Construction d'individualité par un mécanisme de sélection d'action basé sur les motivations

THÈSE

présentée et soutenue publiquement le 4 novembre 2010

pour l'obtention du

Doctorat de l'Université Lille 1 - Sciences et Technologies

(spécialité informatique)

par

Tony Dujardin

Composition du jury

Président :	Mohamed Daoudi, Professeur	LIFL, Université Lille1
Rapporteurs :	Rachid Alami, Directeur de Recherche	LAAS, Toulouse
	Pierre Chevaillier, HDR	CERV, ENIB, Brest
Examinateur :	Tristan Cazenave, Professeur	LAMSADE, Université Paris-Dauphine
Directeurs :	Jean-Christophe Routier, Professeur	LIFL, Université Lille1
	Philippe Mathieu, Professeur	LIFL, Université Lille1

UNIVERSITÉ LILLE 1 - SCIENCES ET TECHNOLOGIES
Laboratoire d'Informatique Fondamentale de Lille — UMR USTL/CNRS 8022
U.F.R. d'I.E.E.A. – Bât. M3 – 59655 VILLENEUVE D'ASCQ CEDEX
Tél. : +33 (0)3 28 77 85 41 – Télécopie : +33 (0)3 28 77 85 37 – email : direction@lifl.fr

« Le commencement de toutes les sciences, c'est l'étonnement de ce que les choses sont ce qu'elles sont. »
Aristote

Résumé

Cette thèse est une proposition pour la conception de comportements crédibles dans les simulations informatiques pour agents situés dans un environnement virtuel. Le comportement d'un agent se définit à partir de l'observation des actions qu'il exécute dans un environnement. Chaque action exécutée résulte d'un choix de l'agent parmi l'ensemble des actions qu'il peut effectuer. Le comportement se construit donc à partir des capacités de l'agent lui permettant de résoudre ses buts, le raisonnement, et d'un choix influencé par ses traits de caractère, l'individualité. Ces traits de caractère peuvent être de natures différentes, par exemple les préférences de l'agent sur les actions qu'il peut exécuter, la prise en compte de la proximité avec la cible d'une action ou encore la persévérance dans une résolution en cours. De par leurs natures différentes, il est important de définir une structure permettant à la fois de prendre compte tous ces éléments, d'autoriser l'ajout et la suppression d'un trait de caractère tout en gardant la cohérence du comportement obtenu. Enfin chaque trait de caractère doit être paramétrable simplement et réutilisable pour la construction d'autres individualités.

Ma contribution consiste à définir des comportements qui sont composés de ces deux parties : le *raisonnement* et l'*individualité*. Le raisonnement utilise un planificateur pour déduire l'ensemble des résolutions possibles des buts de l'agent à partir de ses connaissances et de ses capacités. L'individualité utilise un mécanisme de sélection d'action afin de déterminer la meilleure action parmi l'ensemble des actions exécutables.

De nombreux travaux ont déjà été effectués sur les planificateurs, cette thèse se concentre sur la proposition d'un mécanisme de sélection d'action pour la partie individualité du comportement. Ce mécanisme se base sur les notions de *motivations* et d'*alternatives*. Les motivations sont l'expression de traits du caractère de l'agent, elles sont indépendantes et elles influencent le choix de l'action à exécuter. Les alternatives sont les résolutions possibles calculées par la partie raisonnement du comportement. Le mécanisme de sélection d'action s'appuie sur les alternatives pour déterminer, à l'aide des motivations la meilleure action à exécuter à chaque instant.

Les motivations influençant le comportement de l'agent sont définies indépendamment du contexte d'application (et indépendamment les unes des autres) permettant leur réutilisation (même partiellement) pour d'autres agents et d'autres simulations. Ma contribution vise également à apporter un enrichissement au projet CoCoA par l'apport d'un mécanisme de sélection d'action concret basé sur les motivations ainsi que la réalisation d'un atelier de conception de comportement. L'enrichissement d'un planificateur basé sur l'approche centrée interaction, promue dans le projet CoCoA, par mon mécanisme de sélection d'action basé sur les motivations permet d'obtenir un moteur comportemental identique pour les agents. Par sa généricité, ce moteur peut être utilisé dans différents environnements et pour différents agents en s'adaptant aux spécificités et capacités de chacun, tout en proposant une diversité dans les comportements réalisables.

Abstract

This thesis proposes a new design approach for building software agents that reflect believable behaviors in simulated virtual environments. Observing a specific agent's actions usually leads to defining its overall behavior schema, which is done according to the actions it performs while attempting to reach a goal and, the choices this agent decides to make because of specific personality traits. Due to the diversity of nature related to personality traits, defining an agent's behavior is a challenging problem since it requires, for example, the consideration of agent's preferences concerning the actions it can perform, agent's closeness to the action target, and the level of agent's insistence to reach a certain goal. The consistency of an agent's behavior is achieved by means of a structured design that considers the changeability of personality traits while making them possible to reconfigure or reuse by other agents, plus, the impacts it makes on an agent's actions' selection.

The contribution of this thesis is directly related to the design phase wherein an agent's behavior is to be defined. Defining an agent's behavior consists of *reasoning* and *individuality*. In the reasoning part, relying on a planner is required to infer all possible resolutions of agent's goals from its knowledge and abilities. In the individuality part, we introduce an action selection mechanism for determining an agent's subsequent move from all possible ones.

This thesis focuses on the introduction of a new action selection mechanism to an agent behavior's individuality. The main concepts behind the design of our action selection mechanism are *motivations* and *alternatives*. We look at motivations as the independent expressions of a specific agent's traits that influence its upcoming action selection. We look at alternatives as possible resolutions computed by the reasoning part of an agent's behavior.

In our approach, the motivations influencing agents' behaviors are domain-free. Therefore, we were able to utilize our work in contributing to the CoCoA project by providing a concrete motivation-based action selection mechanism. Further to the interaction-oriented approach promoted by the CoCoA project, the action selection mechanism we propose makes it possible to provide the means to have reliable behavioral engine that is the same for all agents. This generic engine can be used in different environments and for different agents. It also takes into account the agent's abilities and individualities to provide diversified and realistic behaviors.

Remerciements

Je tiens tout d'abord à remercier l'ensemble des membres de mon jury de thèse : Mohammed Daoudi pour avoir accepté de le présider, Rachid Alami et Pierre Chevaillier pour avoir accepté d'être rapporteurs et d'avoir permis l'amélioration et l'aboutissement que représente ce document et enfin Tristan Cazenave pour avoir examiné mon travail.

Je tiens également à remercier mes directeurs Jean-Christophe Routier et Philippe Mathieu, pour avoir su me guider tout en me laissant libre de mes choix, pour ce sujet passionnant et motivant ainsi que pour les discussions qui m'ont permis d'avancer.

Merci également à toute l'équipe Systèmes Multi-Agents et Comportements du Laboratoire d'Informatique Fondamentale de Lille, pour leur accueil et leur aide à chaque étape de mes travaux. Merci à Sameh Abdelnaby, Bruno Beaufils, Jean-Paul Delahaye, Patricia Everaere, Yoann Kubera, Jean-Baptiste Leroy, Maxime Morge, Antoine Nongaillard, David Panzoli, Sébastien Picault et Irina Veryzhenko. Je tiens à rendre hommage en particulier Yann Secq pour avoir été mon tuteur du coté enseignement avec qui j'ai beaucoup aimé travailler.

Je remercie mes amis, Benjamin Barbry, Christophe Bouin, Meriem Bouisouden, Fadila Khadar, François Lecroart, Frédéric Maquet, Ludovic Martin, Sylvain Petit, Asli Ürgüplü et François Vantomme. Je n'oublie pas non plus les anciens doctorants SMAC qui sont également mes amis, Laetitia Bonte, Damien Devigne et Julien Derveeuw, qui ont su m'aider et me conseiller depuis le Master Recherche.

Sans oublier Moulay-Driss Benchiboun et Jean-Marie Place, pour m'avoir permis d'enseigner au département informatique de l'IUT 'A' de Lille.

Un grand merci à mes parents, Myriam et Jacques et à ma famille, en particulier, Delphine et Lucie pour tout le soutien et la patience dont ils ont fait preuve. Enfin, un merci à ceux qui ne sont plus là aujourd'hui mais que je n'oublierai jamais.

Table des matières

Introduction Générale

Le comportement d'un agent se définit à partir de l'observation des actions qu'il exécute dans un environnement. Chaque action exécutée résulte d'un choix de l'agent parmi ses capacités. Le comportement se construit donc à partir des actions que l'agent peut effectuer pour résoudre ses buts (le raisonnement) et d'un choix influencé par ses traits de caractère (l'individualité). Cette thèse se situe dans le cadre des simulations informatiques de comportements pour des agents situés dans un environnement virtuel.

L'application principale visée par mes travaux de simulation de comportements est les jeux vidéo. Comme le dit John Laird le jeu vidéo est la "killer application" de la modélisation de comportements humains[LvL00]. La modélisation de comportements dans les jeux vidéo est un critère important. Plus les comportements des personnages dans un jeu sont variés, plus l'immersion du joueur est intense et captivante. Parmi les techniques d'IA les plus utilisées dans les jeux vidéo, on retrouve principalement les scripts et les machines à état finis [Ork06]. Ces techniques reposent sur l'énumération des situations possibles et la définition pour chaque situation de l'action que le personnage doit effectuer. Ainsi, accroître la diversité des comportements implique d'augmenter le nombre de situations différentes (afin notamment de surprendre le joueur), rendant la conception des comportements longue et difficile. Depuis quelques années, des outils sont créés afin de simplifier cette conception et rendre le plus réutilisable possible un travail déjà effectué. Certains de ces outils possèdent notamment des interfaces représentant les comportements sous forme de graphes pour en simplifier la compréhension. Bien que ces outils permettent d'alléger la tâche du concepteur, concevoir un comportement à partir de l'ensemble des situations présentes dans un jeu reste une tâche longue et compliquée.

Si l'on désire définir un comportement particulier, cela ne requiert pas d'information sur les autres agents ou sur la simulation dans laquelle l'agent va être utilisé. Bien qu'il puisse être important de prendre en compte son contexte (son environnement et les autres agents), pour être crédible la définition d'un comportement ne nécessite pas d'en être dépendant. Définir un agent gourmand, ne requiert pas d'information sur l'environnement dans lequel l'agent va être utilisé. Par contre, l'expression de sa gourmandise peut être contrainte par l'environnement (s'il n'y a rien à manger par exemple). C'est pourquoi, je propose de définir le comportement de l'agent à travers les facteurs qui le poussent à agir, indépendamment des situations ou de l'environnement dans lequel l'agent sera utilisé. Ainsi, ce n'est plus la situation qui va déterminer le comportement de l'agent mais ces facteurs qui vont être adaptés à la situation pour influencer le comportement de l'agent. Cette application de la définition du comportement à la situation doit être déléguée à une procédure automatique afin de libérer le concepteur de cette tâche pour lui permettre de se focaliser sur la conception du comportement de l'agent. Le but de cette thèse est de faire une proposition dans ce sens.

Ma contribution consiste à définir des comportements qui sont composés de deux parties : le **raisonnement** et l'**individualité**. Le comportement est construit à partir des capacités de l'agent lui permettant de résoudre ses buts, le raisonnement, et d'un choix influencé par ses traits de caractère, l'individualité. Le raisonnement utilise un planificateur pour déduire l'ensemble des résolutions possibles des buts de l'agent à partir de ses connaissances et de ses capacités. L'individualité utilise un mécanisme de sélection d'action (ASM pour *Action Selection Mechanism*) afin de déterminer la meilleure action parmi l'ensemble des actions exécutables.

De nombreux travaux ont déjà été effectués sur les planificateurs, cette thèse se concentre sur la partie individualité et sur la proposition d'un mécanisme de sélection d'action. Ce mécanisme se base sur les notions d'alternative et de motivation. Une alternative est une séquence d'actions composée d'une action exécutable, d'actions intermédiaires et de l'action permettant de résoudre un but. Les alternatives sont des prévisions à moyen terme des résolutions possibles des buts, elles sont calculées par la partie raisonnement du comportement. Les motivations sont l'expression de traits du caractère de l'agent qui influencent le choix de l'action à exécuter. Leur rôle est d'évaluer à chaque instant les actions exécutables par l'agent en prenant en compte les alternatives correspondantes. Le mécanisme de sélection d'action est le même pour tous les agents, son rôle est de combiner les évaluations des motivations afin de déterminer la meilleure action à exécuter. Les motivations sont indépendantes les unes des autres, rendant le mécanisme de sélection d'action robuste aux ajouts et aux suppressions de motivation. Les motivations sont définies indépendamment du contexte d'application du comportement, elles sont donc réutilisables pour d'autres agents et d'autres simulations. Enfin, les motivations sont paramétrables afin de créer des profils comportementaux. Ainsi, deux agents dans la même situation, ayant le même ASM, les mêmes capacités, les mêmes connaissances et les mêmes motivations peuvent effectuer des actions différentes, s'ils possèdent des profils comportementaux différents.

La réutilisation du travail est un aspect important de l'allègement du temps consacré à la conception. Le but est de définir des agents avec des comportements pouvant exprimer des traits de caractère. Les agents ne sont pas seulement distingués selon leurs capacités mais également suivant la manière dont ils décident d'effectuer une action. Ma proposition repose sur une définition d'individualité sans a priori sur l'environnement dans lequel le comportement sera appliqué. Cette définition se veut être réutilisable (même partiellement) pour d'autres environnements et d'autres agents.

De plus, mes travaux s'inscrivent dans le cadre du projet CoCoA de l'équipe SMAC de Lille. Ma contribution vise à apporter un enrichissement au projet du point de vue de la conception d'individualité. Cet enrichissement du planificateur du projet CoCoA basé sur l'approche centrée interaction par mon mécanisme de sélection d'action permet d'obtenir un moteur comportemental générique et une définition du comportement réutilisable tant sur la partie raisonnement que sur la partie individualité. Afin de pouvoir expérimenter ce moteur, j'ai implémenté un mécanisme de sélection d'action concret dans la plateforme de simulation CoCoA. Ce mécanisme extrait les alternatives à partir d'un plan (i.e. un $Arbre - et/ou$) produit par la plateforme de simulation (la partie raisonnement du comportement). Puis, l'ASM évalue chaque action exécutable (et donc chaque alternative) à l'aide de sept motivations. Chaque motivation donne une évaluation indépendante pour chaque alternative. Ces évaluations sont ensuite combinées afin de déterminer la meilleure action que l'agent doit exécuter

(la partie individualité du comportement). Ces sept motivations répondent aux critères de Tyrrell définissant un bon mécanisme de sélection d'action[Tyr93b]. Un atelier de conception de comportement a été également mis en place afin de faciliter la construction de l'individualité des agents CoCoA. Enfin, plusieurs simulations ont été réalisées afin de présenter les différents aspects du mécanisme de sélection d'action pour la conception de l'individualité des agents.

Organisation du document

La conception de comportement regroupe plusieurs parties. Tout d'abord, le modèle, puis la méthodologie de conception, ensuite l'implémentation et enfin l'interface de conception. Un modèle, ou ce qui peut être présenté sous la forme d'une architecture, permet de déterminer comment le comportement est défini. Une méthodologie de conception définit comment concevoir un comportement ainsi que les principes de cette conception. Une implémentation permet de concrétiser le modèle afin d'exécuter les comportements définis. Une interface de conception fournit les outils nécessaires à un utilisateur pour construire des comportements pour le modèle en suivant la méthodologie afin de pouvoir l'exécuter.

Le premier chapitre est consacré au contexte de mes travaux. Je commencerai par donner une définition du comportement avant de parler de sa modélisation. Je présenterai ensuite des travaux relatifs à la conception de comportement qui ont marqué ou influencé mes recherches. Certains de ces travaux sont évidemment destinés à la conception de comportements pour les jeux vidéo. Ce chapitre présente également l'approche centrée interaction de l'équipe SMAC de Lille. Cette approche a notamment donné naissance au projet CoCoA.

Le chapitre deux est un état des lieux du projet CoCoA avant mes travaux de recherche sur la conception de comportement. Le projet CoCoA se compose également d'une plateforme de simulation sur laquelle j'ai pu réaliser une implémentation concrète du modèle que je propose. Le projet CoCoA offrait déjà certaines solutions de conception pour des agents cognitifs situés dont le comportement est orienté par des buts. Ce chapitre permet donc de mieux appréhender l'implémentation qui a été faite dans CoCoA (présentée dans le chapitre quatre).

Le chapitre trois présente le modèle que je propose pour la conception de comportement. Ce modèle est basé sur la séparation de la partie raisonnement et de la partie individualité du comportement. La partie raisonnement est déléguée à un planificateur dont la tâche sera de définir les résolutions possibles des buts de l'agent, en fonction de ses capacités et de ses connaissances. La partie individualité est gérée par un mécanisme de sélection d'action basé sur les notions de motivation et d'alternative. Les motivations sont vues comme des éléments influençant le choix des actions à effectuer et donc le comportement de l'agent. Les alternatives sont les résolutions possibles sur lesquelles le mécanisme de sélection s'appuie afin de choisir à chaque instant la meilleure action à exécuter. Ce chapitre présente également la méthodologie de conception ainsi que les contraintes auxquelles doivent répondre toutes les implémentations du modèle.

Le chapitre quatre illustre les points présentés au chapitre précédent à travers l'implémentation du mécanisme de sélection d'action réalisée dans la plateforme CoCoA. Cette implémentation nécessitait la prise en compte de trois éléments influençant le comportement : les propriétés, les capacités de l'agent et les motivations. De plus, afin d'ajouter des possibili-

tés supplémentaires de modélisation et d'évolution du comportement de l'agent, ce chapitre présente la mise en place de propriétés qui évoluent durant la simulation dans CoCoA. Le mécanisme de sélection d'action basée sur les motivations nécessite l'apport d'alternatives de la partie raisonnement du comportement. Ce chapitre présente également une implémentation concrète du mécanisme de sélection d'action avec sept motivations.

Le chapitre cinq se focalise sur la partie utilisation du modèle. Il présente l'atelier de conception de comportement ainsi que des simulations permettant d'illustrer le fonctionnement du modèle proposé. L'atelier de conception de comportement est une interface destinée à un utilisateur expérimenté permettant une conception modulaire du comportement tout en n'ayant aucun a priori sur les simulations dans lesquelles le comportement sera utilisé. La partie des simulations permet de comprendre le fonctionnement du mécanisme de sélection d'action à travers quatre parties. La première partie présente l'effet, un à un, de chaque motivation sur le comportement de l'agent. La deuxième partie est l'illustration de l'influence d'un ensemble de motivations sur le comportement. La troisième partie montre un cas d'utilisation du modèle pour la conception de comportement basé sur les motivations afin de gérer des propriétés internes de l'agent. La dernière partie regroupe des expérimentations que j'ai menées afin de percevoir les utilisations possibles du modèle dans un cadre coopératif.

Les annexes A et B, présentent les différentes fonctions mathématiques que j'ai testées comme candidates potentielles pour être des fonctions de combinaison (en annexe A) des motivations et des préférences (en annexe B). Dans ces deux annexes, chaque fonction est critiquée par rapport à des critères précis attendus de la fonction de combinaison pour laquelle elle est candidate. Les deux fonctions retenues (la combinaison des motivations et la combinaison des préférences) ont été mises en place dans le mécanisme de sélection de CoCoA. L'annexe C est consacrée à la présentation en détail de l'atelier de conception. Cette annexe présente les différents écrans, les configurations possibles de l'atelier de conception et les fichiers générés permettant une conception modulaire du comportement.

Chapitre 1

Contexte

« Le commencement de toutes les sciences, c'est l'étonnement de ce que les choses sont ce qu'elles sont. »
Aristote

L'étude du comportement et sa modélisation est un sujet pluridisciplinaire. Ce sujet touche à la fois à la psychologie, la biologie et la chimie des organismes étudiés, l'économie, la sociologie et évidemment l'informatique. Nous nous limiterons à la présentation de travaux principalement en informatique permettant la compréhension de cette thèse. Nous aborderons dans un premier temps la thématique de la définition du comportement. Puis je présenterai des modélisations informatiques du comportement et je me focaliserai sur ceux qui ont été appliqués aux jeux vidéo et j'expliquerai la problématique liée aux jeux massivement multijoueurs. Enfin je présenterai l'approche centrée interaction qui est la base du projet CoCoA dans lequel j'ai travaillé.

1.1 Qu'est ce que le comportement ?

Avant de parler de la manière de modéliser un comportement, il faut d'abord s'entendre sur la définition du mot "comportement". Pour cela, j'utiliserai une définition basée sur l'éthologie et expliquerai en quoi il est difficile de pouvoir qualifier un comportement. Je présenterai également une approche psychologique du comportement et plus généralement des profils comportementaux.

1.1.1 La définition du comportement

Définition de l'éthologie

Dans [Set98], l'auteur donne une définition du comportement basée sur l'éthologie qui est l'étude du comportement et plus spécialement l'étude du comportement animal sauvage dans son milieu naturel. Le comportement est défini comme le résultat de l'observation de

l'ensemble des actions d'un agent dans son environnement. Cette définition met en jeux trois entités : l'agent, l'environnement et l'observateur. Le comportement existe, il est observable, mais on a souvent envie de le nommer, c'est-à-dire de vouloir l'identifier avec un concept par nominalisme. Le nominalisme est une doctrine de pensée fondée par Roscelin de Compiègne (1050-1121), opposée au réalisme de Platon, privilégiant ce qui est construit par l'observation et définissant la généralisation par concepts comme un simple outil de commodité pour la communication.

Identifier un comportement

Donner un nom à un comportement dépend de l'auteur c'est-à-dire de l'observateur du comportement. Qualifier un comportement est subjectif car il dépend de l'observateur, de sa culture et de ses connaissances. Par conséquent, donner un nom à un comportement est une interprétation personnelle de l'observateur sur les actions que l'agent a effectuées. En outre, l'introduction d'un observateur souligne le problème du sens que l'on veut donner au comportement lors de sa conception. L'interprétation d'une observation étant subjective, il est difficile de prétendre faire des comportements, de les nommer, de les montrer à des observateurs et de pouvoir obtenir l'unanimité que ce soit sur le nom utilisé ou sur l'interprétation que font les observateurs du comportement. Le problème se pose également, du fait que la notion de comportement d'un personnage ne vient pas seulement de l'observation des actions qu'il mène, mais aussi d'une appréciation qui est faite de sa personnalité. C'est, par l'observation des actions entreprises, que l'observateur construit sa propre appréciation du comportement et le classe en fonction de son évaluation : brutal, convivial, etc. C'est pourquoi, les termes utilisés dans ce document pour qualifier un comportement seront évidemment subjectifs et n'auront principalement pour but que de les identifier.

Les profils comportementaux

Dans [Don01] Jean-Paul Donckèle, décrit des portraits afin de mieux comprendre les étudiants et mieux communiquer avec eux. Dans ce livre, basé sur l'analyse transactionnelle, il décrit six profils : *sentimental*, *idéaliste*, *travailleur*, *ludique*, *songeur* et *entreprenant*. Chaque profil est présenté en fonction des attitudes adoptées, de ses réactions face à certaines situations, des modes de communication privilégiés et ceux évités. Ce que l'auteur décrit ce sont des profils comportementaux qui sont des caricatures de comportements. Il précise que les comportements humains sont tous composés de ces six profils dans des proportions différentes (certains sont dominants et d'autres sont presque inexistants).

Il est intéressant de pouvoir définir des comportements via certaines caractéristiques et de pouvoir doser chacune d'elles. Nous verrons plus tard que je prendrai cette idée afin de construire ce que j'appelle le profil d'individualité des agents.

1.1.2 Architectures et comportements

Cette thèse se place dans le cadre des systèmes multi-agents (SMA). Une qualification usuelle d'un agent dans les SMA est la distinction entre réactif et cognitif. Mais la notion de comportement de l'agent ne doit pas être confondue avec celle de l'architecture agent. Ainsi,

en parlant d'agent réactif [Nar98], cognitif ou même d'hybride [MP93] on ne détermine pas nécessairement son type de comportement.

