 1993.

[Ven94] Venturini (G.,)- *Apprentissage Adaptatif et Apprentissage Supervisé par Algorithme Génétique*-. Thèse de doctorat, Université de Paris-Sud, France, 1994.

[Wat 89] Watkins (C.J.C.H.) -*Learning from delayed rewards.*- Doctoral dissertation, King's College, Cambridge UK, 1989.

[WH60] Widrow (B.) et Hoff, (M.), -*Adaptative Switching Circuits.*- Western Electronic Show and Convention, 4, pp. 96-104, 1960.

[Wil85] Wilson (S.W.) - *Knowledge growth in an artificial animal*.- Proceedings of the First International Conference on Genetic Algorithm and Their Applications, pp 16-23, Lawrence Erlbaum Associates. Hillsdale, New-Jersey. 1985.

[Wil87] Wilson (S.W.) -*Classifier Systems and the Animat Problem.*- *Machine Learning*, pp.199-228, 1987. Also Research Memo RIS-36r, the Rowland Institute for Science, Cambridge, MA, 1987

[Wil94] Wilson (S.W.), -*ZCS: A Zeroth Level Classifier System.*- *Evolutionary Computation*, 2(1), pp, 1-18, 1994.

[Wil95] Wilson S.W. -*Classifier Fitness Based on Accuracy*.- Evolutionary Computation, 3(2),pp,149-175, 1995

[Wil98] Wilson (S.W.), -*Generalization in the XCS Classifier System.*-

[WJ95] Genetic Programming 1998: Proceedings of the Third Annual Conference, pp. 665-674. 1998.

Wooldridge (M.) et Jennings (N.R.) -*Intelligent agents: theory and practice*-. Revue Knowledge Eng. vol 10(2), pp115-152. 1995.