Généralement, les comportements sont vus comme dépendant de facteurs internes comme la température, la faim, la fatigue et d'autres variables homéostatiques qui poussent l'agent à accomplir une action spécifique et surtout dans le cas des architectures réactives. Dans [Fer95] l'auteur précise que la principale différence entre un agent réactif et cognitif vient du fait que l'agent réactif se base sur la perception de son environnement pour agir, alors qu'un agent cognitif se base sur une représentation symbolique qu'il possède de son environnement. Ainsi le comportement réactif ne se limite pas à l'ensemble des actions qui sont effectuées par l'agent en réponse à des stimuli internes ou externes et le comportement cognitif à l'ensemble des actions planifiées qui sont effectués par l'agent dans le but de résoudre ses buts.

Il est possible avec des agents réactifs de construire des plans à moyen-terme sans représentation symbolique et avec des agents cognitifs de prendre en compte son voisinage proche pour prendre une décision. Dans mes travaux, j'ai utilisé les agents cognitifs de la plateforme CoCoA. Néanmoins, je montre qu'il est possible d'obtenir des comportements qui peuvent être considérés comme réactifs, tel que l'opportunisme.

1.1.3 Crédibilité du comportement

La notion de crédibilité

La crédibilité d'un comportement est une notion particulièrement importante. Dans le cadre des jeux vidéo, plus la crédibilité des comportements des personnages non joueurs est grande, plus elle entraine une immersion intense des joueurs. Toutefois il ne faut pas confondre crédibilité, réalisme ou rationalité. Le réalisme dans son sens commun (on ne parle pas ici du réalisme philosophique issu de l'idéalisme, ni du courant littéraire et artistique du XIXe siècle) est une tendance à décrire la réalité en ne masquant aucun aspect. Dans le cadre des jeux vidéo, le réalisme pur disparaît bien souvent en laissant place à la fiction. La rationalité introduit l'idée d'une correspondance entre la définition du comportement et l'exécution de celui-ci. La rationalité peut également être liée à la notion de comportement optimal par rapport aux buts, c'est-à-dire maximiser ses intérêts ou son bien-être. Nous préférons dans le cadre de cette thèse ne pas nous focaliser sur l'aspect d'optimalité du comportement. En effet, nous ne cherchons pas à atteindre un optimal en minimisant, par exemple, le nombre d'actions mais à produire des comportements crédibles. Pour être crédible, un comportement doit être en accord avec le contexte.

Dans sa thèse[Sep07] Cyril Septseault s'intéresse à cette notion de crédibilité du comportement des agents. Selon l'auteur, pour être crédible, le comportement d'un agent doit être consistant et cohérent. C'est-à-dire qu'il doit être le même devant des situations similaires sans pour autant avoir un aspect trop mécanique et il doit également s'adapter à la situation courante. Pour cela, le personnage doit appréhender son environnement : en avoir une bonne représentation et une bonne compréhension. Pourtant, l'auteur constate que généralement la représentation de l'environnement ne prend pas en compte la dynamique du monde, alors que cela permettrait aux agents d'anticiper les choses et d'avoir un comportement plus crédible. L'auteur propose donc un agent observateur qui n'a pas de comportement mais qui possède une représentation de son environnement qui reste cohérente en prenant en compte la dy-

namique de l'environnement. À partir de sa perception partielle de l'environnement, l'agent observateur va se construire une représentation de l'environnement. Il va tenter de comprendre la dynamique de son environnement afin de pouvoir prédire l'évolution des éléments qui ne se trouvent plus dans son champ de perception. Pour cela, l'agent observateur se décompose en cinq parties (voir la figure 1.1) :

1. La perception : ce système récupère les informations de l'environnement.
2. La représentation : cette partie stocke les informations provenant de la perception et contient les états potentiels des objets perçus.
3. La mise à jour de la représentation : cette partie tente de prédire l'évolution des états des objets en connaissant la dynamique des interactions subies par ces objets.
4. Le processus attentionnel : cette partie détermine les objets sur lesquels l'agent porte son attention afin de focaliser la mise à jour de la représentation sur ces objets ainsi que les objets qui pourraient être en interaction avec eux.
5. Le système d'adaptation : ce système met à jour les informations sur l'application des interactions (comme la fréquence ou la durée) en fonction des changements perçus par l'agent.

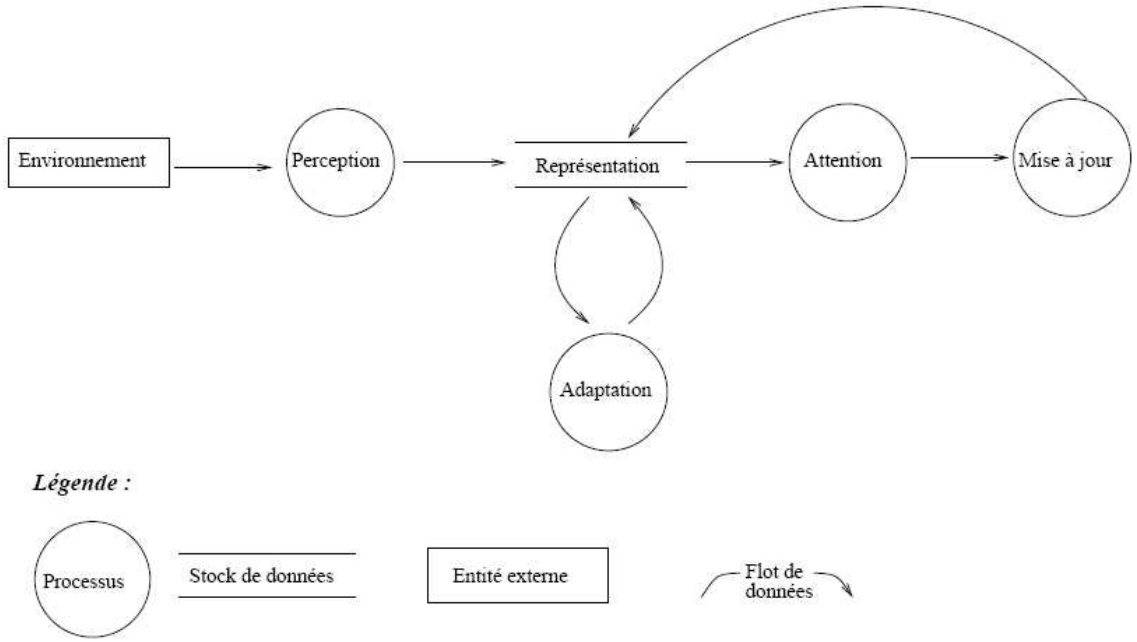


FIG. 1.1 – Architecture de l'agent observateur proposé par Cyril Septseault permettant le maintien de la représentation de l'environnement, présentée dans [Sep07].

Dans sa thèse, Cyril Septseault met en œuvre un agent voleur dont le but est de déterminer la ronde des gardiens afin de subtiliser un objet sans se faire repérer. L'environnement est généré aléatoirement, il contient plusieurs pièces et l'agent n'a pas de connaissance a priori de l'environnement.

Bien que ses travaux permettent une représentation plus crédible de l'environnement et d'augmenter par conséquent la crédibilité des personnages, deux points sont à souligner. Le premier est que concrètement l'environnement informé contient des informations supplémentaires dans le but de simplifier la mise à jour de la représentation et donc de guider le comportement de l'agent. Dans des environnements de grandes tailles et avec un plus grand nombre d'éléments participant à la dynamique de l'environnement, la question se pose sur le choix entre les informations qui doivent être portées par l'environnement (et qui doivent être valables pour tous les comportements de tous les agents) et les informations portées par l'agent. Le second, dans le cadre restreint de l'exemple du voleur développé dans cette thèse, les calculs pour la représentation monopolisent déjà beaucoup de ressources, comment est-il alors possible de gérer un nombre important d'informations et d'agents tout en permettant à chaque agent d'exprimer un comportement[Per09].

Comment valider un comportement ?

Comme présenté précédemment, l'interprétation que l'on peut faire d'un comportement observé est subjectif. Il serait donc hasardeux de prétendre dans cette thèse pouvoir réaliser des comportements répondant à une définition basée sur des qualificatifs qui peuvent être eux-même sujet à interprétation. Il est donc nécessaire de se fixer des critères pour évaluer un comportement. Ces critères peuvent être de différentes natures et se focaliser sur différents aspects de la conception du comportement. On peut se focaliser sur l'aspect pratique et donc sur l'utilisation du modèle : est-ce que la conception d'un comportement est simple à réaliser ? Est-ce que les différents éléments de la conception sont compréhensibles et maitrisables ? On peut également attendre d'un comportement certains résultats : le comportement de l'agent est-il défini indépendamment du contexte d'utilisation, est-il réutilisable ? Le comportement répond-t-il à un scénario prédéfini ?

1.2 La sélection d'action comme mécanisme définissant le comportement

1.2.1 Le comportement est une question de choix

Généralement lorsqu'on cherche à représenter un comportement on parle d'un choix qu'un agent aura à faire. Ce choix peut être influencé par plusieurs facteurs. Les mécanismes mis en place pour effectuer ce choix peuvent être de différentes natures (par exemple hiérarchique ou décentralisé).

La subsomption de Brooks

Dans [Bro85], l'auteur propose une architecture où chaque comportement est géré par un module. Tous les modules ont accès aux senseurs et aux effecteurs. Un module peut contenir un ensemble de conditions et d'actions mais celles-ci doivent gérer le même comportement spécifique. Ces modules appartiennent à une hiérarchie dans laquelle ils sont présentés verticalement (voir la figure 1.2). Dans cette hiérarchie les niveaux supérieurs sont prioritaires à

ceux des niveaux inférieurs. Lorsqu'il y a un conflit, les modules supérieurs peuvent inhiber la sortie des modules inférieurs. Chaque comportement précis (ou partie du comportement) de l'agent étant gérer par un module, il est facile d'en supprimer une partie en enlevant le module correspondant ou d'en ajouter en positionnement correctement le nouveau module. Dans cette architecture, la relation hiérarchique n'intervient qu'à la sortie, tous les modules ont accès aux informations des senseurs en même temps et produisent en parallèle les actions pour les effecteurs.

Bien que l'aspect modulaire de cette architecture simplifie la construction et la réutilisation de comportement, les critères sélection de comportement sont définis à travers la structure hiérarchique. La sélection d'action est donc un processus statique permettant de dérouler un scénario défini à travers cette hiérarchie.

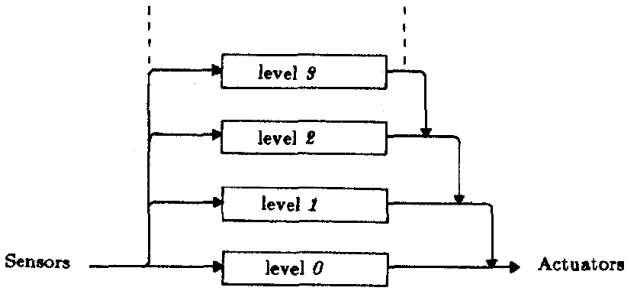


FIG. 1.2 – Architecture hiérarchique et modulaire de Brooks, présentée dans [Bro85].

La méthode DEM

Dans [Wit99] Mark Witkowski présente la méthode DEM (Dynamic Expectancy Model), permettant à un agent de résoudre ces buts en utilisant de l'apprentissage non supervisé et un mécanisme de sélection d'action. L'agent ne sait pas a priori comment résoudre ses buts et va l'apprendre par des tentatives successives. Pour cela, l'agent effectue des hypothèses prédisant les actions qu'il doit effectuer pour atteindre une situation à partir d'une situation courante. Chaque prédiction possède une mesure qui augmente lorsqu'elle s'avère être bonne (et qui diminue si elle s'avère être fausse). L'agent se focalise dans ses résolutions sur le but le plus prioritaire. Chaque but possède une priorité, celui qui possède la valeur de priorité la plus importante est le plus prioritaire. Pour cela, DEM construit un DPM (Dynamic Policy MAP) qui forme une séquence de liens entre les situations à atteindre (résolvant le but le plus prioritaire) avec les autres situations. Le DPM forme un graphe, où les nœuds sont les contextes et les arcs sont les actions. Chaque arc possède un coût qui dépend du coût unitaire de l'action (en temps ou en ressource) et de la mesure de la prédiction associée. Chaque nœud possède un niveau indiquant le nombre d'arcs à traverser pour atteindre le but le plus prioritaire (qui est de niveau 0). Le mécanisme de sélection d'action a pour rôle de sélectionner la meilleure action à exécuter. Pour cela, le mécanisme calcule à partir de chaque situation d'un même niveau n dans le DPM, la valeur de la politique. Cette valeur est la valeur minimale du trajet dans le DPM du niveau n vers le but le plus prioritaire en prenant en compte les coûts des actions et les niveaux des nœuds traversés. L'action à exécuter est l'action dont la valeur de la

politique est la plus faible à partir d'une situation active (une situation qui est actuellement vérifiée). La valeur des prédictions est mise à jour au cours de l'exécution par apprentissage, les coûts des actions sont donc mis à jour ainsi que les valeurs des politiques, afin de privilégier les actions de faibles coûts avec les prédictions les plus faibles.

La méthode DEM se base sur les priorités des buts et l'ensemble des actions à exécuter pour atteindre un but (ainsi que les coûts associés à ses actions) pour déterminer les actions à exécuter. Le mécanisme de sélection d'action présenté dans cette thèse se base sur des motivations et la notion d'alternative (i.e. les séquences d'actions à exécuter pour atteindre un but). Les motivations influencent le choix de l'action à exécuter mais peuvent être de différentes natures. De plus, tous les buts sont pris en compte simultanément. Nous verrons que le mécanisme de sélection d'action concret que je propose, prend en compte des motivations liées à la priorité des buts, les coûts des actions et d'autres motivations comme par exemple les préférences de l'agent. Enfin, contrairement à DEM, l'ASM que je propose permet à l'agent de se détourner temporairement d'un but prioritaire afin de profiter de la proximité d'une cible par opportunisme.

Les critères de sélection d'action selon Tyrrell

Les animats (pour la contraction de anima-materials) sont des robots animaux physiques (ou simulés par ordinateur) qui se comportent dans le monde réel (ou dans un monde simulé) d'une manière réaliste. Dans sa thèse de doctorat[Tyr93b], Toby Tyrrell compare plusieurs modèles (concrets ou seulement théoriques) de mécanismes de sélection d'action pour animats. Il en a mis en œuvre les modèles théoriques, les a testé et comparé, afin d'extraire des contraintes qui doivent être prises en compte dans l'évaluation d'un "bon" mécanisme de sélection d'action. Dans le cadre de cette thèse, nous nous focalisons sur des agents virtuels qui n'ont pas d'actionneurs physiques, ni de capteurs. La liste des critères défendus par Tyrrell qui restent applicables à des agents virtuels est présentée dans le tableau 1.1.

Dans ses travaux Tyrrell défend les architectures hiérarchiques à flux libres. En effet, les structures hiérarchiques classiques sont considérées comme trop rigides et peu robustes. Pourtant par nature, les mécanismes de sélection d'action sont hiérarchiques. Pour résoudre ce problème, Tyrrell propose d'utiliser une architecture hiérarchique à flux libres. Contrairement aux architectures hiérarchiques classiques, les architectures hiérarchiques à flux libres n'évaluent pas à chaque étape de la hiérarchie les actions à effectuer évitant ainsi une sélection du type "winner-takes-all". Ces architectures évaluent les actions en fin de hiérarchie afin de déterminer la meilleure action "candidate du compromis" lorsque l'on prend en compte toutes les contraintes ou motivations.

Dans [Tyr93a], Tyrrell montre que les architectures hiérarchiques à flux libres sont plus adaptées pour concevoir des mécanismes de sélection d'action que les architectures hiérarchiques classiques ou que les architectures distribuées[Mae90].

DAMN

DAMN (Distributed Architecture for Mobile Navigation) est défini par Rosenblatt[Ros97] comme une architecture distribuée où plusieurs modules gèrent le contrôle d'un robot. Chaque module est responsable d'un comportement individuel, il ne reçoit que les informations de

Critères de Tyrrell	**Définition**
Persistance jusqu'à l'achèvement d'une action	continuer jusqu'à l'achèvement d'une action, pour éviter le coût d'un changement d'action.
Activations proportionnelles à l'état courant	dans les systèmes homéostatiques, le besoin d'avoir des activations proportionnelles à l'état courant
Concurrence équilibrée entre les actions	les nœuds aidant à la réalisation d'un seul but ne doivent pas être dévalorisés par rapport aux nœuds achevant plusieurs buts
Continuité des séquences d'actions	besoin d'une autre persistance car changer de séquence d'actions a un coût élevé.
Candidat du compromis	l'action sélectionnée ne doit pas être la meilleure solution pour chaque sous-problème mais la meilleure sur l'ensemble des sous-problèmes simultanément
Interruptibilité si nécessaire	interrompre une séquence d'actions de priorité relativement faible pour en effectuer une de priorité plus importante.
Profiter de l'avantage qu'offrent des opportunités	permettre d'interrompre une séquence d'actions afin de profiter de l'avantage qu'offrent des opportunités.
Combinaison de préférences	prendre en compte des priorités de nœuds de hauts niveaux pour choisir parmi les nœuds de bas niveaux.
Combinaison flexible des stimuli	utiliser d'une fonction arbitraire pour combiner les valeurs des stimuli.

TAB. 1.1 – Les critères de Tyrrell pour un bon mécanisme de sélection d'action. Ces critères correspondantes aux critères de Tyrrell applicables à un agent virtuel.

l'environnement qui lui sont pertinents. Suivant ces informations chaque module donne son avis (allant de -1 à +1 : de contre à pour) sur les actions que le robot peut effectuer. Les modules travaillent de manière asynchrone et en parallèle. L'ensemble des avis est envoyé à un module d'arbitrage qui va combiner les avis pour évaluer globalement la meilleure action à effectuer. Chaque module possède un poids déterminant l'importance de son évaluation, il est possible d'utiliser dans DAMN un contrôleur dont le but est de mettre à jour ces poids suivant le contexte courant (voir la figure 1.3). La méthode de sélection permet de choisir la meilleure action en prenant en compte l'ensemble des évaluateurs (les comportements) et évite une sélection du type "winner-takes-all" (la sélection d'action est le résultat du compromis comme le préconise Tyrrell dans [Tyr93b]). Cette structure permet également de ne définir que le comportement du robot, via les comportements particuliers, sans se focaliser sur l'ensemble des situations possibles. Ainsi l'ajout et la suppression d'un comportement passe par l'ajout et la suppression d'un module, cet aspect modulaire rend la structure évolutive et les poids sur chaque module permettent de spécialiser le comportement du robot.

Le mécanisme de sélection d'action proposé par DAMN, permet de définir des modules votant pour ou contre une action. Le fait de décentraliser ces modules permet de simplifier la conception et de rendre le mécanisme plus évolutif. Dans cette thèse, je propose un mécanisme basé sur un principe similaire où des évaluateurs donneront leur avis sur les actions à effectuer de manière indépendante les uns des autres. Toutefois, je garde la possibilité pour

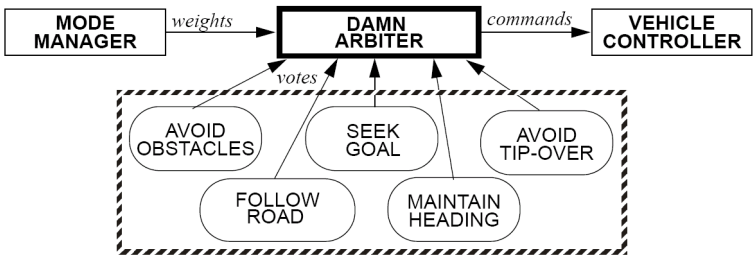


FIG. 1.3 – Architecture de DAMN avec les modules de comportements indépendants qui votent pour la meilleure action, présentée dans [Ros97].

ces évaluateurs d'inhiber les autres évaluateurs et d'avoir des seuils d'activation, en ce sens je me rapprocherai également des travaux de Blumberg dans [Blu94]

1.2.2 Les systèmes motivationnels

Selon le Grand Dictionnaire de la Psychologie une motivation est un "*processus physiologique et psychologique responsable du déclenchement, de l'entretien et de la cessation d'un comportement ainsi que de la valeur appétitive ou aversive conférée aux éléments du milieu sur lesquels s'exercent le comportement*". Pour Spencer A. Rathus [Rat95], "*les motivations sont définies comme des états hypothétiques au sein de l'organisme qui activent le comportement et poussent l'organisme vers un but*" et toujours selon lui "*le comportement des organismes est censé être en grande partie engendré par des motivations. Les besoins, les tendances et les incitateurs sont des concepts étroitement liés*". Dans [NDA08] l'auteur décrit la notion de motivation comme étant "*provoquée par un besoin physiologique (oxygène, nourriture, eau, ..) ou un besoin psychologique (accomplissement, pouvoir, estime de soi, approbation sociale et appartenance). Ces besoins donnent lieu aux tendances. Exemples : l'épanouissement de nourriture provoque une tendance de la faim. Être poussé à gravir les échelons professionnels.*". D'après [Clo05], "*du point de vue psychologique, la motivation correspond aux forces qui entraînent des comportements orientés vers un objectif, forces qui permettent de maintenir ces comportements jusqu'à ce que l'objectif soit atteint.*"

Globalement pour ces différentes définitions et dans la plupart des systèmes motivationnels, les motivations s'activent en fonction de propriétés internes et vont influencer le choix des actions à exécuter. La notion de but dans ce cas, se limite généralement au maintien des propriétés internes. Dans notre cas, une motivation se limite au moyen d'influencer le choix des actions à effectuer. Ce choix n'est qu'une partie de ce qui compose le comportement et nous verrons dans la suite que j'appelle cette partie du comportement **l'individualité**. Les actions à effectuer représentent les premiers pas de solutions possibles permettant de résoudre les buts de l'agent. Nous faisons donc intervenir les motivations beaucoup plus tard dans le mécanisme comportemental, c'est-à-dire après la recherche des résolutions possibles par la partie du comportement que j'appelle **le raisonnement**.

Tendances et Motivations

Si nous revenons sur la définition du comportement comme le résultat de l'ensemble des actions qu'un agent effectue dans un environnement, nous pouvons déduire deux éléments participant à sa conception. Le premier élément est composé des actions que l'agent peut effectuer. Si l'on restreint les capacités de l'agent, on restreint par conséquent son comportement. Par exemple, un agent qui ne pourrait que casser des portes pour les traverser pourrait être perçu comme plus brutal qu'un agent qui aurait la capacité de les ouvrir. Le deuxième élément est composé des actions que l'agent a effectuées. Si à un moment donné l'agent peut effectuer, parce que le contexte s'y prête bien, plusieurs actions et qu'il doit faire un choix pour n'en effectuer qu'une seule à la fois, alors il faut définir une manière de sélectionner la meilleure action à exécuter à un moment donné. Nous pouvons donc voir le comportement comme les actions exécutées par l'agent dans l'environnement, qui sont issues d'un choix parmi l'ensemble des actions qu'un agent peut effectuer à chaque instant. Ce choix étant un processus répondant à des contraintes diverses. Prenons par exemple *manger* : un humain peut préférer aller au restaurant plutôt que de cuisiner, car il est fatigué. Ou alors, il peut préférer manger des plats surgelés, car il n'a pas les moyens d'aller au restaurant. Ainsi, le comportement est orienté (de façon positive ou négative) par des **tendances**. La notion de tendances repose sur une théorie du comportement psychologique développée par Albert Burloud [Bur36]. En accord avec ce constat, nous proposons de définir notre vision du comportement comme les actions effectuées qui résultent d'un ensemble de tendances subies par l'agent.

Dans [Fer95] les tendances sont définies par les cognitons (i.e. particules élémentaires permettant de définir l'état mental d'un agent selon Ferber) qui poussent ou contraignent un agent à agir ou qui l'empêchent d'agir. Elles sont issues de combinaisons de motivations qui sont des cognitons plus élémentaires. Le fonctionnement de l'approche par motivation est expliqué au travers d'un système conatif proposé par l'auteur, ce dernier est présenté dans la figure 1.4. Dans ce système, les motivations forment la base de l'élaboration des tendances qui contraignent la décision de l'agent. L'auteur propose une classification des motivations en quatre catégories suivant les origines des motivations :

motivations personnelles : ce sont les motivations qui procurent un certain plaisir à l'agent. Par exemple, avoir faim est une motivation personnelle qui va pousser l'agent à chercher de la nourriture.

motivations provenant de l'environnement : ce sont les motivations qui proviennent d'éléments de l'environnement de l'agent. Par exemple, percevoir de la nourriture ou être à proximité d'elle, va pousser l'agent à manger.

motivations sociales : ce sont les motivations liées à la société qui poussent l'agent à agir d'une certaine façon. Par exemple, il est préférable d'avoir un travail et de bien le faire.

motivations relationnelles : ce sont les motivations provenant des autres agents. Par exemple un agent va en aider un autre à porter une charge qui est trop lourde pour un seul agent.

Chaque motivation peut orienter le comportement de l'agent selon les tendances qu'elle fournit. Une tendance allant de l'*attraction* à la *répulsion* va influencer le choix de l'ASM. Une attraction va pousser l'agent à réaliser une action alors qu'une répulsion va le freiner.

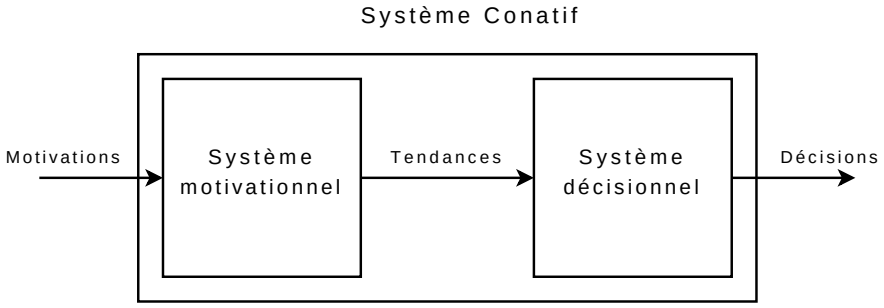


FIG. 1.4 – Le système conatif proposé par Ferber est composé de deux sous-systèmes : le système motivationnel qui élabore les tendances et le système décisionnel qui décide des actions à effectuer en fonction des tendances.

PECS

Schmidt propose dans PECS (*Physical conditions Emotional state Cognitive capabilities Social status*) un modèle pour représenter les comportements humains[Sch05]. Ce modèle a été fait pour remplacer le modèle BDI (Believe Desire Intention) dans le cadre de la conception de comportement humain. L'architecture de PECS se veut être universellement applicable avec une adaptation à l'individualité simple. Les agents sont capables d'exprimer des comportements différents en ayant la même structure profonde et peuvent être décrits par le même modèle de référence. La structure du monde est représentée à l'aide de trois entités : l'environnement, un connecteur et les agents.

L'environnement représente tous les facteurs externes pouvant influencer le comportement de l'agent (ce qui peut également inclure les autres agents). Chaque agent PECS possède une représentation de cet environnement qui peut être incomplète, incertaine et même erronée.

Le connecteur est un composant gérant les échanges d'informations entre les agents.

Les agents PECS (voir la figure 1.5) sont composés d'éléments dont l'organisation est décomposée en trois couches horizontales : les entrées, les états internes et les sorties.

- Les entrées sont composées des modules de perception et senseur. Le module senseur prend les informations provenant de l'environnement, le module de perception traite et filtre les informations pour les mémoriser.
- Les états internes gèrent le comportement de l'agent et sont composés de quatre modules : le module social, le module cognitif, le module émotionnel et le module physique. Chaque module possède des propriétés (notées Z) représentant des propriétés internes de l'agent et des fonctions (notés F) gérant la mise à jour de ces propriétés. Le module physique gère l'aspect physique de l'agent (par exemple ses variables homéostatiques comme la température), le module émotionnel gère les émotions de l'agent (tristesse, colère), le module social gère la collaboration entre les agents et le module cognitif gère les connaissances de l'agent.
- Les sorties sont gérées par les modules de comportement et d'action. Le module de comportement choisit les actions à effectuer (il permet d'utiliser notamment la planification ou l'apprentissage) en fonction des états internes. Le module d'action réalise l'action

choisie. Les actions sont de deux types, les actions internes (qui se font sur l'agent lui-même) et les actions externes qui se font dans l'environnement.

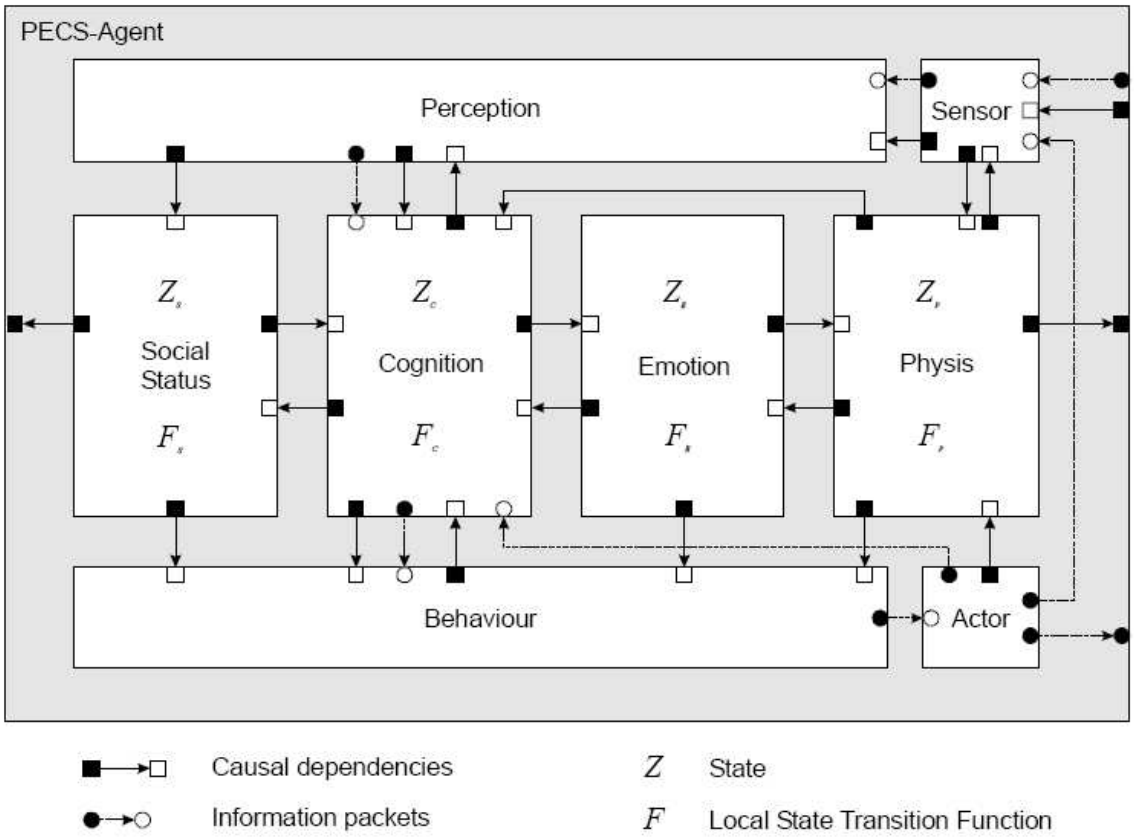


FIG. 1.5 – Architecture d'un agent PECS présentée dans [Sch05].

Dans PECS, il est possible de mettre en place des motivations ("*motives*") qui vont influencer le comportement de l'agent. L'auteur définit quatre types de motivations se distinguant par leurs constructions et leurs origines (la pulsion, l'intensité émotionnelle, la volonté et le désir social). L'auteur propose une sélection d'action basée sur un winner-takes-all. À chaque motivation correspond une intensité, les motivations sont en concurrence entre-elles et celle qui a la plus forte intensité détermine l'action à exécuter et donc le comportement de l'agent. Les intensités sont calculées suivant les variables internes de l'agent et leurs valeurs changent au cours de l'exécution. Par exemple, la faim pilote le comportement : "aller *au réfrigérateur*". La méthodologie proposée fonctionne en quatre étapes (voir le tableau 1.2).

Il est intéressant pour des agents purement cognitifs qui sont classiquement dirigés par les buts de prendre en compte ces propriétés internes. En effet, un comportement réaliste devrait pourvoir être influencé par des propriétés internes telles que son niveau d'énergie ou son niveau de faim. La conception de comportement que je proposerai prendra en compte les propriétés de l'agent. Dans l'implémentation que je ferai pour CoCoA, j'ajouterai la notion de propriétés dynamiques permettant notamment de représenter des propriétés homéostatiques.

Étapes	Définition
1	Calculer les nouvelles valeurs de variables d'état internes.
2	Calculer l'intensité de chaque motivation correspondante.
3	Comparer et sélectionner la motivation avec l'intensité la plus importante.
4	Effectuer l'action défendue par la motivation choisie.

TAB. 1.2 – Fonctionnement en quatre étapes du modèle PECS.

Motivations et hormones

Dans [CAG07, Cañ97] les auteurs présentent un mécanisme de sélection d'action basé sur les motivations et les émotions. Dans cette proposition chaque agent possède des propriétés internes dont les valeurs doivent être maintenues entre certaines bornes prédéfinies. Un agent possède également des comportements (i.e. des actions qu'il peut effectuer) qui ont un effet sur ses propriétés internes (le comportement manger de la nourriture augmente la valeur de la propriété sucre dans le sang). Une motivation est un élément qui pousse l'agent à sélectionner un comportement, sa tendance est calculée suivant des stimuli internes (ses propriétés) et externes (l'environnement). Une fois les tendances calculées, le mécanisme de sélection d'action choisit la motivation ayant obtenu la plus grande valeur. Puis, il choisit le comportement actif (pouvant être exécuté) satisfaisant le plus le besoin de l'agent lié à cette motivation. Si aucun comportement ne permet de satisfaire le besoin, le mécanisme sélectionne un comportement d'exploration permettant d'activer un comportement satisfaisant le besoin (notions d'appétence et d'exploration en psychologie). De plus, les auteurs utilisent la notion d'hormone dont la quantité peut faire varier la valeur des stimuli (internes et externes) et des motivations. La quantité d'hormones est définie en fonction des émotions de l'agent et de son état interne, ce qui influence le choix des comportements.

Dans cette approche, la sélection d'action s'effectue en fonction de la motivation la plus importante. Mon mécanisme de sélection d'action choisit l'action à exécuter en fonction de plusieurs motivations, cette action est un compromis entre toutes ces motivations. Enfin, contrairement à cette approche, les buts de mes agents ne seront pas uniquement focalisés sur le maintien de propriétés internes. Les motivations que je propose ne sont pas des buts à satisfaire mais des éléments influençant le comportement de l'agent.

1.2.3 Motivations et hiérarchie

Certains travaux proposent une sélection d'action où les motivations sont organisées dans une hiérarchie. Je présenterai deux travaux proposant une hiérarchisation des motivations basée sur une étude du comportement humain[Mas98] de Maslow, un psychologue américain.

Hiérarchie des besoins de Maslow

Maslow[Mas98] propose une représentation des besoins des humains connue comme "la pyramide des besoins de Maslow" (voir la figure 1.6). En économie, l'utilisation de cette théorie défend l'idée que si les besoins primaires des salariés sont satisfaits, ils auront alors de nouveaux besoins plus complexes comme ceux liés à l'accomplissement personnel et notamment l'accomplissement au travail, ce qui les poussera à être plus performants. Dans cette théorie les

besoins humains sont organisés dans une structure hiérarchique organisée en cinq couches selon la priorité des besoins. Dans la couche inférieure de cette pyramide sont contenues les besoins les plus prioritaires et les plus primaires : les besoins physiologiques et biologiques (comme manger ou respirer). La couche supérieure est composée des besoins les moins prioritaires qui sont également les plus complexes, les besoins de réalisation de soi. Cette théorie repose sur la satisfaction des besoins. Une fois que les besoins primaires sont satisfaits, d'autres besoins plus complexes et moins prioritaires apparaissent. Ainsi, lorsque les besoins d'une couche sont entièrement satisfaits, un individu cherche à satisfaire des besoins plus complexes provenant de la couche supérieure.

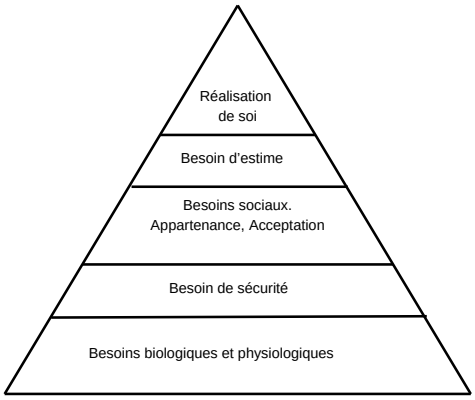


FIG. 1.6 – Pyramide des besoins de Maslow.

Le système MASLOW défendu par Andriamasinoro

Dans [And03], l'auteur nous présente un modèle d'agents hybrides basé sur la motivation naturelle. Ce modèle fonctionne pour des agents réactifs et des agents cognitifs. Par motivation naturelle, l'auteur veut décrire les besoins qui font agir l'agent. Le modèle d'agent proposé, nommé MASLOW (pour Multi-Agent System based on Low-Needs), est composé de trois éléments : la pyramide des besoins (ou des motivations) nommée Π, un réseau d'actions Ω et un mécanisme nommé NIM (Need Importance Manager).

La pyramide des besoins Π est basée sur la pyramide des besoins de Maslow[Mas54]. Comme la pyramide de Maslow, l'auteur propose pour chaque besoin d'attribuer un niveau (correspondant au niveau dans la pyramide) représentant l'importance du besoin. Mais, l'auteur définit également pour chaque besoin une catégorie. Cette catégorie définit l'importance des besoins appartenant à un même niveau.

Chaque besoin est également défini par une liste d'états possibles du besoin (qui peuvent être par exemple : *satisfait*, *bas* ou *insatisfait*) ainsi qu'une liste d'actions à effectuer pour satisfaire chaque état du besoin correspondant (sauf dans le cas de l'état satisfait qui ne requiert pas d'action car le besoin est satisfait). Un besoin peut être de trois natures différentes :LN,MN,HN.

- LN ou Low Needs représentent les besoins innés qui sont communs à tous les agents et qui ne dépendent pas de l'application (par exemple la faim).

- MN ou Medium Needs correspondent aux besoins répondant immédiatement aux LN (par exemple le besoin d'aller au restaurant).
- HN ou High Needs font référence aux besoins d'anticiper la satisfaction des LN (par exemple le besoin d'aller au travail pour gagner de l'argent qui permettra de se payer de la nourriture).

Les MN et HN sont des besoins liés à l'application contrairement aux LN qui sont génériques.

Le réseau d'actions Ω est un graphe dont les nœuds sont des actions (primitives ou composées) et dont les connecteurs permettent de définir les liens entre les différentes actions (par exemples la non possibilité d'exécuter deux actions simultanément, l'inclusion d'une action dans une action complexe ou la relation de succession entre deux actions). Une action est définie par le besoin que l'action satisfait et la liste des actions qui ne peuvent pas être exécutées simultanément. Un besoin doit être satisfait pour effectuer l'action (Le besoin est vide si l'action est toujours exécutable). Une action peut être une action primitive (PR) élémentaire et non-interruptible ou une action complexe (AC) composée d'actions primitives ou d'AC.

Le mécanisme NIM utilise un algorithme qui détermine les actions à exécuter. Pour cela, NIM collecte toutes les actions (PR ou AC) que l'agent peut effectuer à travers Π et Ω pour en déduire l'ensemble des actions primitives que l'agent peut exécuter. Ce mécanisme n'est pas accessible à l'utilisateur, il fonctionne de manière générique en respectant la manière dont l'utilisateur a défini Π et Ω. En cas de conflit entre les actions (des actions qui ne peuvent pas s'exécuter simultanément), l'algorithme va faire un choix en fonction des besoins correspondant, en privilégiant (filtrant) les besoins non satisfaits, les besoins les plus proches de la satisfaction des LN (MN>HN), le niveau du besoin, la catégorie du besoin, l'état du besoin et le ratio du besoin (la position par rapport à l'état le plus bas du besoin). En cas d'égalité de ces filtres un choix aléatoire est effectué.

Cette proposition permet de définir des comportements dirigés par des motivations naturelles (ou besoins). L'utilisateur ne doit pas coder les comportements, il doit uniquement définir la pyramide des besoins Π et le graphe des actions Ω afin que NIM puisse calculer automatiquement les actions à exécuter à chaque cycle. Selon l'auteur, la conception de la pyramide Π peut être simplifiée en s'appuyant sur des critères provenant d'études scientifiques (par exemple en biologie ou en sociologie suivant ce qu'on cherche à modéliser). La construction de Ω dans le cadre d'un comportement complexe peut s'avérer une tâche difficile et notamment, comme le déclare l'auteur, lors de la définition des relations entre les actions. Dans cette proposition, l'état d'un besoin est lié à une sémantique particulière et peut être représenté soit par une valeur (booléenne ou numérique), soit par un intervalle de valeurs numériques. Nous verrons que je propose une notion assez similaire à l'état d'un besoin qui s'appelle le prototype d'un profil d'individualité qui fait correspondre à chaque motivation, une valeur ou un intervalle de valeurs définissant l'expression plus ou moins importante d'une motivation. Ces valeurs pouvant également être définies via des propriétés de l'agent qui évoluent pendant la simulation (comme les besoins).

Le système de classeurs hiérarchiques

Dans [dS06] l'auteur propose un mécanisme de sélection d'action basé sur des classeurs hiérarchiques pour les personnages virtuels dans un monde persistant (comme les jeux vidéo).

Le principe d'un classeur classique est de produire des actions répondant à certaines conditions en respectant des règles prédéfinies. Les actions sont internes (mises à jour des propriétés de l'agent) ou externes (des actions à effectuer dans l'environnement). Par contre, un système de classeurs hiérarchiques (HCS pour Hierarchical Classifier System) sépare les règles qui vont générer des actions internes des règles qui vont générer des actions externes, à l'aide de deux classeurs (voir la figure 1.7). Dans un HCS les règles produisant des actions nécessitent une condition liée à l'état interne et une condition liée à la perception pour être activées. Les règles sont plus simples à définir (que dans un classeur classique) et la recherche des règles à activer est plus efficace. Bien que nécessitant plus de règles que le précédent, l'organisation hiérarchique d'un HCS permet de définir des sous-problèmes (de tailles inférieures) et ainsi de diminuer l'espace de recherche.

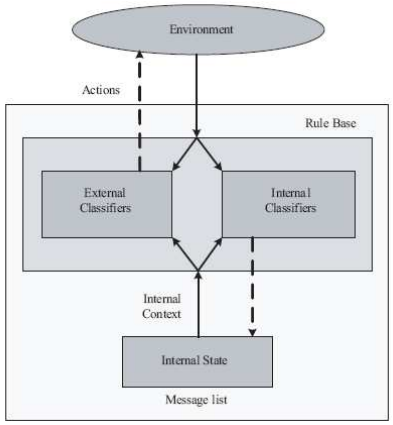


FIG. 1.7 – Le système de classifieur hiérarchique dans [dS06].

Le comportement de l'humain virtuel est influencé par des motivations dont le but est de répondre à ses besoins, c'est-à-dire d'assurer la stabilité de ses propriétés homéostatiques. Ces motivations sont regroupées suivant trois catégories (dont la priorité est décroissante) : basique, essentielle et secondaire (les motivations basiques étant les plus prioritaires). Cette organisation s'inspire sur la pyramide des besoins de Maslow[Mas54]. Le modèle de sélection d'action motivationnel proposé par l'auteur se décompose en quatre parties.

- Les **variables internes** qui représentent les variables homéostatiques de l'humain virtuel.
- Les **motivations** qui vont produire des buts à satisfaire, suivant les informations provenant de l'environnement, des variables internes et un cycle d'hystérésis.
- Les **comportements orientés buts** qui représentent le contexte interne du HCS et qui sont utilisés pour planifier les séquences d'actions.
- Les **actions** qui sont séparées en deux catégories, les *actions motivées* qui vont satisfaire une ou plusieurs motivations et les *actions intermédiaires* qui sont les actions rendant possible l'exécution d'une action motivée.

Les motivations sont évaluées suivant la valeur de l'état interne correspondant. Cette évaluation est faite via deux seuils définissant trois zones : la zone de confort, la zone de tolérance et la zone critique. Dans la zone de confort l'agent ne prendra pas en compte la motivation, dans la zone de tolérance la motivation aura le poids correspondant à la valeur de la variable

interne et dans la zone de danger la valeur de la motivation est amplifiée par rapport à sa variable interne. Les actions peuvent également avoir un seuil d'activation (suivant le seuil de la motivation correspondante), ce dernier seuil pouvant être modulé en fonction des préférences de l'agent.

La sélection de l'action à exécuter dépend de l'évaluation de la motivation, du cycle d'hystérésis, des informations de l'environnement, du contexte interne du HCS, du poids du classeur, des préférences de l'agent et des autres motivations. L'auteur a mis en place une hiérarchie à flux libres sélectionnant la meilleure action suivant différents critères (la candidate du compromis pour Tyrrell).

1.3 Comportements et Jeux vidéo

1.3.1 La conception de comportement dans les jeux vidéo

Les jeux vidéo : leur succès est une contrainte

Depuis plusieurs années, l'industrie du jeu vidéo ne cesse de se développer. À titre de comparaison, depuis 2002, le chiffre d'affaires généré par l'industrie du jeu vidéo a dépassé celui de l'industrie du cinéma.

En 2008, les chiffres de vente des jeux vidéo ont dépassé les 33 milliards d'euros, dont 13.9 milliards pour l'Europe, 9.7 milliards pour le Japon et 17.1 milliards pour les États-Unis (d'après l'IDATE : l'Institut de l'audiovisuel et des télécommunications en Europe). En France à cette même période cette industrie aurait employé plus de 10 000 personnes avec plus de 430 entreprises (selon l'agence française pour le jeu vidéo). Le marché du jeu vidéo comprend cinq grandes parties.

- le marché des consoles de salon
- le marché des consoles portables
- le marché des logiciels pour ordinateur
- le marché des logiciels pour consoles de salon
- le marché des logiciels pour consoles de portables

Le premier marché est celui des logiciels de consoles de salon (avec plus 16 milliards d'euros dans le monde en 2008). Fin juillet 2009, il s'est vendu 52.6 millions de Nintendo Wii, 30 millions de Xbox 360 et 24 millions de PS3. Aujourd'hui, il faut également prendre en compte le marché des logiciels sur mobiles qui sont également en pleine expansion.

En 2007, le secrétariat d'État à la prospective, à l'évaluation des politiques publiques et au développement de l'économie numérique, a mis au point le plan France numérique 2012. Ce plan de développement de l'économie numérique prévoit notamment de développer le secteur du jeu vidéo (point 2.6 du rapport datant d'octobre 2008).

Ainsi, le marché du jeu vidéo prospère, il est soutenu par le gouvernement et de plus en plus de projets d'innovations sont réalisés dans ce domaine. La concurrence dans ce marché est donc très importante et c'est pourquoi il est souvent difficile d'avoir des informations concrètes sur le fonctionnement d'un jeu vidéo. Ceci est particulièrement vrai pour l'aspect intelligence artificielle de ce domaine qui est un point de comparaison entre les différents produits de plus en plus important. C'est pourquoi, dans mes recherches sur ce qui se fait du coté industriel, je

me suis focalisé sur les informations publiées qui ne sont malheureusement pas nombreuses et sur ce qui peut être déduit du fonctionnement d'un jeu par rapport au constat que l'on peut faire en tant qu'utilisateur.

Les Scripts

Lorsque l'on étudie comment est codée l'intelligence artificielle dans les jeux vidéo deux principales techniques sont employées pour leur facilité d'application : les scripts et les automates à états finis. Un script, du point du vue du comportement, est la description des actions à effectuer selon certaines situations prévues. L'exécution d'un script est linéaire, le comportement est préprogrammé, il n'a pas de raisonnement pour déterminer l'action à exécuter. Les inconvénients des scripts sont bien connus[Toz02], les comportements ne sont pas assez variés. Il est possible de déterminer le comportement proposé par un script sans trop de difficulté (puisque le comportement n'évolue pas) ce qui lasse rapidement les joueurs et il n'est pas possible de s'adapter à un évènement non prévu dans le script.

Des solutions ont été apportées pour contourner les défauts de cette approche et créer des scripts dynamiques [PS04, SPSKP06]. Les scripts dynamiques sont généralement construits à partir d'une sélection d'un ensemble de règles. Chaque règle possède un poids correspondant à son efficacité. Les scripts sont construits à partir des règles de plus haut poids pour chaque partie du comportement que l'on souhaite concevoir (la collecte de ressources, la survie, ... etc). Par apprentissage, il est possible de modifier les poids des règles en fonction de l'efficacité du script en cours. Il est également possible d'affiner cette création de script par l'utilisation d'un algorithme évolutionnaire.

Bien que les scripts soient souvent utilisés pour leur simplicité (*NeverWinter Nights, Morrowind, Unreal Tournament*), leur utilisation oblige à définir un comportement qui est très dépendant du contexte d'exécution, ce qui limite leur réutilisation d'un jeu à l'autre.

Les automates à états finis

Les automates ou machines à états finis ou FSM (Finite State Machines), comme les scripts, sont fréquemment utilisés pour leur simplicité d'application (*Dune II, Quake, Warcraft III, F.E.A.R.*). Généralement, une machine à états finis est composée d'un nombre fini d'états, de transitions entre ces états et d'actions. Chaque état contient une ou plusieurs actions que l'agent doit exécuter (voir figure 1.8). Une transition représente les conditions nécessaires au passage d'un état à un autre. Généralement, un FSM possède un état d'entrée qui démarre le comportement et zéro, un ou plusieurs état terminaux déterminant la fin du comportement.

Les machines à états finis hiérarchiques ou HFSM (Hierarchical Finite State Machines), sont des FSM dont certains états et transitions sont regroupés à l'intérieur d'un état. Par exemple, si l'on reprend le FSM de la figure 1.8, on se rend compte que lorsque l'énergie est faible, l'agent arrête ce qu'il fait pour récupérer de l'énergie, qui est une action plus prioritaire. On peut alors regrouper les autres actions à l'intérieur d'un état pour définir les autres actions qui ne sont pas liées à l'énergie de l'agent (voir la figure 1.9). Le HFSM nécessite dans ce cas, une pile d'états pour retourner à l'état précédent, une fois l'énergie récupérée.

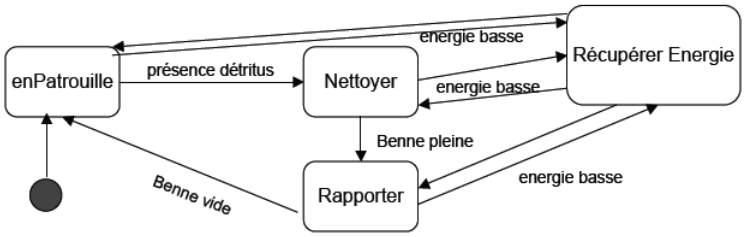


FIG. 1.8 – Exemple de FSM, tiré du cours "Programmation orientée agents" de Jacques Ferber, Université de Montpellier II.

Le principal intérêt des HFSM est l'aspect modulaire de l'architecture qui regroupe dans un état les différentes parties d'un comportement. Définir un comportement revient alors à relier les modules par des transitions. Ils ont été notamment utilisés pour *Destroy All Humans 2* et *Halo 2*.

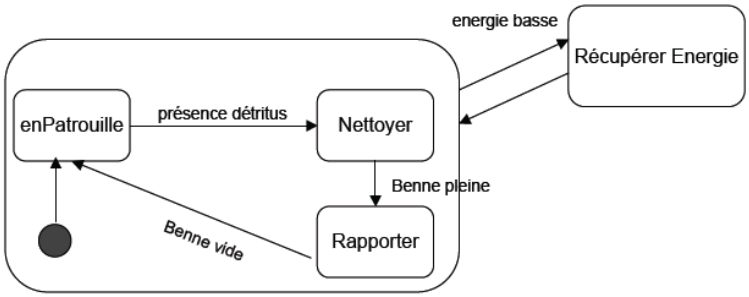


FIG. 1.9 – Exemple d'une machine à états finis hiérarchique reprenant l'exemple de la figure 1.8, tiré du cours "Programmation orientée agents" de Jacques Ferber, Université de Montpellier II.

Mascaret

Dans l'étude des comportements dans les jeux vidéo, je me suis intéressé à un Serious Game. Un Serious Game est un logiciel dont le but n'est pas de distraire le joueur. La plupart des serious game ont comme but d'apporter un contenu sérieux (pédagogique, informatif, marketing) dans un contexte ludique issu d'un jeu vidéo. MASCARET (Multi-Agent Systems to simulate Collaborative, Adaptative and Realistic Environnements for Training)[Che06, BQLC03] est un modèle dont le but est de former des personnes à travers une simulation (que l'on peut qualifier de serious game) dans un environnement virtuel réaliste, collaboratif et adaptatif. Ce modèle est basé sur les concepts d'agent, d'organisation, de rôle et d'élément de comportement (voir la figure 1.10). Le rôle correspond aux responsabilités de l'agent dans l'organisation. La notion d'organisation permet de structurer les interactions entre les agents, de mettre à un

agent de connaître ces partenaires et son rôle dans la collaboration. Les agents ont donc un comportement lié à l'organisation à laquelle ils appartiennent.

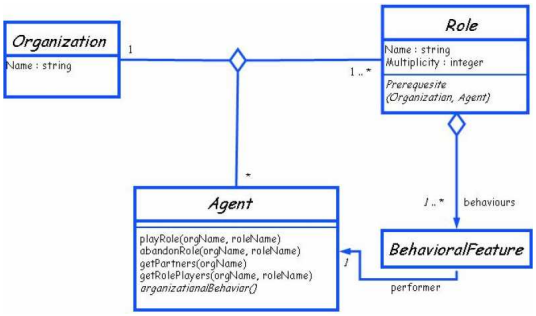


FIG. 1.10 – Modèle générique de Mascaret basé sur les notions d'agent, d'organisation, de rôle et d'élément de comportement, présenté dans [BQLC03].

Ce modèle générique (abstrait) se concrétise suivant les différentes organisations de la simulation : physique, social, etc. Les phénomènes physiques (définis grâce à une concrétisation du modèle générique) sont représentés par des agents réactifs qui peuvent être désactivés selon les besoins de la simulation. Ces agents réactifs ont un comportement simple qui est déterminé en fonction de leurs états et de facteurs externes. L'environnement possède également des agents rationnels qui subissent et agissent comme les agents réactifs. Ces agents appartiennent à une équipe (une organisation sociale définie dans une concrétisation du modèle générique) permettant la coordination des actions. Ces agents déterminent les actions à effectuer en fonction des autres agents et des procédures permettant la réalisation de la mission. Les personnes en formation sont représentées par des avatars qui correspondent aux agents rationnels (la seule différence vient de l'inhibition de l'envoi de message pour l'exécution d'une action afin de permettre aux utilisateurs d'effectuer des choix pour leur formation). Ces utilisateurs peuvent ainsi interagir avec l'environnement et les autres agents. SÉCURÉVI (SÉCUrité et RÉalité VIrtuelle) est une application de MASCARET à la sécurité civile. Elle permet d'aider à la formation des officiers sapeurs-pompiers. Cette application permet de simuler une intervention avec les phénomènes physiques (feu, fumée, jet d'eau...). Les apprenants jouent les rôles des différents chefs de groupes intervenant lors de l'incident et le formateur participe à la simulation pour provoquer des dysfonctionnements, aider les apprenants ou jouer un rôle dans une équipe.

Ce modèle élaboré pour la formation permet de définir le comportement d'un agent à travers son rôle dans une organisation. Néanmoins, le choix de l'exécution de l'agent ne prend pas en compte des facteurs personnels permettant de définir une individualité propre à l'agent.

Halo 2

Dans [Isl05], Damian Isla présente comment la complexité de l'IA du jeu Halo 2 a été gérée. Halo 2 est un jeu de tir à la première personne (First Person Shooter ou FPS) développé par Bungie Software et édité en 2004. Dans ce jeu, l'IA est conçue avec un HFSM. Dans les HFSM, pour déterminer quel état atteindre parmi l'ensemble des états inclus dans un état parent, il

est généralement commun d'utiliser des priorités. Soit les priorités sont définies dans l'état parent, soit ce sont les états fils qui fournissent une évaluation de leur pertinence. L'auteur précise que dans le cas où il y a plus de vingt états, ce système devient trop gourmand. C'est pourquoi, dans Halo 2 et lorsque le nombre d'états fils est trop important, ces états sont organisés dans une file suivant leur priorité qui est fixe. Comme il est possible, qu'une action A soit plus prioritaire qu'une action B et que suivant le contexte, l'action B peut être plus prioritaire que l'action A, l'auteur a mis en place des **impulsions de comportement**. Une impulsion est un trigger lié au contexte dont la priorité n'est pas fixe et qui va désigner une action particulière (comme un pointeur d'action). Ainsi l'action B peut être mise en avant grâce à une impulsion de priorité dynamique qui se trouvera devant l'action A (ou derrière suivant le contexte). La partie contextuelle d'application est ainsi sortie des comportements pour réduire la complexité en temps de calcul des actions à effectuer. Ces impulsions peuvent également servir à rediriger l'exécution vers une autre partie du comportement par exemple lorsque la survie du personnage est en jeu ou pour ajouter à certains moments des animations ou des sons. De plus, le gestionnaire d'évènement peut dynamiquement ajouter des impulsions pour gérer des comportements (appelés comportement stimulus) très spécifiques liés à un évènement particulier (comme la fuite si le chef du groupe est mort).

Enfin, l'auteur présente comment pour Halo 2, le travail du concepteur de comportements a été simplifié. Le but était que le concepteur n'ait pas à tout spécifier mais qu'il précise à gros grains quel comportement adopter (agressif, lâche, ... etc). Pour cela, deux notions ont été mises en place : la notion d'ordre qui est une référence à une position de tir pour un groupe et la notion de style qui est un ensemble de comportements autorisés ou rejetés. Ces deux mécanismes permettent d'utiliser la même IA mais de faire agir différemment suivant la progression dans le jeu. Autre simplification, la personnalisation des personnages est paramétrable, mais si une personnalisation est constituée de 3 paramètres, qu'il y a 115 comportements et 30 types de caractères différents, cela fait 10350 nombres à maintenir. C'est pour cela que les caractères et les paramètres de comportement sont contenus dans un fichier. Ce fichier de caractères est un bloc logique regroupant certains aspects du comportement. Les fichiers de caractères possèdent une relation de spécialisation (ainsi lorsqu'un fichier de caractères ne possède pas les données, une recherche est effectuée parmi ses parents). Un caractère générique a été défini pour être à la racine de l'arbre de spécialisation.

F.E.A.R.

Dans [Ork06], l'auteur présente les machines à états finis et l'algorithme A^* comme étant certainement les deux techniques les plus communément employées lors de la conception de l'intelligence artificielle pour les jeux vidéo.

L'algorithme A^*[BF81] est un algorithme de recherche de chemin entre deux nœuds dans un graphe. L'algorithme A^* va chercher à minimiser le coût total du déplacement à effectuer pour se rendre d'un nœud initial vers un nœud final. Pour cela, A^* utilise une heuristique minorante qui évalue le coût d'un déplacement d'un nœud vers un autre pour calculer la solution optimale.

L'auteur présente la construction de l'IA des PNJ (Personnages Non Joueurs) dans un FPS nommé *F.E.A.R.* (First Encounter Assault Recon) développé par Monolith Productions et sorti en 2005. Pour cela, Jeff Orkin utilise une machine à états finis et un A^* de manière

non conventionnelle, le FSM ne contient que trois états et le A^* est utilisé pour planifier les déplacements et les séquences d'actions à exécuter. Notons que Orkin présente la difficulté et la complexité d'utiliser les FSM bien qu'elles soient communément utilisées dans la conception de jeux, c'est pourquoi il en propose une utilisation particulière. Selon l'auteur, puisque l'IA que l'on cherche à construire est si "intelligente", alors elle doit être capable de faire des choses par elle-même et ainsi simplifier la tâche du programmeur. Ainsi, alors que les FSM contiennent généralement tous les états du monde et les conditions de transitions entre les états, celui utilisé dans F.E.A.R. ne contient que trois états `Goto`, `Animate` et `UseSmartObject`. L'état `UseSmartObject` est utilisé pour les animations, le comportement se résume donc à deux états `Goto` et `Animate`. De plus la logique déterminant les transitions d'un état à un autre n'est pas contenue dans le FSM mais elle est gérée par un système de planification (i.e. un processus qui recherche les séquences d'actions pour satisfaire un but) inspiré par STRIPS[FN71]. Le FSM gère donc les animations et le système de planification gère le comportement du PNJ. L'avantage est qu'un FSM classique contient toutes les actions à exécuter pour chaque situation, qu'il faut explicitement définir, alors qu'un système de planification requiert uniquement les actions et les buts de l'agent pour construire la séquence d'actions à exécuter pour satisfaire les buts. Un FSM est procédural alors que le système de planification est déclaratif.

L'utilisation du système de planification allège donc la charge de travail du concepteur et permet de définir les comportements des PNJ en fonction des actions qu'ils peuvent exécuter et des buts qu'ils doivent résoudre. Ainsi deux PNJ avec les mêmes buts, mais différentes actions se comportent différemment même s'ils se trouvent exactement dans le même niveau (environnement).

Un coût est associé à chaque action, ce qui permet d'utiliser un A^* pour sélectionner la meilleure séquence d'actions à exécuter. Dans cette utilisation du A^* proposée par Orkin, les arcs sont les actions, les nœuds sont les états du monde et le coût de déplacement correspond au coût d'une action. Le calcul du coût par action dépend de nombreux facteurs et situations présents dans F.E.A.R., cette tâche n'est donc pas facilitée. Le A^* est donc utilisé à la fois pour la gestion des déplacements et pour le choix des actions à exécuter. La séparation du déclaratif et du procédural ne s'effectue que sur une des deux parties du comportement (la construction des séquences d'actions exécutables par l'agent). Pourtant il est dommage que ce qui a été fait pour simplifier la définition des actions que l'agent peut faire, ne soit pas appliqué à la définition de ce que l'agent choisit de faire.

Creatures

Le jeu *Créatures*[GCM97], développé par Creature Labs, est sorti en 1996. Le principe de ce jeu est d'éduquer des créatures virtuelles appelées des Norns (Cyberlifogenis cutis) dans une environnement limité (voir figure 1.11). Cette éducation se fait par apprentissage par renforcement : par punitions et récompenses. Pour cela le joueur interagit avec les créatures à l'aide du curseur de la souris pour la chatouiller (renforcement positif) ou pour la frapper (renforcement négatif). Le comportement des Norns est déterminé par un réseau de neurones artificiels. Ce réseau est sous-divisé en "lobes" qui détermine les caractéristiques électriques, chimiques et morphologiques d'un groupe de cellules (neurones). Chaque neurone peut être connecté avec un ou plusieurs neurones appartenant aux autres lobes via des synapses. Cette structure se veut être biologiquement plausible, elle est formée d'approximativement 1000 neurones regrou-

pés dans 9 lobes interconnectés à travers 5000 synapses. Chaque Norn est également composé d'un système biochimique qui permet de simuler des fonctions endocriniennes (gérant le système hormonal) ainsi que certains aspects du métabolisme et un système immunitaire simple. Certaines caractéristiques structurelles et fonctionnelles des Norns sont déterminées par leurs gênes, le génome étant représenté par un seul chromosome haploïde (pas de paire, les humains ont des chromosomes diploïdes qui vont par paire). Les Norns se reproduisent, les phénomènes d'enjambement (crossing-over) se produisent, ainsi que les "erreurs" d'omissions, de duplications et de mutations aléatoires de gênes afin de favoriser le brassage génétique. Les Norns apprennent, la lignée évolue et certains comportements sociaux sont parfois exprimés (comme la coopération). Les réseaux de neurones sont rarement utilisés dans un jeux commercial du fait de leur paramétrage important.

FIG. 1.11 – Image-Ecran du jeu Creature tirée du site officiel http ://www.gamewaredevelopment.co.uk/.

The Sims

Le jeu *The Sims*, développé par Maxis, est sorti en 2000. Le principe de ce jeu est de gérer la vie d'un personnage virtuel dans un environnement simulant la réalité (voir la figure 1.12). Le personnage évolue dans un environnement restreint à son habitation et il est en interaction avec d'autres Sims du même quartier (ses voisins). Pour satisfaire ses besoins un Sims effectue des actions sur des objets ou d'autres Sims. L'utilisateur personnalise les éléments du jeu (les objets et les décorations) en fonction d'une quantité d'argent virtuel que le Sims gagne en travaillant chaque jour. Le comportement d'un Sims est dirigé par ses propriétés internes qu'il doit satisfaire (faim, énergie, confort, etc.). Le joueur peut également ordonner au Sims

d'effectuer des actions afin d'orienter l'évolution de la simulation. Le troisième opus des Sims est sorti en 2009, toujours développé par Maxis. Les possibilités de personnalisation sont accrues et l'intelligence artificielle gère les besoins naturelles du Sims afin de permettre au joueur de se concentrer sur les aspects sociaux comme la carrière ou les relations avec les autres Sims. Les Sims sont plus autonomes et leur comportement personnalisable à l'aide des traits de personnalité qu'il faut choisir lors de la création du personnage (par exemple maladroit, hypersensible, artiste ou travailleur).

FIG. 1.12 – Image-Ecran du jeu The Sims.

Black and White

Dans le jeu vidéo *Black and White*, développé par Lionhead Studios et édité en 2001 par Electronic Arts, le joueur doit éduquer une créature en utilisant un apprentissage par renforcement par punitions et récompenses. La créature répond également à des désirs liés à des propriétés internes comme la faim, la douleur, la colère, les dégâts subis. Dans [Eva02], l'auteur nous présente comment il est possible d'obtenir une créature comme celle du jeu *Black and White*. Il définit que l'on doit utiliser plusieurs techniques d'apprentissage pour couvrir toutes les capacités de l'agent. Par exemple, la créature apprend quel objet satisfait le mieux une motivation avec un arbre de décision et détermine les poids de chacun de ses désirs en utilisant un perceptron.

Pour éduquer sa créature, le joueur doit, en fonction de la dernière action que la créature a exécutée, soit la gratter ou la caresser pour la récompenser, soit la gifler pour la punir (l'apprentissage avec le bâton et la carotte pour Richard Evans). Ainsi la créature apprenait

les bonnes actions à faire (ou les mauvaises actions à ne pas faire). Du point de vue du joueur, ce processus d'apprentissage n'est pas évident à utiliser et les récompenses données à tort (par exemple, une caresse, lorsque l'animal a mangé l'un des serviteurs du joueur) sont très difficiles à corriger.

FIG. 1.13 – Image-Ecran du jeu Black and White 2, où l'on voit (en bas) le menu représentant pour chaque action les préférences de la créature. Ses préférences peuvent être changées à tout moment.

D'ailleurs dans *Black and White 2* (également développé par Lionhead Studios et édité par Electronic Arts en 2005), les valeurs de récompense pour chaque action de la créature sont clairement identifiées et le joueur peut à tout moment modifier complètement ses valeurs pendant le jeu (voir la figure 1.13). Ainsi, dans ce deuxième opus, l'impact de la phase d'apprentissage a été considérablement réduit.

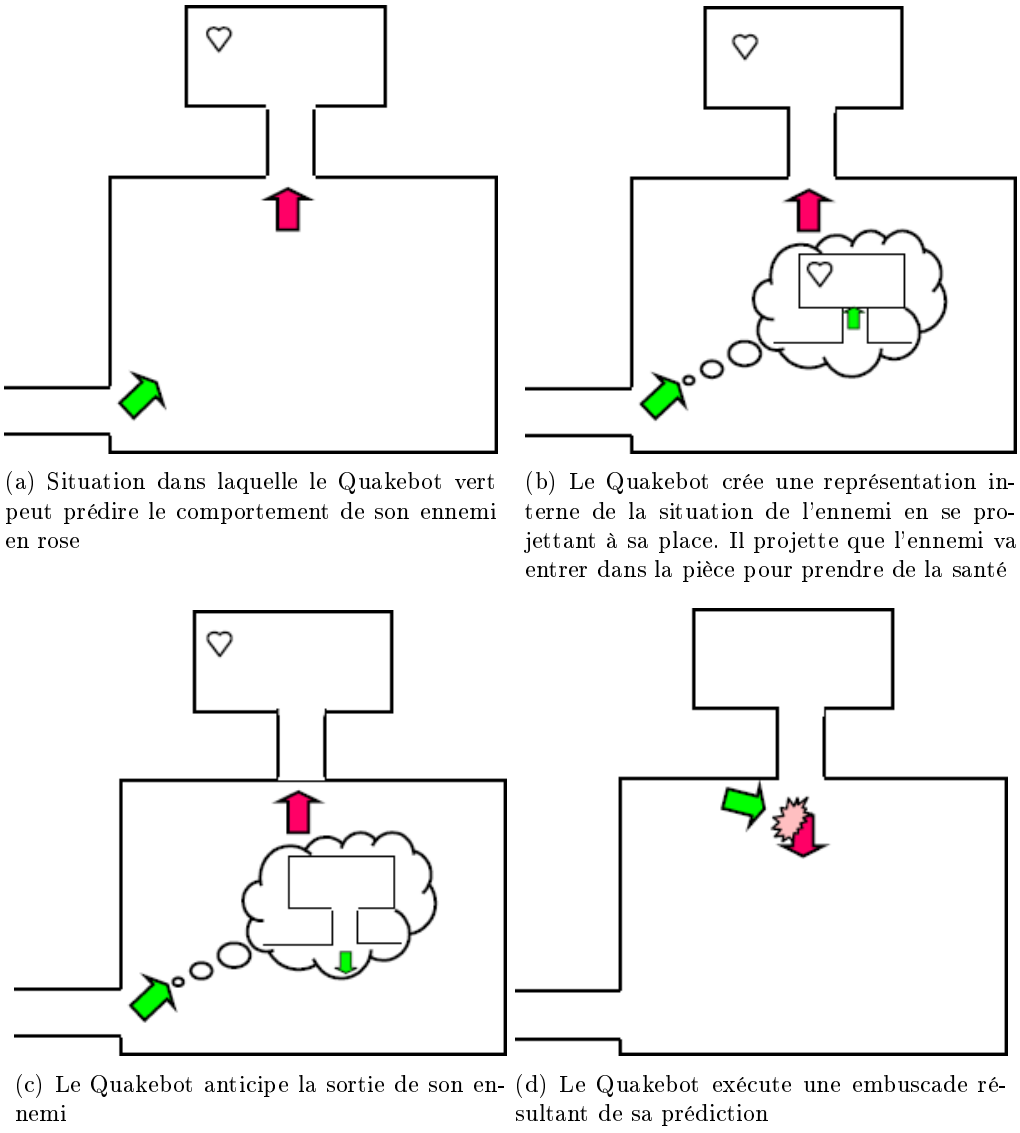
Si l'utilisation de techniques d'apprentissage est une tâche complexe pour un joueur, il en va de même pour un concepteur de jeux vidéo. En cela je rejoins [HYH09], quand ils disent :" *The developer not only needs to know how they want the character to behave, but also the AI principles required to make that behavior emerge*".

Quakebot

Dans [Lai01], John Laird propose de créer des bots qui ont la capacité d'anticiper les actions des adversaires. Ces bots sont conçus pour *Quake 2* (développé par id Software et distribué par Activision en 1997) qui est un FPS (First-Person Shooter) et pour une partie de Deathmatch où celui qui gagne est celui qui a tué le plus d'adversaires. Dans ce jeu, les terrains de jeu ne sont pas de grandes tailles (le but n'étant pas d'isoler les joueurs, mais qu'ils s'entretuent) et l'environnement n'est pas dynamique : tous les bonus (armes, armures, santé ou munitions) ont un emplacement prédéfini et une fois l'objet pris, ce dernier revient à la même place au bout d'un temps spécifique (défini dans les règles du jeu). Les grands joueurs de FPS connaissent très bien les cartes du jeu, l'emplacement des bonus, le temps avant qu'ils réapparaissent et savent anticiper le comportement des autres joueurs. Par exemple, si un joueur va dans une pièce pour prendre un bonus, il est possible de lui tendre une embuscade en se positionnant de telle manière qu'à sa sortie de la pièce on puisse lui tirer sur le flanc ou dans le dos. C'est exactement le type de comportement que l'auteur souhaite obtenir avec son Quakebot (voir figure 1.14)

L'auteur propose d'utiliser SOAR[LNR87] pour gérer le comportement du quakebot. SOAR utilise des règles de production de la forme `si ... alors ...`. Le principe de fonctionnement de SOAR est de balayer les possibilités (l'espace de problème) à partir du contexte courant (la phase d'élaboration) d'après les informations de la mémoire de travail (suivant le principe d'un chaînage avant qui part du contexte courant pour définir toutes les actions possibles jusqu'à atteindre un contexte résolvant un but) pour ensuite évaluer par pondérations les différentes possibilités pour décider de l'action à exécuter (la phase de décision). Lorsque la phase de décision ne peut pas déterminer l'action à effectuer (bien souvent cela vient du manque d'informations sur son environnement) un sous-but est construit pour lever cette impasse par une *méthode-faible*. SOAR peut utiliser de nombreuses méthodes faibles (comme le min-MAX). Une fois l'impasse levée par l'une des méthodes faibles, SOAR crée une règle par la phase d'apprentissage nommée `chunking` et la garde en mémoire. Ainsi si le bot se retrouve dans une situation semblable, il n'aura qu'à appliquer la règle et la situation ne sera plus considérée comme une impasse.

Le quakebot perçoit son environnement et mémorise sa topologie, l'emplacement de tous les objets et plus particulièrement les objets qui sont recherchés par les joueurs (les bonus) dans une représentation interne. Il est possible de réutiliser une représentation de l'environnement plusieurs fois pour gagner en performance. L'auteur précise que son quakebot peut être utilisé dans un environnement inconnu, mais il a besoin d'un laps de temps pour mémoriser l'emplacement des objets et sa topologie pour ainsi améliorer son comportement. De plus, l'environnement n'est pas très grand ni très dynamique, ce qui limite les impasses. Dans d'autres types de jeux, le nombre d'acteurs, la dynamique de l'environnement et donc le nombre de situations d'impasses est beaucoup plus important. Dans le cas de jeux en ligne qui sont des applications en constante évolution, cette apprentissage devrait se faire à chaque mise à jour (même partiellement). Enfin, bien que la prédiction permette de définir des comportements complexes comme une embuscade, l'activation de cette dernière dépend selon l'auteur de certaines situations qu'il faut définir au préalable.



(a) Situation dans laquelle le Quakebot vert peut prédire le comportement de son ennemi en rose

(b) Le Quakebot crée une représentation interne de la situation de l'ennemi en se projettant à sa place. Il projette que l'ennemi va entrer dans la pièce pour prendre de la santé

(c) Le Quakebot anticipe la sortie de son ennemi

(d) Le Quakebot exécute une embuscade résultant de sa prédiction

FIG. 1.14 – Le quakebot en bas à gauche (a,b,c) sait que son ennemi va chercher de la santé dans la pièce au nord. Il anticipe donc son comportement (a,b,c) afin de pouvoir lui tendre une embuscade (d).

MHiCS

Gabriel Robert et Agnès Guillot présentent dans [RG06] MHiCS (Motivational and HIerarchical with Classifier Systems) une architecture motivationnelle et hiérarchique à base de systèmes de classeurs pour la conception de personnages non joueurs dans les jeux vidéo. Le terme hiérarchique désignant l'architecture en quatre niveaux (voir la figure 1.15). Le niveau 1 est celui des motivations qui expriment des besoins internes à satisfaire. Le niveau 2 est constitué de systèmes de classeurs (CS) à base de règles, chaque système de classeurs (et donc les règles également) est spécifique à une motivation. Les règles sont de la forme *Si Condition Alors Action*. Ces règles rendent activables les actions présentes au niveau 3, qui deviendront

exécutables en fonction des ressources d'actions disponibles situées au niveau 4. Comme [dS06] cette architecture est dite hiérarchique à flux libres (qui est la meilleure selon Tyrrell [Tyr93b]).

Les motivations sont définies par un *facteur de personnalité* (FP) décrivant les préférences de l'agent pour ses motivations et la *valeur motivationnelle* (VM) définissant le niveau de la motivation en fonction de l'état du PNJ. La valeur d'une motivation, son *intensité* (IM), est calculée par le produit de FP et VM. Les CS à base de règles sont spécifiques à une motivation. Ainsi chaque CS recherche dans un espace restreint augmentant l'efficacité du système. De plus lors de l'ajout de motivations la modularité de l'architecture évite de réécrire totalement la base de règles. À chaque règle est associée une force déterminant une relation de priorité entre les règles. Chaque CS va déterminer les actions activables (AC) selon la partie condition des règles. Pour chaque AC, il va ensuite déterminer son intensité d'activation en faisant le produit de l'intensité de la motivation (IM) et de la valeur de la plus grande force pour l'AC. L'exécution des actions se détermine en fonction de l'intensité de l'action activable afin d'attribuer à chaque ressource l'action avec la plus grande intensité possible pour que l'allocation des ressources atteigne l'optimal (avec une notion de bruit permettant d'effectuer des actions proches de l'optimal). Afin d'ajuster les forces des règles à l'environnement, les auteurs utilisent un apprentissage en ligne évaluant une règle pendant son activation en fonction de la valeur motivationnelle VM, donnant ainsi plus de poids aux règles qui font diminuer une motivation.

MHiCS a été appliquée à Team Fortress Classic (TFC) une extension multi-joueurs pour le jeu Half-Life et plus spécifiquement pour une partie de Capture The Flag (CTF). Dans un CTF, deux équipes s'affrontent pour récupérer le drapeau de l'équipe ennemie qui est situé à l'autre bout de la carte. Ce jeu est un FPS et l'environnement est connu et n'est pas de grande taille, les bonus (et les drapeaux) ont une place prédéfinie. Le nombre de motivations est faible, seulement trois motivations (le score individuel, le score collectif et la survie du PNJ) et les ressources d'actions se limitent au déplacement, à la visée et au tir.

De cette proposition, il est intéressant de retenir la définition de l'intensité d'une motivation qui est calculée en fonction du facteur de personnalité et de la valeur motivationnelle. L'idée de représenter la personnalité d'un agent, c'est-à-dire une interprétation des motivations, ressemble à l'idée que je propose de mettre en place un profil d'individualité qui détermine l'expression des motivations de l'agent. Enfin, même si les règles restent simples à concevoir, dans un environnement très dynamique avec de nombreuses possibilités d'actions ou de contextes différents, leur réalisation peut s'avérer être une lourde tâche [Ork06].

Being-in-the-world

Dans [DZ01], les auteurs proposent un agent (nommé being-in-the-world) capable de jouer à un jeu du type MUD (Multi-User-Dungeons). L'agent a pour but de survivre dans l'environnement, c'est-à-dire se déplacer dans l'environnement, maintenir sa santé, éviter la faim, interagir avec des joueurs humains ou des PNJ. Il doit également comme les joueurs, collecter des ressources pour avoir un meilleur équipement et acquérir de l'expérience afin de devenir plus puissant, tout cela en temps réel.

L'agent being-in-the-world fonctionne dans le jeu ScryMUD qui est un MUD en mode texte sous licence GPL. Le jeu a été modifié pour offrir une interface visuelle et pour avoir des

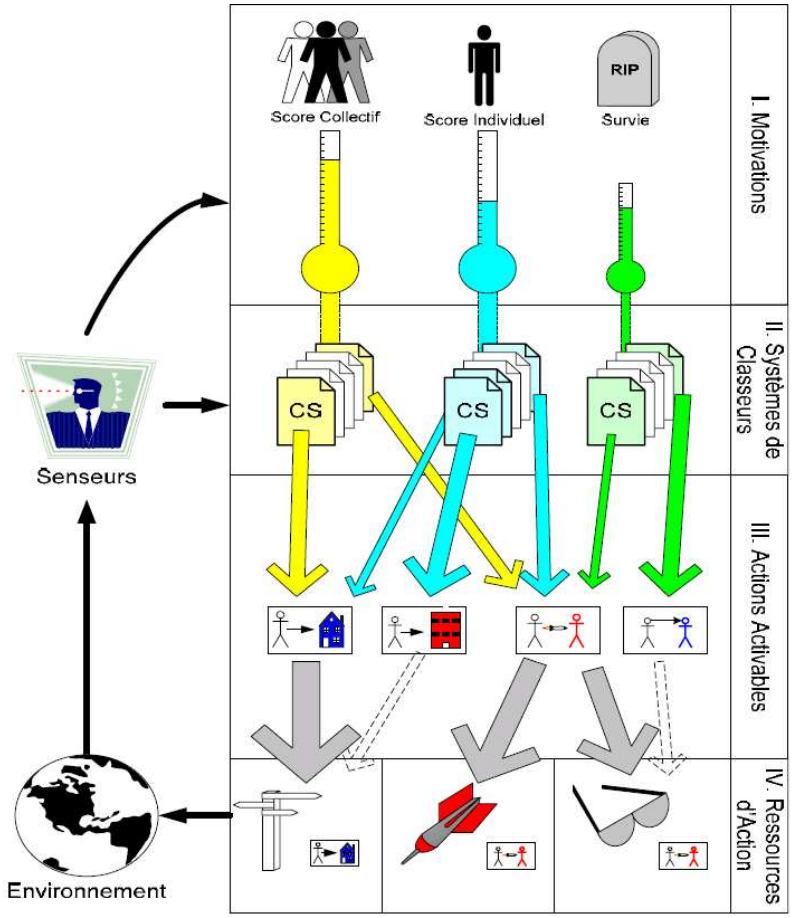


FIG. 1.15 – Architecture de MHiCS avec les motivations et les ressources d'actions pour Team Fortress Classic présentée dans [RG06].

identifiants sur les objets et les propriétés du jeu. Toutefois, l'agent communique avec le jeu comme les autres joueurs.

L'architecture de l'agent est composée de deux modules : Descartes et Heidegger. Descartes est le module de raisonnement basé sur un système de maintenance de la vérité (ou TMS pour truth maintenance system), qui inclut l'état interne de l'agent, la compréhension de l'environnement et la gestion des buts. Ce module décompose des buts de haut niveau (comme le but d'acquérir un item particulier) en buts immédiats (comme tuer une créature X) en effectuant un chaînage arrière (pour acquérir un item il faut de l'or, pour avoir de l'or il faut tuer et voler les créatures, la créature la plus faible est la créature X) selon les connaissances et l'état de l'agent. Ces buts sont envoyés au module Heidegger qui tentera de les réaliser. Heidegger est le module gérant la réactivité de l'agent (important pour l'aspect temps réel), c'est-à-dire les interactions entre l'agent et l'environnement (acquisition et mémorisation d'informations provenant de l'environnement et réalisation d'action) en tentant de satisfaire les buts immédiats. Heidegger gère la réalisation des actions à accomplir dans l'environnement et dans une situation critique (une attaque par exemple) il décide de l'action à effectuer sans prendre en compte les directives du module Descartes. Heidegger met également à jour la mémoire de

l'agent (l'ontologie du monde) qui est utilisée par le module Descartes. Les deux modules travaillent de manière asynchrone et assez indépendamment, ils communiquent via une file de buts et une ontologie partagée du monde.

La séparation entre le raisonnement et le choix des actions à effectuer permet à l'agent de planifier les actions sur le long terme et d'agir immédiatement selon le contexte courant. Toutefois, cette proposition pose deux contraintes, la première est la gestion limitée des informations mémorisées.L'environnement n'est pas très dynamique et sa topologie est connue a priori, l'agent ne construit pas sa propre représentation de l'état de l'environnement. La deuxième contrainte vient de la communication entre les deux modules qui se fait via une file. Comme les auteurs l'expliquent la gestion du changement de la valeur de véracité d'une précondition dans cette file ne permet pas de réintroduire à nouveau le but. Il est dommage que les auteurs ne précisent pas comment Heiddeger choisit entre deux buts ou même comment il choisit entre deux actions (ou séquences d'actions) permettant de résoudre un même but.

1.3.2 Les interfaces de conception pour les jeux vidéo

Si le modèle utilisé est important, l'interface de conception ne doit pas être négligée. Ces interfaces vont permettre d'alléger la tâche du concepteur suivant le public visé : novice, averti ou expérimenté. Je présenterai donc dans cette partie quelques travaux dont l'intérêt réside principalement dans l'interface de conception, plus que sur le modèle utilisé.

Behaviorshop

Dans [HYH09] les auteurs proposent BehaviorShop, une interface graphique basée sur l'architecture de subsomption pour la conception de personnages interactifs. Leur but est de concevoir une interface intuitive et utilisable par des novices en intelligence artificielle. En effet, les auteurs font le constat qu'aujourd'hui pour construire l'IA d'un personnage il faut à la fois se concentrer sur comment on désire que le personnage agisse et avoir des connaissances en IA pour pouvoir réaliser le comportement.

Initialement, leur interface était basée sur un automate à état finis (FSM), mais la plupart des utilisateurs trouvaient cette approche trop complexe et pas intuitive. Ils ont donc décidé d'abandonner l'idée d'utiliser des FSM ou des HFSM alors qu'ils sont généralement utilisés dans l'industrie du jeu vidéo. Les auteurs ont donc privilégié l'utilisation d'une architecture de subsomption qui est plus simple à prendre en main par des novices, cette architecture étant statique et hiérarchique.

Dans BehaviorShop, l'utilisateur conçoit le comportement de son agent (son rôle) pour un environnement précis. Il peut, lors de la création d'une couche, avoir accès à la carte de l'environnement pour spécifier par exemple des déplacements précis. Pour cela il navigue principalement entre deux écrans, un écran résumant l'architecture avec les différentes couches et un écran spécifique par couche (voir la figure 1.16). Dans ce dernier écran, le comportement d'une couche est défini par un trigger (une condition d'activation de la couche) et une action à exécuter. La couche inférieure de l'architecture possède le trigger "Always" signalant que cette couche doit toujours être active (c'est le comportement par défaut). Pour l'utilisateur, une couche est présentée comme une phrase : *si le [rôle de l'agent] [condition du trigger], alors*

le [rôle de l'agent] fera [action]. Le *[rôle de l'agent]* est prédéfini, c'est le rôle que l'utilisateur est en train de construire. La partie *[condition du trigger]* et *[action]* sont des choix dans une liste à partir d'éléments provenant d'une ontologie de comportement. Certains choix peuvent être combinés à d'autres éléments via des opérateurs logiques. Par exemple, la condition du trigger peut être la conjonction de deux états dont le premier dépend d'un personne et le second d'une distance par rapport à l'agent.

Pour valider leur résultat, les auteurs ont demandé à 10 personnes âgés de 18 à 24 ans de créer 3 rôles. Le premier, un gardien de sécurité qui gère l'entrée dans un bâtiment. Le second rôle est une personne autorisée à entrer après avoir montrer son badge. Le troisième rôle est une personne non autorisée qui doit essayer plusieurs fois d'entrer dans le bâtiment. Au final 8 personnes ont réussi à concevoir des rôles répondant aux critères (3 ont réussi à concevoir totalement le résultat désiré). Les résultats de cette étude montrent qu'avec BehaviorShop, la conception de comportement semble être plus simple. Néanmoins, bien que la hiérarchie de subsomption soit modulaire, les comportements conçus sont très spécifiques au contexte d'application ce qui en limite leur réutilisation. De plus, l'utilisation de BehaviorShop permet de simplifier la tâche pour des novices, mais lorsqu'il faut créer un véritable jeu avec un nombre important de rôles différents, cette architecture peut rendre la tâche plus fastidieuse. Il est donc important de cibler les utilisateurs d'une interface pour en fournir une qui soit la plus adaptée possible. Dans cette thèse, je présenterai un atelier de conception permettant la création de comportements utilisables dans CoCoA. Pour son interface nous avons ciblé les utilisateurs avertis ou expérimentés. L'interface est donc moins intuitive mais elle permet notamment de créer des comportements indépendamment du contexte d'exécution.

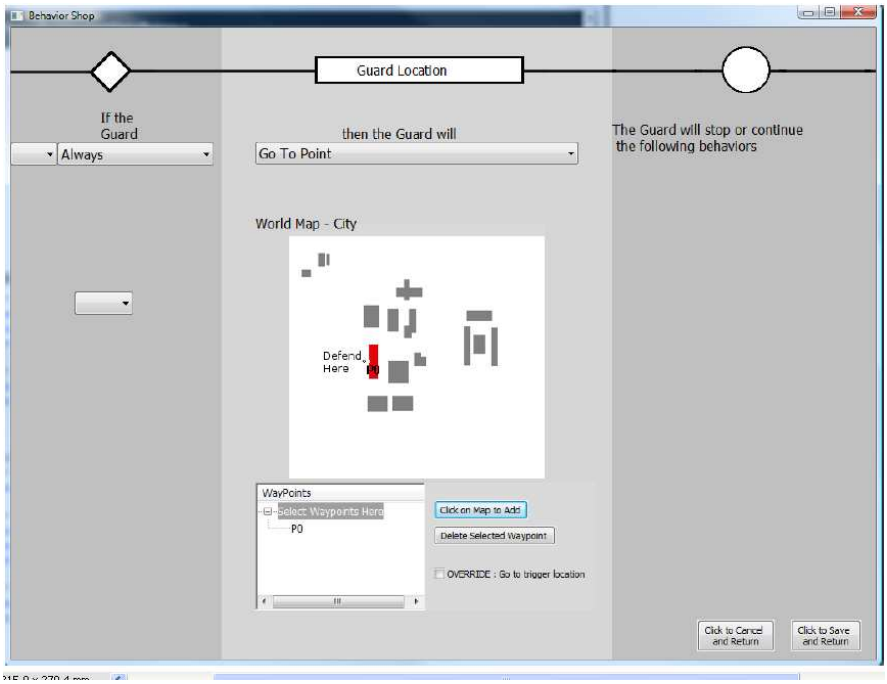


FIG. 1.16 – Fenêtre de BehaviorShop gérant la création d'une couche, présentée dans [HYH09].

SpirOps

SpirOps est un outil commercial permettant la création des comportements des personnages dans les jeux vidéo qui contient notamment un éditeur (voir la figure 1.17), un générateur de code (en C++) et un débugger. Cet outil est basé sur les travaux de recherche d'Axel Buendia [Bue05] et a été utilisé dans le jeu *Splinter Cell - Double Agent* d'Ubisoft. SpirOps fait l'objet d'un brevet depuis 2003, les informations sur son fonctionnement ne sont pas accessibles facilement. Le principe dans SpirOps est que l'utilisateur puissent concevoir des comportements en ce concentrant sur chaque problème indépendamment et non plus du point de vue des situations "si je suis dans tel état et que je suis à telle position alors je fais ça". En effet, plus le nombre de situations est important, plus le comportement est complexe mais également plus la réalisation est difficile. En se focalisant sur les problèmes et les manières de résoudre ces problèmes SpirOps permet de soulager la tâche du concepteur. Chaque problème (appelé centre d'intérêt) est géré indépendamment et SpirOps s'occupe de gérer les interférences entre les problèmes s'il y en a. Un centre d'intérêt est défini par des actions (des solutions possibles aux problèmes) et des percepteurs (des éléments de l'environnement utilisés dans le raisonnement). A partir d'un centre d'intérêt trois paramètres sont calculés : la motivation, l'opportunité et les paramètres d'action. La motivation étant l'importance du centre d'intérêt par rapport à l'état de l'agent, l'opportunité est le fait de savoir s'il est déjà possible de résoudre le centre d'intérêt en fonction de ce qu'il y a dans l'environnement et des paramètres d'action qui influencent le choix des actions comme la personnalité ou les émotions.

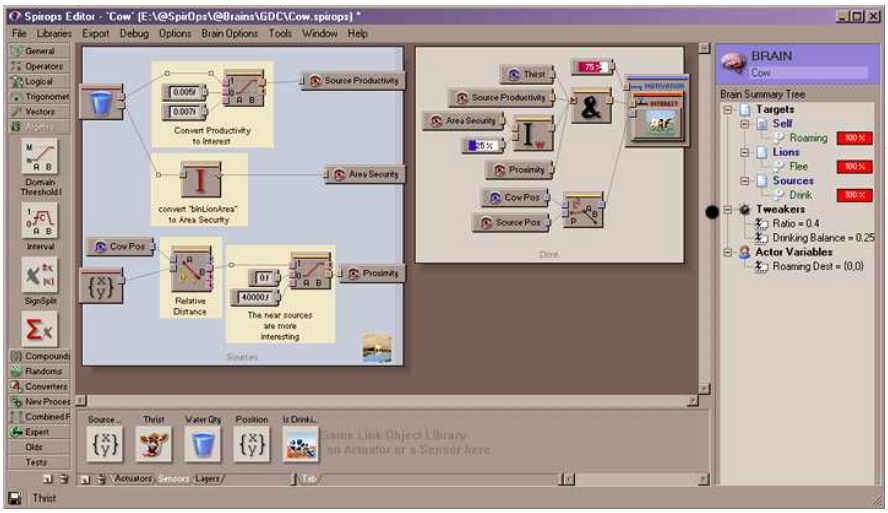


FIG. 1.17 – Editeur de SpirOps, image tirée du site officiel http ://www.spirops.com.

StarLogo TNG et Scratch

StarLogo TNG (The Next Generation) et Scratch sont deux projets du MIT (Massachusetts Institute of Technology) pour la modélisation d'applications et la simulation. Ces deux projets reposent sur une interface graphique où les instructions sont représentées par des blocs

colorés suivant la sémantique des instructions. Ces interfaces (voir la figure 1.18) ont pour but de faciliter la programmation en se focalisant sur la dynamique de la simulation. L'interface de programmation est couplée avec des outils pour la gestion du son et de l'aspect graphique. Ces projets apportent un aspect plus ludique à la programmation mais la réalisation des comportements reste très dépendante du contexte d'application. Toutefois ces deux projets ne s'intéressent pas à la conception de comportement, mais plus à la simplification de la programmation. Je m'intéresserai également à ce point en proposant une illustration de ma proposition qui se veut être simple et compréhensible dans sa réalisation et dans la configuration des comportements.

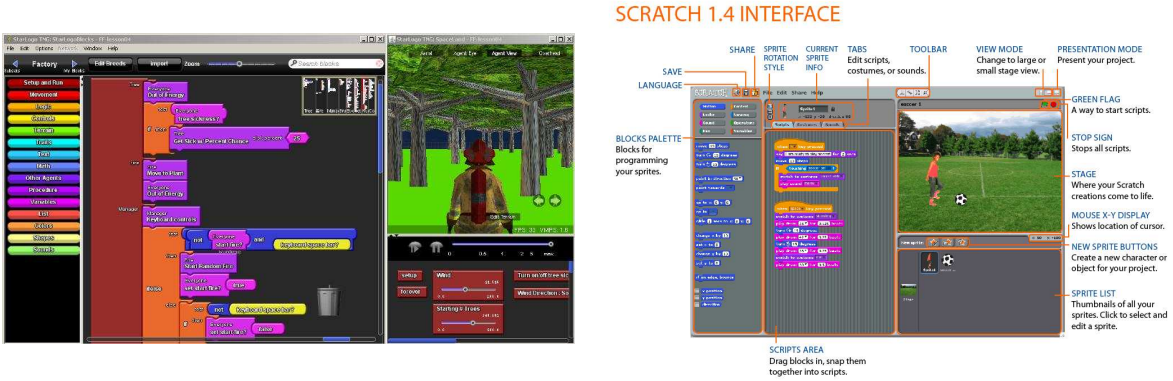


FIG. 1.18 – Interfaces de Starlogo TNG (à gauche) et de Scratch (à droite).

Kodu

Kodu[Kud10] (anciennement appelé Boku) est un environnement de conception permettant de créer des jeux vidéo. Kodu est un projet conçu par Microsoft qui devrait fonctionner sur Xbox 360 et sur PC. Il est composé d'un éditeur de terrains, d'un constructeur de chemins et de ponts et d'une vingtaine de personnages avec des capacités différentes. L'interface de conception des comportements des personnages repose entièrement sur des icônes pour représenter les personnages, les actions ou les évènements (voir la figure 1.19). Les comportements sont décrits par des règles de la forme WHEN-DO. La partie WHEN décrit une action du joueur, une collision, un son, une vision, une temporalité ou un état de l'environnement (voir la figure 1.20). La partie DO décrit les conséquences une fois la partie WHEN satisfaite. L'ordre de définition des règles correspond à l'ordre d'exécution.

Kudo propose une interface pour la conception de comportement en masquant totalement l'aspect programmation via l'utilisation d'icônes. On peut voir sur différentes vidéos que son utilisation est accessible aux enfants d'une dizaine d'années. Cette interface est simple d'utilisation mais les comportements sont fortement restreints.

FIG. 1.19 – Interface de conception de Kudo.

FIG. 1.20 – La sélection des règles présentée par une roue, ici, la roue des déclencheurs d'actions.

1.3.3 Les MMORPG

Les MMORPG (*Massively Multiplayer Online Role Playing Game*) constituent une catégorie spécifique des CRPG (*Computer Role-Playing Game* pour les jeux de rôle sur ordinateur). Ce sont des jeux de rôles en ligne où un grand nombre de personnages (joueurs inclus) évoluent dans un environnement virtuel persistant (qui continue d'évoluer que le joueur soit connecté ou non). Un MMORPG correspond à un modèle d'application en "perpétuel développement", le développement de ces applications est incrémental et fréquemment l'environnement est étendu : les capacités des PNJ et des joueurs changent, de nouveaux éléments sont ajoutés au jeu, etc. Ce contexte est donc fortement contraint, de part l'immensité de l'environnement, le nombre important d'agents (joueurs ou personnages non joueurs) et aussi son développement incrémental. Les MMORPG offre donc un cadre applicatif intéressant de mes travaux, de part la contrainte liée à l'environnement.

De plus, dans les MMORPG commerciaux, les personnages non joueurs sont souvent cantonnés à des comportements simples : soit ils sont présents comme décors (aucune interaction avec les joueurs ou même avec l'environnement n'est possible), soit leur rôle est limité. Bien souvent le comportement d'un PNJ est restreint à une activité fonctionnelle ou quelques interactions avec les joueurs. Les PNJ sont ainsi utilisés comme des marchands vendant des items aux joueurs, des gardiens ouvrant des portes ou des donneurs de quêtes. Il existe aussi des PNJ qui combattent les joueurs, ces derniers ont également un comportement limité (la réalisation d'une séquence d'actions identiques qui se répètent selon une temporalité pré-définie) ou un champ d'action restreint (à une zone de l'environnement). Il est rare de voir des PNJ ayant les mêmes capacités que les joueurs.

Le marché du MMORPG devrait constituer en 2010 un chiffre d'affaire de presque 5 milliards de dollars US (chiffre de l'agence française pour le jeu vidéo) dans le monde. En France, les deux MMORPG les plus joués sont Dofus (d'Ankama Games, une société française) et World of Warcraft (de Blizzard Entertainement, une société américaine).

DOFUS

DOFUS est un MMORPG, dans un univers médiéval-fantastique, édité et développé par Ankama Games sorti 2004 en France. Ce jeu en 2D en Flash[1], connaît un franc succès, surtout en France et de plus en plus dans le monde. Comme tout MMORPG, il est caractérisé par le nombre important de joueurs et la taille de l'environnement. En 2010, plus de 20 millions de comptes (abonnés et non-abonnés) sont recensés et le jeu compte jusqu'à 200 000 connexions simultanées (l'abonnement mensuel de ce jeu coûte environ 4 à 5 euros). L'environnement du jeu est discrétisé en zones (10 000 zones différentes dans le jeu), une zone est composée de 16x16 cases.

Dans ce jeu, le joueur incarne un personnage par les 12 classes possibles. Son but est de collecter les Dofus pour devenir plus puissant. Pour cela, le joueur doit détruire des monstres et résoudre des quêtes. Dans ce jeu, il existe deux types de PNJ, les donneurs de quêtes et les monstres. Les donneurs de quêtes sont des entités dont le comportement est très limité, ils restent dans une zone précise, donnent des missions que le joueur doit accomplir, échangent, achètent ou vendent des items. Les monstres sont des entités que le joueur doit combattre, ils

[1] la version 2.0 de Dofus sortie début 2010 est un compromis entre Flash et Java

restent dans une zone précise mais ils peuvent disparaitre en cas de défaite (dans ce cas un nouveau groupe de monstres réapparaît au bout d'un certain temps).

Les combats ont un rôle important dans le jeu, le type de combat est du tour par tour (chaque entité du combat : joueur ou monstre, joue à tour de rôle) dans une zone qui est discrétisée en cases. Chaque attaque à un coup en points d'actions (PA) et possède une portée (distance minimale et/ou maximale pour atteindre une cible). Pendant son tour le joueur peut se déplacer (changer de case) en utilisant des points de mouvements (PM). Une fois le tour terminé, le joueur (et les monstres) retrouvent leur PA et leur PM.

Dans [Bon08], Laetitia Bonte (responsable IA du jeu DOFUS) explique comment l'IA des monstres est calculée pendant le combat. L'algorithme de sélection d'action à exécuter pour chaque monstre est le même pour toutes les créatures, il se découpe en trois étapes :

1. La sélection des sorts possibles : les sorts ont des contraintes comme des intervalles de relances ou l'état du monstre, cette étape établit la liste des sorts exécutables.
2. La construction des combinaisons possibles : cette étape détermine en fonction des PA du monstre et du coût en PA de chaque sort, l'ensemble des combinaisons possibles.
3. Le choix de la meilleure combinaison : cette étape est la plus complexe, elle prend en compte les cibles et les préférences du monstre.

Pour le choix de la meilleure combinaison, l'algorithme prend chaque sort et détermine quelles sont les cibles pour lesquelles le sort est applicable (si la cible est à portée où si le monstre a assez de PM), va simuler son application et en déduire un score. Puis l'algorithme passe à un sort dont la combinaison avec le premier est possible, il va déterminer les cibles et simuler son application pour en déduire un nouveau score. L'algorithme continue ainsi de suite avec tous les sorts de la combinaison (voir la figure 1.21). Les scores des sorts d'une combinaison sont agrégés par une somme pour obtenir le score de la combinaison, la combinaison ayant obtenu le meilleur score sera celle que le monstre exécutera. La détermination du score d'une action (d'un sort) dépend des préférences qu'a la créature sur les actions qu'elle peut faire (par exemple soigner un allié ou attaquer un ennemi) ce qui rend la sélection d'action personnalisable et lui permet d'être applicable à tous les monstres du jeu.

World of Warcraft

World of Warcraft (aussi parfois nommé WoW) est un MMORPG, dans une univers médiéval-fantastique, développé par Blizzard Entertainment qui est sorti en France en 2005. Basé sur la série de jeux Warcraft du même développeur, ce jeu connaît un énorme succès (11,5 millions d'abonnés dans le monde, l'abonnement mensuel coûtant une dizaine d'euros) qui en fait la référence en terme de MMORPG. Ce jeu propose d'incarner une des 10 races (12 avec la prochaine extension) du jeu et de choisir parmi les 10 classes disponibles pour devenir le joueur le plus puissant (les restrictions sur la compatibilité entre la race et la classe ne permettent pourtant pas de pouvoir choisir parmi 100 types de personnages). Le jeu présente le personnage dans un environnement en 3D, les actions et les combats se font en temps-réel.

Dans World of Warcraft, les joueurs interagissent avec deux types de PNJ, qui sont comme pour Dofus les donneurs de quêtes et les monstres (voir la figure 1.22). Au niveau du comportement, les monstres ont un comportement assez basique, le but du jeu étant de les tuer en masse (les joueurs utilisent le verbe "to farm" pour qualifier cette phase de jeu répétitive),

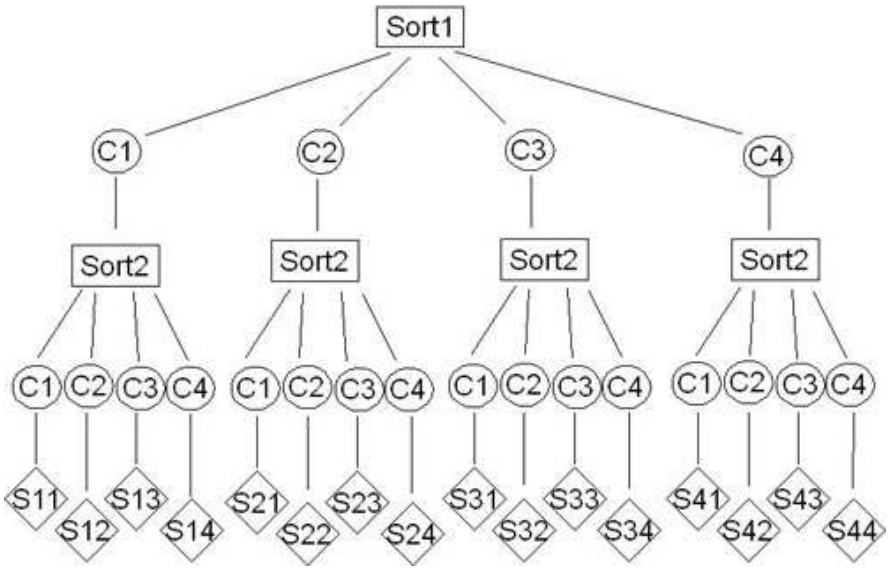


FIG. 1.21 – La simulation des combinaisons de sorts possibles (C1, C2, C3 et C4 sont les cibles et les Si en feuille sont les scores de chaque combinaison, image tirée de [Bon08].

FIG. 1.22 – Interface World of Warcraft. Dans cette image-écran nous pouvons voir un PNJ donneur de quête (avec le point d'exclamation au dessus de la tête) et un monstre entouré de rouge.

les monstres de plus haut niveau (les boss) ont un comportement très prévisible, d'ailleurs un module non-officiel (nommé Bigwigs) a été fait pour que les joueurs puissent connaitre le temps restant avant que le monstre ne fasse certaines attaques (avec un système de compte à rebours). De plus les monstres possèdent une zone d'agression qui limite leur comportement (en dehors de cette zone le joueur n'est pas attaqué).

1.4 L'équipe SMAC Lille

L'équipe Systèmes Multi-Agents et Comportements (SMAC) de Lille, dans laquelle j'ai effectué cette thèse, s'intéresse à la recherche de solutions agents pour la représentation de phénomènes divers. Parmi les différentes thématiques sous-jacentes, il est possible d'extraire quatre groupes de travail : l'aspect réactif avec la modélisation large échelle et multi-niveaux, l'aspect cognitif avec la conception de comportements orientés buts et les stratégies d'équipes, les systèmes de négociation et la finance computationnelle. Les deux premiers groupes de travail sont réunis pour défendre l'approche centrée interaction pour la modélisation de systèmes multi-agents. Afin de mieux appréhender l'existant qui sera évoqué dans ce document (c'est-à-dire le projet CoCoA), je présenterai cette approche et la méthodologie IODA.

1.4.1 L'approche centrée interaction

Classiquement les systèmes multi-agents sont dits "centrés agent", c'est-à-dire qu'ils se focalisent sur la conception d'agent afin de résoudre une problématique. Cette conception permet d'étudier individuellement les caractéristiques des agents, de comprendre les conséquences de ces caractéristiques sur leur comportement et sur le déroulement de la simulation (i.e. la résolution du problème). L'approche que défend l'équipe SMAC de Lille depuis quelques années est une approche centrée sur les interactions. Cette approche sépare la conception des agents de la conception des interactions. Les agents sont donc des entités dans lesquelles le développeur peut ajouter ou supprimer des interactions.

Le principe

L'approche centrée interaction est également une approche tout agent, ce qui veut dire que chaque entité présente dans l'environnement est un agent. Dans l'approche centrée interaction, un agent peut subir et/ou peut effectuer des interactions dans l'environnement. Cette approche distingue les agents qui ne peuvent que subir des interactions qui sont qualifiés de **passifs**, des agents qui peuvent effectuer au moins une interaction qui sont appelés **actifs** (les actifs pouvant également subir des interactions). Les agent passifs sont vus comme des objets de la simulation, alors que les actifs sont des acteurs qui jouent un rôle dans la simulation. Le principe d'une simulation basée sur l'approche centrée interaction est que la dynamique de simulation est engendrée par l'association d'un agent qui peut effectuer une interaction avec un agent qui peut la subir (voir figure 1.23).

Soit $\mathcal{I}$ l'ensemble des interactions, alors le principe peut être défini ainsi :

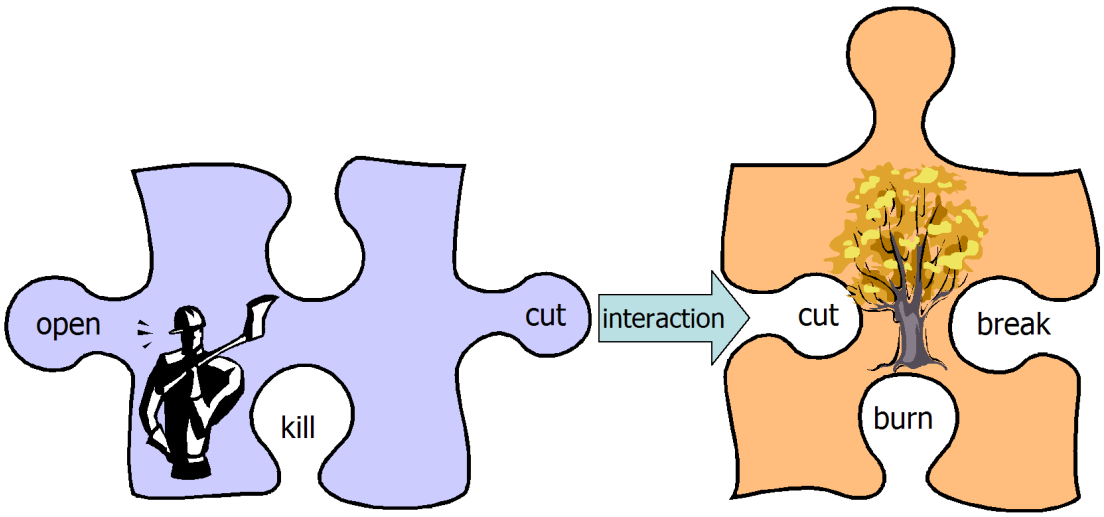


FIG. 1.23 – Si un agent `Arbre` (qui est un agent passif) peut subir l'interaction *couper*, *casser* et *brûler* et un agent `Bûcheron` (qui est un agent actif) peut effectuer les interactions *ouvrir* et *couper*, alors l'agent `Bûcheron` peut effectuer l'interaction *couper* sur un agent `Arbre`.

$$\forall c \in \mathcal{A}ctives, t \in \mathcal{A}gents,$$
$$\text{si } \exists i \in \mathcal{I} \mid c.\text{peut-effectuer}(i) \cap t.\text{peut-subir}(i)$$
$$\text{alors } c.\text{exécuter}(i, t)$$

Dans les chapitres suivants, je nommerai **capacité**, l'interaction qu'un agent **peut effectuer**.

Interaction

Les interactions décrivent les lois qui régissent le monde simulé et constituent une connaissance manipulable par les agents. Une **interaction** est définie par un nom et trois parties :

- la *condition* teste le contexte d'exécution de l'interaction, ce qui consiste à tester principalement les valeurs de la cible ou les propriétés de l'acteur.
- la *garde* exprime une condition plus générale qui dépend plus de l'environnement. Elle vérifie les conditions d'application de l'interaction dans l'environnement, le plus souvent cette garde sert à exprimer la distance minimale d'exécution d'une interaction. Cette partie de l'interaction n'intervient que dans le cas d'une simulation située.
- une *action* décrit les conséquences de l'exécution de l'interaction. Elle peut agir sur les propriétés de l'agent cible, l'agent acteur ou l'environnement (comme la création ou la destruction d'un agent).

Dans l'approche centrée interaction, les agents (passifs et actifs) sont des entités qui possèdent des propriétés. Une propriété est une caractéristique propre à l'agent, par exemple son nom, sa couleur, son âge, son niveau d'énergie. Dans l'approche centrée interaction, il est donc important de vérifier que les agents qui ont un rôle dans l'interaction possèdent bien

$$open : \begin{cases} \texttt{condition} & : & \texttt{target.opened} = \texttt{false} \\ \texttt{garde} & : & \texttt{distance}(\texttt{actor}, \texttt{target}) < 1 \\ \texttt{action} & : & \texttt{target.opened} = \texttt{true} \end{cases}$$

FIG. 1.24 – L'interaction ouvrir est définie par une condition (que la cible soit fermée), une garde (ouvrir est possible à une distance inférieure à 1) et une action (faire passer l'état de la cible à ouvert).

les propriétés nécessaires à l'exécution de l'interaction. Par exemple, si un agent `Pomme` peut subir l'interaction `manger`, cette interaction a pour effet de donner de l'énergie à l'agent qui a effectué l'interaction et de détruire l'agent cible. Il faut donc que l'agent qui peut effectuer l'interaction `manger` possède une propriété `énergie`. De plus, si la valeur de l'énergie gagnée dépend de la valeur de l'énergie de la cible de l'interaction, il faut que cette cible, dans notre exemple l'agent `Pomme`, possède également la propriété `énergie`.

Afin d'illustrer la présentation d'une interaction, de ses trois parties et de l'utilisation des propriétés des agents, prenons l'exemple de l'interaction *ouvrir* de la figure 1.24. Cette interaction fait passer un objet (porte, fenêtre, etc.) qui est cible (target) de l'interaction, de l'état *fermé* à l'état *ouvert*. L'état de l'objet ouvert et fermé est assuré par la propriété `opened` de l'agent cible. La condition de l'interaction précise que la cible doit être fermée, en testant la valeur de sa propriété `opened`. La garde teste la distance entre la cible et la source (ou l'acteur noté actor). Comme on peut le constater, l'interaction nécessite évidemment que la cible et la source soient proches pour pouvoir exécuter l'interaction *ouvrir*. Enfin la partie action de l'alternative change la propriété `opened` de l'agent cible afin qu'elle passe dans un état *ouvert*.

La nature de la cible n'a aucune importance ici, il suffit qu'elle puisse subir l'interaction *ouvrir* (et donc qu'elle possède la propriété `opened`). Cette connaissance est ainsi représentée d'une manière "universelle" via l'interaction. Dans ce sens, les interactions sont des connaissances déclaratives : elles décrivent une action mais pas comment la résoudre ou l'utiliser ni même que l'acteur doit se déplacer vers la cible. Ainsi les interactions sont réutilisables pour différents agents et différentes simulations.

Une garde de distance est dite satisfaite si l'expression booléenne sur la distance est évaluée à vrai. Une condition est dite satisfaite si l'expression booléenne associée est évaluée à vrai. Une interaction est dite **exécutable** lorsque la condition est satisfaite et la garde de distance est satisfaite. Néanmoins, une interaction exécutable ne va pas nécessairement s'exécuter. La garde et la condition de l'interaction n'indiquent pas les contraintes d'exécution des interactions, mais bien les conditions nécessaires pour que l'interaction puisse être exécutée.

1.4.2 Méthodologie IODA

Définition

IODA (pour Interaction Oriented Design of Agent simulations) est une méthodologie pour la conception de simulations multi-agents. Cette méthodologie est l'héritière de l'approche centrée interaction [MRU01, KMP08]. IODA a été conçue à l'origine pour les simulations d'agents réactifs. Plusieurs plateformes de simulation à base d'agent utilisent la méthodologie,

dont SimuLE (qui est l'acronyme de Simulations Large Echelle) et JEDI (Java Environment for the Design of agent Interactions).

Cette méthodologie a pour but de permettre à des utilisateurs, qui ne sont pas forcément des spécialistes dans la modélisation agent, de mettre en œuvre des simulations agents dans des domaines variées. Cette méthodologie se décompose en trois étapes :

1. Identification des agents et des interactions
2. Construction de la matrice d'interactions
3. Implémentation des interactions et des primitives

La méthodologie IODA permet une construction incrémentale des simulations. À tout moment, il est possible d'ajouter des éléments et de reprendre les trois phases de conception. Cette méthodologie a été initialement prévue pour les simulations réactives, néanmoins moyennant quelques modifications nous montrerons comment elle a été appliquée pour des simulations cognitives dans le projet CoCoA.

Identification des agents et des interactions

Cette phase de conception a pour but de décrire les interactions et les agents qui auront un rôle dans la simulation. Dans certains domaines, tels que la biologie, les modèles que les utilisateurs cherchent à simuler n'ont pas de représentations formelles. Il est donc important de pouvoir prendre en compte dans cette phase une description en langage naturelle des éléments présents dans la simulation.

Construction de la matrice d'interactions

La matrice d'interactions permet de déterminer les cibles et les sources des interactions de la simulation. La présentation sous forme de matrice permet de présenter les relations cible-source et de pouvoir détecter les incohérences ou les manques de précision dans les descriptions de la phase précédente. La méthodologie IODA pour les simulations réactives, préconise l'utilisation de la matrice comme référent centralisé pour tous les agents. Cette matrice permet ou non d'effectuer ou de subir une interaction, limitant ainsi les erreurs de conception. Dans l'exemple du loup, de la chèvre et du chou (voir figure 1.25), pour éviter que loup puisse manger le chou il y a deux possibilités. Soit, les agents se réfèrent à la matrice d'interaction (une information centralisée) pour savoir s'ils peuvent ou non exécuter une interaction sur une cible particulière. Soit il faut définir deux interactions `mangerCarnivore` et `mangerHerbivore`, afin que le loup qui est carnivore ne mange pas le chou, tout en assurant que la chèvre qui est herbivore puisse encore manger le chou.

Implémentation des interactions et des primitives

En utilisant la matrice d'interactions et les descriptions de la première phase, le concepteur peut ensuite s'attaquer au codage des interactions et des primitives.

Contrairement à l'approche centrée interaction que nous avons présentée précédemment, la méthodologie IODA ne préconise pas de manipuler directement les propriétés des agents dans le code de l'interaction. La méthodologie IODA propose de manipuler dans les interactions des

Cibles \ Sources	loup	chèvre	chou
loup			
chèvre	manger		
chou		manger	

FIG. 1.25 – Voici un cas où la matrice peut montrer des incohérences, dans cet exemple le loup peut-effectuer manger sur la chèvre, la chèvre peut-subir manger par le loup et peut-effectuer manger sur le chou, le chou peut-subir manger par la chèvre. En affectant l'ensemble des interactions peut-effectuer et peut-subir de chaque agent, nous remarquons que le chou peut-subir manger et le loup peut-effectuer manger, ainsi par transitivité le loup peut-effectuer manger sur le chou, alors que cette interaction n'est pas autorisée dans la matrice d'interaction. Afin de résoudre ce problème mis en évidence par la matrice d'interaction, l'interaction manger doit être définie de telle manière que cette incohérence ne puisse pas ce produire.

primitives (qui sont des bouts de codes). Ces primitives correspondent à du code implémenté dans les agents, permettant une interprétation d'une interaction qui est spécifique à chaque agent. Ainsi, la notion énergie qui était précédemment une propriété, est représentée par une primitive qui permet pour chaque agent (ou type d'agent) d'avoir sa propre interprétation et évolution de la notion d'énergie.

Les différences avec l'approche centrée interaction

Dans IODA, l'utilisation de primitives permettent une expressivité plus importante et une interprétation différente des interactions selon l'agent. Néanmoins, le fait que ces primitives soient du code à rajouter au sein de l'agent rend l'utilisation de l'approche IODA plus délicate et requiert des compétences en programmation. De plus dans l'approche IODA l'utilisation de la matrice d'interaction pour autoriser l'exécution d'une interaction selon la cible est préconisée. Lorsque l'on cherche à modéliser des agents que l'on souhaite autonome et que l'on désire qu'ils expriment des traits de caractère ou une personnalité, il est important d'éviter de centraliser les informations.

Dans l'approche centrée interaction, ce sont les interactions qui manipulent les propriétés des agents. La séparation entre l'interaction et l'agent est ainsi plus importante, le code de l'interaction n'est pas dépendant du codage de l'agent, l'interaction ne nécessite que l'accès aux valeurs des propriétés de l'agent pour se réaliser. Les agents sont codés de la même manière c'est l'affectation des différentes interactions peut-subir et peut-effectuer qui va personnaliser le comportement de l'agent.

1.4.3 Bilan

L'approche centrée interaction est une approche tout agent, où la connaissance de la dynamique (les actions qui peuvent être effectuées et les changements dans l'environnement) de la simulation est contenue dans les interactions. Un agent peut-subir et/ou peut-effectuer une interaction. Ces deux notions mettant en jeu deux agents, la cible (qui peut-subir l'interaction) et l'acteur (qui peut-effectuer l'interaction). Une interaction contient les informations néces-

open :			*open*(*source*, *target*) :		
condition	:	target.opened = false	trigger	:	target.isClosed()
garde	:	distance(actor,target) < 1	preconditions	:	source.isAbleToOpen(target)
action	:	target.opened = true	actions	:	target.open()

FIG. 1.26 – La description des interactions dans l'approche centrée interaction contient à elle seule toutes les informations nécessaires à son exécution (à gauche). Dans IODA (à droite), une même interaction peut avoir des conditions d'exécution (partie condition et trigger) et des conséquences de l'application (partie action) différentes pour chaque agent (décrites dans le *open*() de la cible), cette description des interactions de par l'utilisation des primitives nécessite donc l'ajout de code dans les agents.

saires à son exécution avec les propriétés des agents acteur et cible, la garde de distance de l'interaction (qui assure le respect de l'aspect situé de la simulation). La partie action de l'interaction décrit les conséquences de l'exécution de l'interaction pour les agents acteur, cible ou l'environnement. Toutes ces informations représentent une connaissance déclarative qui peut être manipulée par un planificateur procédural comme c'est le cas dans CoCoA. La séparation du déclaratif et du procédural permet aux interactions d'être génériques et réutilisables pour d'autres agents ou d'autres simulations. La méthodologie IODA qui est plus orientée pour les simulations d'agents réactifs, est héritière de l'approche centrée interaction. Les différences entre IODA et l'approche centrée interaction, s'appliquent également dans CoCoA, qui en font deux approches cousines pour les simulations d'agents.

1.5 Conclusion

Le problème pour les concepteurs de comportement n'est pas seulement de fournir un personnage (ou agent) avec la possibilité d'effectuer des actions, mais également de concevoir l'agent de telle manière qu'il exprime une certaine personnalité. Un personnage qui possède un comportement uniforme exprimerait un effet "robot" qui pourrait être jugé négativement par les observateurs. C'est le cas notamment dans les jeux vidéo, où des personnages non-joueurs doivent agir. Si ces personnages sont capables de se comporter différemment, ceci aura pour effet d'augmenter le réalisme du jeu. En effet, il est préférable dans un jeu qu'un personnage donné agisse différemment d'un moment à l'autre. Afin de répondre à ce problème, de nombreuses solutions de conception ont été mises en place dans les jeux vidéo. Parmi ces solutions, certaines consistaient à déterminer l'ensemble des situations dans lesquelles l'agent pouvaient se trouver et tentaient d'apporter une solution à chacune d'entre-elles (les Scripts, les FSM, BehaviorShop, Kodu).

Mais agir différemment suivant la situation n'est pas tout. Il faut également que les personnages agissent différemment les uns des autres, qu'ils expriment une certaine personnalité. Il faut donc définir pour chaque situation et pour chaque personnalité, les actions que l'agent doit effectuer. Or, dans un jeu vidéo, les personnages non joueurs sont différents les uns des autres. Ils possèdent des buts différents et leurs connaissances évoluent indépendamment les unes des autres au cours du temps. Néanmoins, leurs comportements doivent s'adapter à ses évolutions, ce qui rend la conception des comportements difficile.

Un autre problème pour le concepteur est la conception elle-même. Cette tâche ne doit pas être trop complexe et il devrait être possible d'obtenir facilement des comportements distincts. Mais depuis quelques années, de plus en plus de propositions tentent de simplifier la conception en se focalisant sur les parties spécifiques du comportement qui rend un personnage unique. L'application ou la gestion de conflits entre les comportements étant gérer automatiquement lors de l'application de ce comportement dans le contexte (SpirOps). Dans ces propositions, le PNJ possède des motivations ou des centres d'intérêts qui vont le pousser à agir. Ce n'est plus la situation qui pousse l'agent à agir mais ses motivations qui vont être adaptées à la situation. Le but de cette thèse est de faire une proposition dans ce sens. Le comportement de l'agent sera défini à l'aide de motivations qui orienteront le choix de l'action à exécuter. Ce choix s'effectue sur l'ensemble des actions que l'agent peut effectuer selon ses capacités et ses connaissances afin de résoudre ses buts. Les buts sont des actions à effectuer ou des situations à atteindre pouvant notamment répondre à des propriétés internes (comme dans les approches motivationnelles PECS, Andriamasinoro, deSevin ou MHiCS) qui évoluent pendant la simulation.

Ce que je propose, c'est d'obtenir un moteur comportemental utilisant une définition du comportement qui soit réutilisable d'une simulation à une autre ou même d'un agent à un autre. Cette définition comprenant les éléments qui influencent le comportement de l'agent : les motivations. Pour cela, il faut s'abstraire des éléments la simulation. En effet, si l'on désire définir un comportement particulier, à gros grains, cette définition ne requiert pas d'informations sur les autres agents ou sur la simulation dans laquelle l'agent va être utilisé. Bien qu'il puisse être important de prendre en compte son contexte (son environnement et les autres agents), pour être crédible la définition d'un comportement ne nécessite pas d'en être dépendant.

Ainsi, je ne propose pas de concevoir des comportements pour une simulation particulière ou pour un ensemble de situations particulières mais bien de définir un comportement d'agent qu'on pourrait qualifier de "générique", c'est-à-dire indépendant du contexte d'application. C'est au moteur comportemental de se charger d'appliquer cette définition au mieux dans un contexte particulier (comme le préconise en partie Orkin pour le jeu F.E.A.R.). Je parle d'appliquer cette définition "au mieux" car si le contexte ne permet pas d'exprimer certains comportements, la meilleure définition ne pourra pas y changer grand chose. Par exemple, si je souhaite utiliser un agent qui possède la définition d'un agent brutal, pour exprimer ces traits de caractères le contexte doit lui en donner l'opportunité (par exemple, l'agent doit pouvoir casser des choses, blesser ou tuer). Pour cela, je présenterai le moteur comportemental pour les agents situés applicable à la représentation des PNJ que nous avons mis en place dans le projet CoCoA. Ce moteur se décompose en deux parties distinctes que j'appellerai **raisonnement** et **individualité**. Ce moteur se base sur un modèle dont chaque élément (motivation, paramètre, fonction de combinaison, etc) est compréhensible (pas d'effet boîte-noire sur des valeurs obtenues et pas de valeurs difficilement interprétables), facilement manipulable, accessible à des concepteurs de jeux vidéo qui ne maitrisent pas forcément de grandes connaissances théoriques en intelligence artificielle, robuste aux évolutions du jeu et dont les éléments sont réutilisables pour d'autres agents, d'autres jeux ou simulations.

Chapitre 2

Le projet CoCoA

« Ne perdons rien du passé. Ce n'est qu'avec le passé qu'on fait l'avenir. »
Anatole France

Cette thèse entre dans le cadre du projet CoCoA (pour Cognitive Collaborative Agents) de l'équipe SMAC de Lille. Le but de mes travaux était de réfléchir sur la notion de comportement et d'en donner une implémentation concrète pour CoCoA et sa plateforme de simulation pour agents cognitifs situés. C'est pourquoi, dans ce chapitre, je présenterai un "état des lieux" de CoCoA, inspiré notamment de [Dev07] et de [Rou05]. Cet état des lieux permettra de mieux appréhender ma contribution au projet CoCoA.

Le projet CoCoA propose une solution pour la modélisation de simulation d'agents cognitifs situés basée sur l'approche centrée interaction. Cette approche elle-même basée sur le modèle "tout-agent" : dans CoCoA, chaque élément de l'environnement est un agent. Dans cet environnement, les agents peuvent subir et/ou effectuer des interactions. Les agents pouvant effectuer des interactions sont appelés des **agents actifs**, les agents ne pouvant que subir des interactions sont des **agents passifs**. Les agents actifs possèdent un moteur comportemental qui leur permet d'être les acteurs de la simulation. Ils ont une perception limitée de leur environnement, ils mémorisent les informations perçues et planifient les interactions à effectuer afin de résoudre leurs buts. Les agents sont géographiquement situés dans l'environnement. Ils ont une position et les notions de distance ou de voisinage ont un sens et peuvent être calculées. La gestion des déplacements d'un agent est un critère important de l'évaluation de la crédibilité d'un comportement observé. Nous verrons dans ce chapitre, que l'aspect situé des simulations a un impact sur la planification des actions à exécuter. L'approche centrée interaction permet de définir un agent à l'aide des interactions qu'il peut effectuer et subir, de ses propriétés, et des buts qu'il doit résoudre. Le même moteur est utilisé pour tous les agents.Les principes du projet CoCoA ont été mis en œuvre dans une plateforme également nommée CoCoA, permettant de créer et de tester des simulations d'agents cognitifs situés.

Dans ce chapitre, j'expliquerai dans un premier temps les principes mis en place par CoCoA pour la conception d'agents cognitifs situés. Puis, je détaillerai l'architecture des agents et les divers éléments qui la composent. Je présenterai ensuite, la plateforme pour la conception de

simulations centrées interaction avec agents cognitifs situés. Enfin, je conclurai sur cet "état des lieux", afin d'introduire ma proposition dans le domaine de la conception de comportement et son application dans CoCoA.

2.1 Les principes du projet CoCoA

2.1.1 Tout est agent

Le projet CoCoA se base sur l'approche centrée interaction (présentée dans la partie 1.4.1). Cette approche est elle-même fondée sur le modèle "tout-agent". Ainsi, toutes les entités présentes dans l'environnement d'une simulation CoCoA sont des agents et un agent CoCoA est caractérisé par les interactions qu'il peut subir ou effectuer et par ses propriétés[2] (voir la figure 2.1). Les propriétés de l'agent permettent de définir son état. Par exemple un agent peut avoir une propriété couleur, une propriété taille ou encore un niveau d'énergie.

```
<agent>          ::=  <passive-agent> | <active-agent>
<passive-agent>  ::=  { ("name", Symbol),
                        ("can-suffer", <interaction>*),
                        <property>*}
<active-agent>   ::=  <passive-agent> ∪
                        { ("can-perform", <interaction>*),
                        ("goals", goal*),
                        ("memory", (degraded) environment),
                        ("engine", engine)}
<interaction>    ::=  Symbol
<property>       ::=  Symbol, value
<goal>           ::=  {(<interaction> | <state>), priority}
<priority>       ::=  value
```

FIG. 2.1 – Définition d'un agent CoCoA.

Dans CoCoA, nous distinguons les agents *passifs* qui ne peuvent que subir des interactions, des agents *actifs* qui peuvent effectuer au moins une interaction et peuvent éventuellement en subir (voir l'exemple d'un agent actif en figure 2.2). Comme les agents actifs effectuent des

[2]Les agents sont identifiés par leur nom qui est une propriété commune à tous les agents de la simulation afin de pouvoir être utilisée comme identifiant des agents

Acteur	propriétés :	`nom, inventaire`
	peut-effectuer :	`open, eat, take, moveTo, moveAway, explore`
	peut-subir :	-

FIG. 2.2 – Exemple d'un agent actif CoCoA, il possède un nom et un inventaire (un sac). Cet agent peut effectuer les interactions `open, eat, take, moveTo, moveAway` et `explore` et il ne peut pas subir d'interactions.

	Actif	Passif
Mobile	bûcheron	`Impossible`
Non Mobile	pommier	pomme

FIG. 2.3 – Exemples de différences entre agents *actifs* et *passifs* et la notion de *mobilité*. Nous définissons trois agents, un agent `bûcheron`, un agent `pommier` et un agent `pomme`. L'agent `bûcheron` qui peut effectuer les interactions *couper*, *manger* et *se déplacer*. L'agent `pommier` qui peut effectuer l'interaction *créer-pomme* et peut subir l'interaction *couper*. L'agent `pomme` peut subir l'interaction *manger*. Dans cet exemple, l'agent `bûcheron` est *actif* et *mobile*, l'agent `pomme` est un agent *passif* et l'agent `pommier` est un agent *actif non mobile*. La mobilité étant liée à l'exécution d'une action de déplacement, il n'est pas possible d'obtenir un agent passif qui peut se déplacer.

interactions, ils possèdent un moteur comportemental leur permettant d'agir dans l'environnement. Les agents actifs sont les acteurs de la simulation, les agents passifs sont des objets. Il est important de faire la différence entre les agents actifs et la mobilité des agents. La mobilité est liée à la capacité d'effectuer une interaction de déplacement, tous les agents mobiles sont des agents actifs, mais la réciproque n'est pas toujours vraie (voir la figure 2.3). La mobilité bien qu'observable n'est donc pas un critère discriminant les agents actifs des agents passifs.

2.1.2 Les agents actifs

Les agents actifs sont cognitifs et proactifs. Ils sont responsables de la dynamique de la simulation : ils peuvent effectuer des interactions et donc agissent dans l'environnement. Les agents actifs sont dirigés par les buts, c'est-à-dire qu'ils agissent afin de résoudre des buts. Pour cela, les agents actifs sont dotés de mécanismes leur permettant d'agir dans l'environnement (voir le détail dans la partie 2.2). Ces agents sont non omniscients, ils ne connaissent pas tout leur environnement et n'ont aucun *a priori* sur sa topologie, la présence d'autres agents, leur position ou leur état avant le début de l'exécution. Ils découvrent leur environnement dont ils ont une vision limitée par le module de perception. Les informations perçues par l'agent sont collectées et ajoutées dans sa mémoire (sa base de connaissances notée KB). Les agents actifs ont des buts à résoudre, la satisfaction de ces buts les mène à planifier des résolutions. Le module de planification détermine les actions à exécuter pour résoudre ces buts. L'exécution d'une action peut entrainer une modification de l'environnement ou de la perception de l'environnement, ces modifications mettent à jour les connaissances de l'agent (dans sa mémoire).

2.1.3 L'influence du situé

Les agents CoCoA sont des agents situés dans leur environnement. C'est-à-dire qu'ils évoluent dans un espace euclidien, qu'ils perçoivent leur environnement et agissent en conséquence. L'environnement est composé de places et d'obstacles et contrairement aux environnements informés comme dans [Sep07], il ne comporte aucune information sur le comportement des agents. Les agents situés CoCoA sont également capables de déterminer la position (perçue ou

mémorisée) d'autres agents ou d'obstacles de l'environnement et de raisonner sur les notions de distance ou de voisinage.

Les déplacements d'un agent sont un facteur important dans l'évaluation de la rationalité du comportement observé. Le caractère situé des simulations a également un impact sur la réalisation du plan de l'agent :

- Soit des déplacements sont nécessaires pour atteindre les positions dans l'environnement permettant d'effectuer les interactions (cf. la garde de distance des interactions).
- Soit les déplacements peuvent engendrer la création de nouvelles actions à accomplir. En effet, un déplacement change la zone de perception de l'agent. Ce changement peut faire acquérir à l'agent de nouvelles informations sur l'environnement et les agents dans cet environnement.

C'est pourquoi il faut distinguer le *plan abstrait* du *plan concret*. Le plan abstrait est le plan d'action, il détermine les interactions à effectuer pour résoudre les buts. Dans le plan abstrait la position des agents n'est pas considérée, ce plan est indépendant de l'aspect situé de la simulation. Le plan concret correspond au plan d'exécution du plan abstrait dans un environnement situé spécifique. Ce plan est enrichi des déplacements nécessaires à l'exécution du plan abstrait.

L'aspect situé intervient quand un agent effectue une interaction sur un autre agent, ou avec l'environnement. L'exécution d'une interaction dans un environnement situé demande des déplacements (notamment à cause de la garde de distance). Ces déplacements doivent être effectués avant l'exécution de l'interaction. Dans ce cas, le plan abstrait à lui seul n'est pas suffisant. Dans CoCoA, ces déplacements d sont donc intégrés dans le plan afin de permettre la progression de la simulation. Ainsi, un plan initial *exécuter* a, devient *exécuter* d et *exécuter* a (voir la figure 2.4). Si pour pouvoir faire le déplacement d il faut franchir une porte, le déplacement d est alors décomposé en deux sous-déplacements d_1 et d_2 interrompus par l'ouverture de la porte. Le plan d'exécution est alors *exécuter* d_1*, ouvrir la porte, exécuter* d_2 *puis* exécuter a. Si l'ouverture de la porte nécessite une clef, le planificateur de CoCoA prendra en compte le déplacement vers la clef, la prise de la clef et le déverrouillage de la porte. Le sous-plan préalable à l'ouverture de la porte (i.e. *exécuter* d_1) est donc *exécuter le déplacement d'*$_1$*, prendre la clef, exécuter d'*$_2$*, déverrouiller la porte*. Le schéma récursif de ce raisonnement est évident et l'on peut imaginer que la récupération de la clef engendre de nouvelles actions et l'on voit comment le contexte situé peut enrichir rapidement le plan abstrait. Cette planification pour agents situés est décrite plus en détail dans [DMR04].

2.1.4 Les interactions dans CoCoA

CoCoA est un projet qui se base sur l'approche centrée interaction pour la conception de simulation agent. Chaque entité de l'environnement est un agent, ces agents possèdent des propriétés et peuvent subir ou effectuer certaines interactions.

La représentation d'une interaction

Respectant l'approche centrée interaction (présentée dans la partie 1.4.1), dans CoCoA, une interaction I est représentée par triplet $I = \{condition, garde, action\}$ où *condition* définit

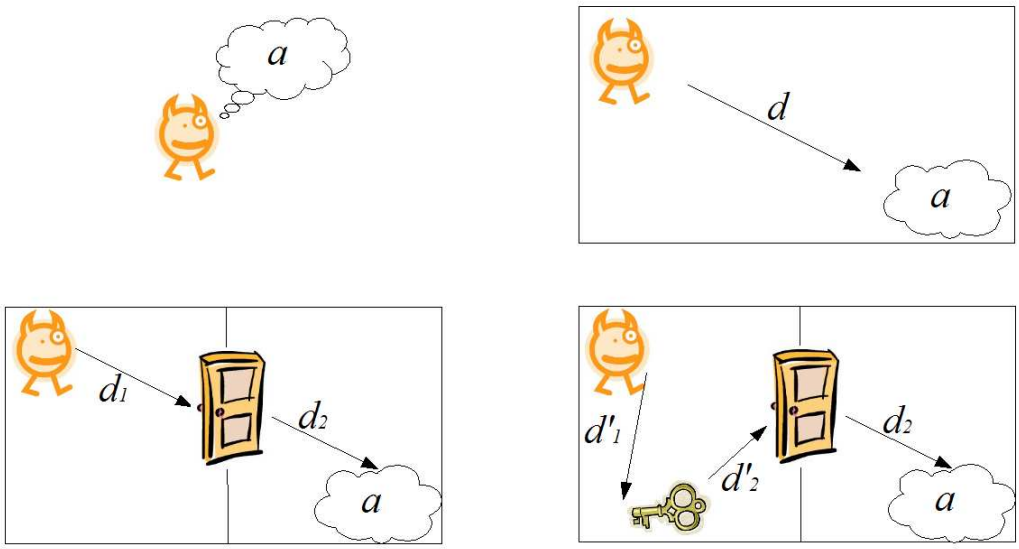


FIG. 2.4 – Influence du situé sur la planification. En haut à gauche, le plan abstrait. En haut à droite, dans un contexte situé le déplacement doit être prévu pour atteindre l'endroit d'exécution de l'action. En bas à gauche, l'environnement impose une planification, ici ouverture d'une porte. En bas à droite, cas où l'agent sait - ou croit - que la porte est cadenassée et doit donc prendre la clef avant de s'y rendre (les interactions *ouvrir* et *déverrouiller* n'apparaissent pas sur les figures).

les conditions nécessaires à l'exécution l'interaction, *garde* sont les conditions liées à l'aspect situé des simulations et *action* définit les conséquences de l'exécution de l'interaction. De par l'aspect géographiquement situés dans l'environnement des agents CoCoA, la garde de distance fait partie intégrante de l'interaction. Ainsi, la représentation d'une interaction ressemble aux règles STRIPS[FN71][3] (sans pour autant avoir la notion de garde de distance).

Les interactions sont entièrement décrites dans une représentation déclarative (voir la figure 2.5). Cette description ne nécessite pas de rajouter du code dans l'agent dans un langage de programmation pour réaliser une interaction. Dans cette description de l'interaction, `actor` représente tous les agents pouvant effectuer l'interaction (en être la source), `target` représente tous les agents pouvant subir l'interaction (en être la cible).

La garde de distance

La garde de distance est une condition sur la distance nécessaire à l'exécution de l'interaction. Dans une interaction, la garde de distance est distinguée de la partie condition, car cette condition particulière est spécifique à l'aspect situé de la simulation. De plus, alors que

[3] un opérateur STRIPS est composé des préconditions et des modifications de l'environnement. Les préconditions sont les conditions nécessaires à l'exécution de l'action. Les modifications représentent les conséquences de l'action, une modification est composée d'une liste d'éléments ajoutés et une liste d'éléments retirés après l'exécution de l'opérateur

```
<interaction>          ::=  <name>, <condition>*, <guard>, <action>*
<name>                 ::=  Symbol
<guard>                ::=  <d> <op> Integer
<d>                    ::=  distance between actor and target
<op>                   ::=  = | > | < | ≤ | ≥
<condition>            ::=  <test-property> | <predicate-primitive>(args)
<test-property>        ::=  { actor | target }.<property-name> <op> Value
<predicate-primitive>  ::=  { actor | target }.<primitive-name>(args)
<action>               ::=  <affect-property> | <primitive>
<affect-property>      ::=  { actor | target }.<property-name> = Value
<primitive>            ::=  <primitive-name>(args)
<primitive-name>       ::=  Symbol
<property-name>        ::=  Symbol
```

FIG. 2.5 – Représentation d'une interaction

les conditions donnent lieu à l'enchaînement des interactions du plan abstrait, la garde de distance est génératrice des déplacements présents dans le plan concret (présenté dans la partie 2.1.3). Enfin, il est important de noter que ces déplacements doivent être effectués après que la partie condition de l'interaction soit satisfaite. Nous verrons comment cet ordre d'exécution des interactions a été mis en place dans la partie sur la planification. En effet, si pour effectuer l'interaction `déverrouiller` sur une porte, l'agent doit posséder une clef, il ne serait pas rationnel que l'agent se déplace vers la porte avant d'en posséder la clef (c'est-à-dire avant d'avoir satisfait les conditions de l'interaction).

Les primitives

Les primitives sont des actions ou des prédicats élémentaires définis pour la création des interactions dans CoCoA. Il existe deux types de primitives, les primitives d'action appelées **primitives** et les primitives prédicat appelées **predicate primitives**.

Les primitives sont des actions élémentaires qui modifient l'environnement. Voici la liste des primitives les plus usuelles :

- **add :** permet d'ajouter un agent à une collection, par exemple ajouter un agent "pomme" dans l'inventaire : `add(actor.inventory, target)`
- **substract :** permet de soustraire un agent à une collection, par exemple retirer un agent "pomme" de l'inventaire : `substract(actor.inventory, target)`
- **createAgent :** créer un agent dans l'environnement, par exemple créer un agent "pomme" par un agent "pommier" : `createAgent(actor.agentcreationtype)`
- **destroy :** détruit un agent et le supprime de l'environnement, par exemple lorsqu'une pomme est consommée il faut la détruire : `destroy(target)`
- **move away :** permet de s'éloigner d'une cible (un pas) : `moveAway(target)`
- **move to :** permet de s'approcher d'une cible (un pas) : `moveTo(target)`
- **put :** permet de déposer un agent à la position actuelle de l'agent

```
<Interaction>
   <Name>explore</Name>
   <Garde />
   <Condition>true</Condition>
   <Action>explore()</Action>
</Interaction>

<Interaction>
   <Name>open</Name>
   <Garde>d\&lt;2</Garde>
   <Condition>target.isOpen = false</Condition>
   <Action>target.isOpen = true</Action>
</Interaction>

<Interaction>
   <Name>eat</Name>
   <Garde />
   <Condition>actor.own(target)</Condition>
   <Action>add(actor.energie,target.energie)
           and destroy(target)
           and substract(actor.inventory,target)
   </Action>
</Interaction>
```

FIG. 2.6 – Exemple de définition des interactions `explore`, `open` et `eat`.

- **remove :** permet d'enlever un agent de l'environnement (sans le détruire), par exemple pour prendre un agent "pomme" de l'environnement pour la placer dans l'inventaire (avec un add)
- **= :** permet d'affecter une valeur à une propriété
- **explore :** ordonne à l'agent de se déplacer dans l'environnement en respectant une stratégie exploration.

Les primitives prédicat, sont des primitives servant à tester la véracité d'un fait :

own : permet de savoir si l'agent possède un autre agent dans son inventaire : `actor.own(target)`

canMoveTo : permet de savoir si l'agent peut se déplacer vers la cible ou non : `actor.canMoveTo(target)`

Contrairement à IODA, les primitives CoCoA ne sont pas des appels à du code provenant d'un agent. Aucun code supplémentaire dans les agents n'est nécessaire à l'exécution d'une interaction. Le code d'une primitive d'une interaction, ne dépend pas des agents concernés par l'interaction (acteur ou cible). Une primitive est utilisable par tous les agents (voir la figure 2.6).

2.1.5 Les buts

Les agents actifs sont dirigés par les buts, les actions qu'ils réalisent ont pour objectif de résoudre ou d'avancer dans la résolution d'un but. Un but est soit une interaction à exécuter, soit un état à atteindre. Nous appelons but-interaction, un but qui est une interaction à

exécuter (voir figure 2.7), pour ce type de but, la cible de l'interaction peut être plus ou moins définie. Une fois l'interaction effectuée le but est retiré des buts à résoudre de l'agent. Un état à atteindre est appelé un but-prémisse (voir figure 2.8), ce but est représenté par un ensemble de propriétés d'agents à satisfaire. Une fois l'état atteint le but est satisfait, si l'état n'est plus valide alors le but réapparait et l'agent doit le résoudre à nouveau.

but	*type d'agent*
`eat(apple_12)`	l'agent "apple" avec le nom `apple_12`
`eat(an apple)`	n'importe quel agent la classe `apple`
`eat(*)`	n'importe quel agent qui peut subir l'interaction `eat`

FIG. 2.7 – Exemples de but interaction avec la cible qui est plus ou moins définie.

```
actor.energy > 100
```

FIG. 2.8 – Exemple de but prémisse où l'état à atteindre concerne la propriété énergie de l'agent acteur, ce but peut se traduire par "*garder un niveau d'énergie supérieur à 100*".

2.2 L'architecture des agents CoCoA

Un agent actif possède des modules lui permettant d'être l'acteur de la simulation. Chaque module prend en charge une étape du cycle d'exécution de l'agent (voir la figure 2.9) :

1. l'agent perçoit son environnement via son module de perception ;
2. il met à jour les connaissances de sa mémoire (ou base de connaissances) via les informations provenant de la perception ;
3. son module de planification met à jour les plans permettant la résolution de ses buts en prenant en compte les informations de sa mémoire et ses capacités ;
4. l'agent détermine une interaction à exécuter ;
5. il exécute l'interaction dans l'environnement.

2.2.1 La perception

Les agents actifs CoCoA ne sont pas omniscients, ils n'ont pas de connaissance a priori sur la topologie de l'environnement ou sur la présence d'autres agents dans l'environnement. Ils découvrent l'environnement au fur et à mesure suivant la perception qu'ils en ont. La perception correspond à un champ limité par une distance maximale de perception (dont la valeur dépend d'une propriété de l'agent) et la topologie de l'environnement. Les sources de perception peuvent être multiples (la vision, l'ouïe, l'odorat).Les informations mémorisées à l'intérieur du champ de perception sont vraies, en dehors, leur validité n'est pas garantie (voir la partie 2.2.3).

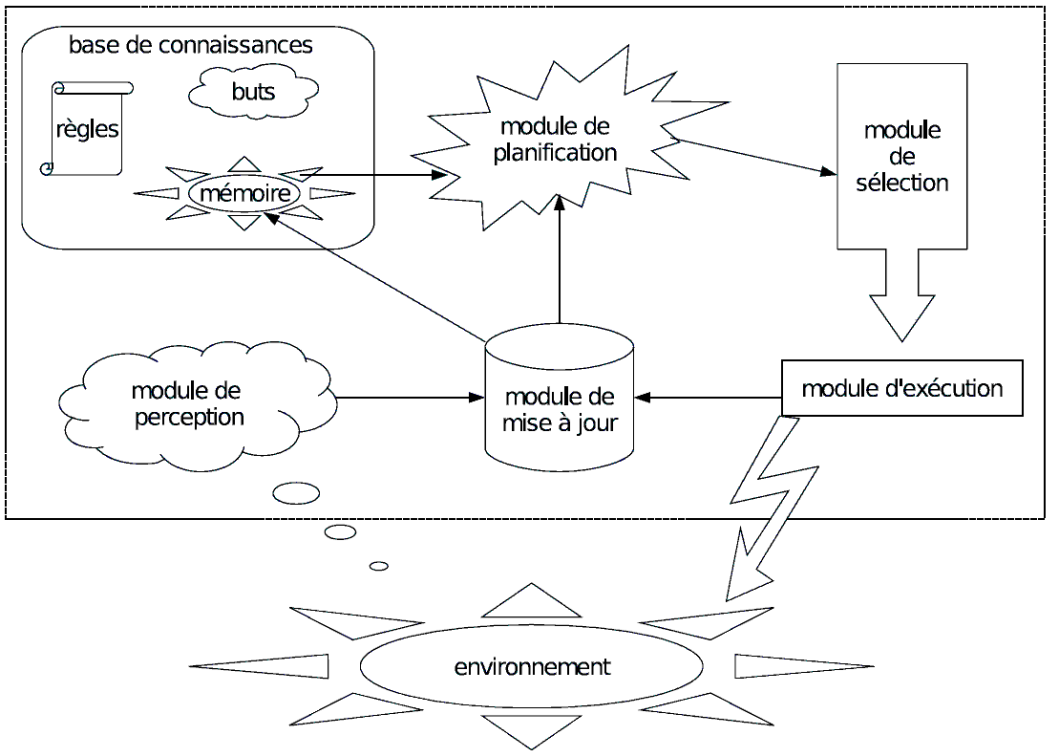


FIG. 2.9 – À chaque cycle d'exécution, un agent actif perçoit son environnement, met à jour ses connaissances et ses plans pour résoudre ses buts, sélectionne l'interaction à exécuter et l'exécute dans l'environnement.

2.2.2 La mise à jour

Une mise à jour a lieu soit parce que l'agent a exécuté une interaction, soit parce qu'il perçoit de nouvelles informations sur l'environnement. Chaque interaction exécutée change l'état global de l'environnement. Un déplacement change l'emplacement de l'agent dans l'environnement. `Manger` change la valeur de la propriété `énergie` de l'agent et détruit l'agent qui a été consommé. Ouvrir une porte change l'état de la porte de fermée à ouverte. Toutes ces modifications sont prises en compte par ce module qui met à jour les connaissances de l'agent. Le planification étant basée sur les connaissances de l'agent, les plans sont donc, si nécessaire, modifiés en conséquence. La perception met à jour les informations de la mémoire, par exemple l'emplacement des autres agents dans l'environnement. Le module de mise à jour a un rôle important notamment pour prendre en compte l'aspect situé pour la résolution des buts de l'agent. Il permet de mettre à jour les plans, et de replanifier partiellement les plans en fonction des nouvelles informations réduisant le coût du recalcul des plans. Nous n'irons pas plus loin dans l'explication de la replanification partielle dans CoCoA qui fait partie de la thèse soutenue par Damien Devigne [Dev07].

2.2.3 La mémorisation

La base de connaissances est composée des interactions que l'agent peut effectuer et subir, des buts que l'agent doit réaliser et d'une version dégradée de l'environnement, appelée mémoire. Nous parlons d'une version dégradée de l'environnement car les informations en dehors du champ de perception sont potentiellement inexactes. Pour un agent donné, chaque place, agent ou propriété d'agent, et donc l'état de l'environnement mémorisé correspond à l'état effectif de l'environnement lors de sa dernière perception. Ainsi, la position exacte des autres agents ne peut être mise à jour correctement (c'est-à-dire correspondre avec l'état "réel" de l'environnement) que s'ils entrent (ou sont déjà) dans le champ de perception.

Ce choix a été fait dans le but d'apporter un réalisme dans le comportement. Les agents CoCoA comme les êtres humains, ne sont pas omniscients et ne peuvent pas savoir ce qui se passe en dehors de leur champ de perception. Dans [Sep07] l'auteur défend l'idée qu'un agent ne doit pas gérer la représentation cohérente de tous les objets de la simulation pour rester crédible, il doit donc pouvoir se tromper : "Les erreurs ne rendent pas le personnage non crédible. Si elles sont compréhensibles, c'est-à-dire si l'observateur humain imagine la raison de l'erreur, le personnage en sera d'autant plus crédible ; par exemple, si le personnage tente d'accéder à un objet qui a été déplacé, il ne se dirigera pas vers la bonne pièce." C'est sur cette version dégradée de l'environnement que l'agent planifie la résolution de ses buts pas sur l'état réel de l'environnement. Cela signifie qu'un plan est une solution correcte pour résoudre un but selon les connaissances de l'agent au moment de la planification. Les agents CoCoA n'anticipent pas sur la position des autres agents ou plus généralement sur l'état de l'environnement en dehors de la perception. C'est-à-dire que les agents ne cherchent pas à faire des hypothèses sur l'état de l'environnement en dehors de leur champ de perception. Toutefois, l'environnement n'étant pas statique, d'autres agents le modifient, il est donc possible qu'un plan se révèle être inexact ou incomplet pendant son exécution.

Des améliorations sont en cours pour la gestion de la mémoire notamment dans le cadre d'environnements de grandes tailles comme c'est le cas dans les MMORPG[Per09].

2.2.4 La planification

Le planificateur utilise les informations contenues dans la mémoire et les capacités de l'agent pour construire des résolutions aux buts de l'agent.

Le plan est généré par un chaînage arrière. Le principe de ce chaînage est de trouver à partir d'un but à réaliser, les séquences d'actions résolvant ce but et débutant par une interaction exécutable[4], en suivant ce raisonnement :

- Pour résoudre un but de type prémisse p, le planificateur recherche une interaction i dont la partie action satisfasse p. Ensuite pour chaque condition (garde de distance incluse) c_i de l'interaction est associée un sous-but de type prémisse. Les sous-buts sont résolus de la même manière que p, provoquant un chaînage qui s'arrêtera lorsqu'à chaque prémisse sera associée une interaction exécutable dont les conditions d'exécution sont satisfaites par le contexte courant (c'est-à-dire l'état global de l'environnement courant).

[4] Une interaction exécutable est une interaction dont la partie condition et la partie garde de distance sont satisfaites

– Pour résoudre un but de type interaction, il suffit de partir des conditions d'exécution de l'interaction du but à atteindre et ensuite de réaliser le même chaînage que décrit précédemment sur chaque condition de l'interaction (garde de distance incluse).

Tous les agents actifs CoCoA peuvent effectuer au moins une interaction qui se nomme `explore`. Cette interaction possède une garde et une condition toujours satisfaite et la partie action est la primitive `explore()`. Cette interaction toujours exécutable, représente l'interaction à exécuter par défaut. Ainsi, lorsqu'un agent ne peut pas résoudre ses buts (notamment car il ne possède pas les connaissances suffisantes), par défaut, il va explorer son environnement afin de trouver de nouvelles informations pouvant l'aider dans sa résolution. Grâce à cette interaction, tous les plans débutent par une interaction exécutable, les plans qui n'ont pas assez d'informations pour proposer une résolution complète débute par l'interaction `explore`.

Dans CoCoA, les plans générés par chaînage arrière sont présentés par un $Arbre-et/ou$. Un $Arbre-et/ou$ est une structure arborescente alternant **nœud-et** et **nœud-ou** de père en fils. Les nœuds-ou représentent les interactions permettant d'atteindre un état de l'environnement représenté par une condition à satisfaire, un but ou un **nœud-et**. Dans les **nœuds-et** se trouvent donc les conditions nécessaires à l'exécution du nœud père. Ses $Arbres-et/ou$ présentent en racine le but à réaliser et en feuilles les interactions pouvant être effectuées (les conditions d'exécution et la garde de distances sont satisfaites). Pour atteindre un état du monde, il faut effectuer au moins une des interactions présentes dans les **nœud-ou** fils. Pour effectuer une interaction, il faut satisfaire chaque condition présente dans tous les **nœud-et** fils.

Il existe un cas particulier dans l'$Arbre-et/ou$ qui est celui des déplacements générés par la garde de distance. Comme expliqué précédemment (dans la partie 2.1.3), lors de l'exécution d'une interaction, les déplacements générés par la garde de distance doivent être effectués après les interactions engendrées par la partie condition. C'est pourquoi, un **nœud-et** associé à une garde de distance est représenté par un **nœud-puis**. Un **nœud-puis** possède les mêmes propriétés qu'un **nœud-et** sauf que les interactions de sa branche seront exécutées qu'une fois que tous ses frères (les **nœud-et** de la même branche engendrées par la partie condition) ont été satisfaits.

Les déplacements des agents actifs sont calculés à l'aide de l'algorithme A^*. Cet algorithme nécessite d'avoir un coût attribué à chaque déplacement. Ce coût dans les simulations CoCoA peut représenter la difficulté pour atteindre une case afin de prendre en compte par exemple le relief de l'environnement ou les différents types de terrain.

2.2.5 La sélection d'interaction

Le planificateur fournit au module de sélection d'interaction l'ensemble d'interactions exécutables qui sont présentes en feuilles de l'arbre de planification. Les agents ne pouvant réaliser qu'une seule interaction à la fois, une heuristique de sélection d'interaction a été conçue. Cette heuristique sélectionne l'interaction dont la cible est la plus proche de l'agent. En cas d'égalité une interaction est sélectionnée aléatoirement parmi les interactions les plus proches. Cette heuristique a été choisie afin d'éviter des déplacements inutiles qui pourraient rendre le comportement moins crédibles.

Si nous considérons qu'une résolution optimale est l'exécution où l'agent résout tous ses buts tout en minimisant la distance totale parcourue (évitant donc des aller-retours qui peuvent être perçus comme non rationnel), alors l'heuristique utilisée peut être vue comme une résolution gloutonne privilégiant les plus courts chemins locaux (c'est-à-dire les plus faibles déplacements d'abord). Les heuristiques gloutonnes bien que ne garantissant pas toujours d'obtenir une résolution optimale, sont généralement meilleures qu'une résolution aléatoire. Cette heuristique permet d'amoindrir le surcoût en nombre d'aller-retours qu'il est nécessaire d'éviter dans la mesure du possible afin d'assurer une certaine crédibilité dans le comportement de l'agent.

2.3 La plateforme de simulation CoCoA

Maintenant que la présentation du fonctionnement de CoCoA est faite, nous pouvons passer à la présentation de la plateforme de simulation pour la conception de simulations à l'aide d'agents cognitifs situés. Afin de simplifier la création d'une simulation, la plateforme introduit la notion de **classes d'agent** permettant de typer les agents. Cette notion de type d'agent permet également de définir des **interactions spécifiques** à un type d'agent. Je présenterai donc ces deux notions avant de présenter la plateforme CoCoA qui se décompose en deux interfaces. Premièrement, l'interface de conception de simulations. Deuxièmement, l'interface d'exécution de simulation.

2.3.1 Les classes d'agent CoCoA

Comme dans un langage objet, les notions de classe d'agent et d'héritage ont été mises en place dans la plateforme de simulation CoCoA. Un agent est une instance de la classe d'agent `Actif` ou de la classe d'agent `Passif`. Un utilisateur peut définir ses propres classes d'agent en héritant de la classe `Actif` ou `Passif`. L'héritage permet la transmission des propriétés et des interactions subies et effectuées de la classe parent aux classes filles (même si elles sont déjà héritées d'une classe d'agent parent). Les agents qui sont des instances de ces classes obtiennent également les propriétés et les interactions de leur classe et peuvent en posséder d'autres qui seront spécifiques à l'instance.

Par exemple, un utilisateur peut définir un agent `Pomme1` qui est une instance de la classe `Pomme`. La classe `Pomme` peut subir l'interaction `manger` et possède une propriété `énergie`[5]. La classe `Pomme` hérite de la classe `Fruit` qui peut subir l'interaction `prendre` et possède une propriété `couleur`. La classe Fruit hérite de la classe `Passif` est possède une propriété `nom`. Ainsi l'agent `Pomme1` possède les propriétés `nom`, `couleur` et `énergie`, et peut subir les interactions `prendre` et `manger`.

2.3.2 Les interactions spécifiques

Puisqu'il est possible de définir des types d'agents, la plateforme offre la possibilité de définir des interactions spécifiques à un type d'agent cible (correspondant à la surcharge de

[5] manger ajoutera la valeur de la propriété énergie de la cible à la valeur énergie de l'agent effectuant l'interaction manger

méthodes dans un langage objet). Une interaction spécifique est une spécialisation d'une interaction déjà existante. L'utilisation principale de cette spécialisation est de pouvoir créer un comportement spécifique à un type d'agent particulier tout en gardant la sémantique de l'interaction d'origine.

Par exemple, prenons le cas de deux agents instances de la classe d'agent `Porte`. Le premier agent `porte1` est une porte qui peut subir l'interaction `ouvrir` et dont l'effet est de changer l'état de cible de `fermée` à `ouverte`. Le deuxième agent `porte2` est une porte dont la classe hérite de `Porte`. Cette classe nommée `PorteAClef` peut subir l'interaction `déverrouiller` (à condition que l'acteur possède la bonne clef) et dont l'effet est de changer l'état de cible de `verrouillée` à `déverrouillée`. Ainsi franchir `porte1` et franchir `porte2` ne correspondent pas aux mêmes interactions que l'agent doit exécuter. De plus, l'ordre d'exécution de ces deux interactions est important, il faut déverrouiller une porte avant de l'ouvrir. C'est pourquoi, il est possible de spécialiser l'interaction `ouvrir` pour les agents du type `PorteAClef`, afin que l'acteur la déverrouille puis l'ouvre pour la franchir.

La spécialisation d'une interaction se définit en précisant, pour l'interaction spécialisée, le nom de l'interaction parente.

nom :	open	nom :	openWithLock
		parent :	open
condition :	target.isOpen = false	condition :	target.isUnlocked = true
garde :	distance(actor, target) < 1	garde :	
action :	target.isOpen = true	action :	

Grâce à la spécialisation d'interaction, il est possible d'utiliser (par chaînage) l'interaction `unlock` pour déverrouiller une porte et ensuite d'utiliser l'interaction `openWithLock` pour ouvrir une porte sans altérer la sémantique de l'interaction `open`. Les portes sans clef subissent toujours l'interaction `open` qui n'a pas besoin d'être redéfinie.

nom :	unlock
condition :	target.isUnlocked = false
	actor.own(target.key)
garde :	distance(actor, target) < 1
action :	target.isUnlocked = true

2.3.3 La création d'une simulation CoCoA

Dans cette interface (voir la figure 2.10), il est possible de créer une simulation. Une simulation est composée d'un environnement, d'interactions, de classes d'agents et d'agents (instances de classe d'agents). Un environnement contient l'ensemble des places accessibles et la place de chaque agent au début de la simulation. Dans cette interface, il est possible d'utiliser des interactions préexistantes ou d'en créer de nouvelles. Il est également possible de créer des classes agents. Une classe d'agent possède des propriétés, des interactions qu'il peut subir et/ou effectuer (suivant que la classe "hérite" de la classe d'agent actif ou passif) et d'une description textuelle. Ensuite l'interface permet de créer des agents soit en utilisant une classe prédéfinie, soit en utilisant une classe que l'on vient de créer. Cette instance de classe d'agent possède les propriétés et les interactions de sa classe (il est possible de rajouter des interactions ou des propriétés propres à l'instance), elle représente un agent de la simulation

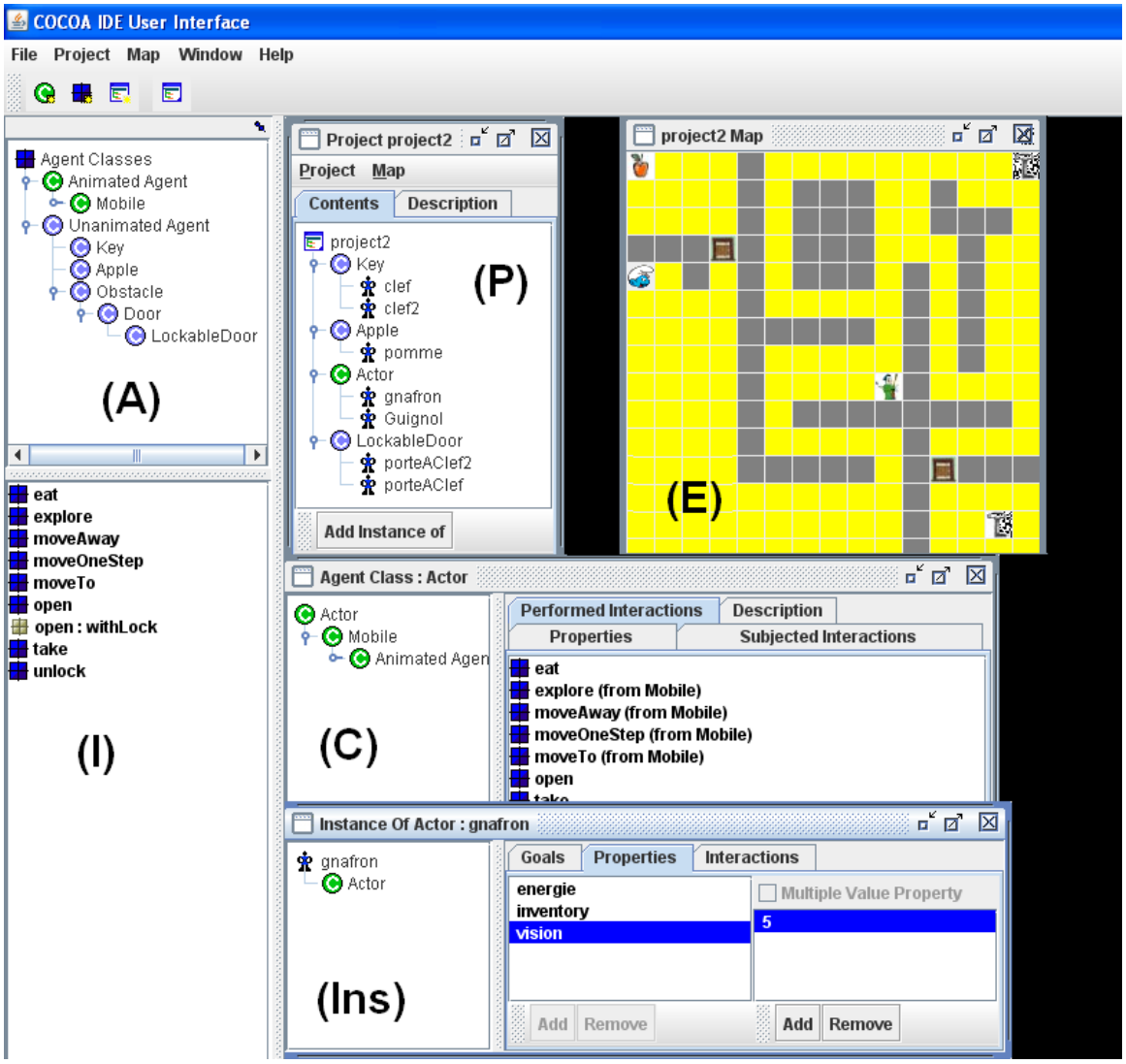


FIG. 2.10 – L'interface de création de simulation de CoCoA permet de définir, un projet (P) avec la liste des agents, des classes d'agents (A), et de définir les spécificités de chaque classe (C) ou de chaque agent (Ins) qui est une "instance" d'une classe d'agent, des interactions (I) et de définir l'environnement (E) avec la place de départ des agents et des zones non atteignables (en gris).

qu'il faut placer dans l'environnement. Si cet agent est actif, il possède également des buts qu'il doit réaliser, les buts sont créés pour chaque agent. Une fois la simulation complètement décrite, l'utilisateur peut passer à l'expérimentation.

2.3.4 L'exécution d'une simulation CoCoA

L'interface d'exécution d'une simulation CoCoA (voir la figure 2.11), présente l'environnement contenant tous les agents de la simulation.

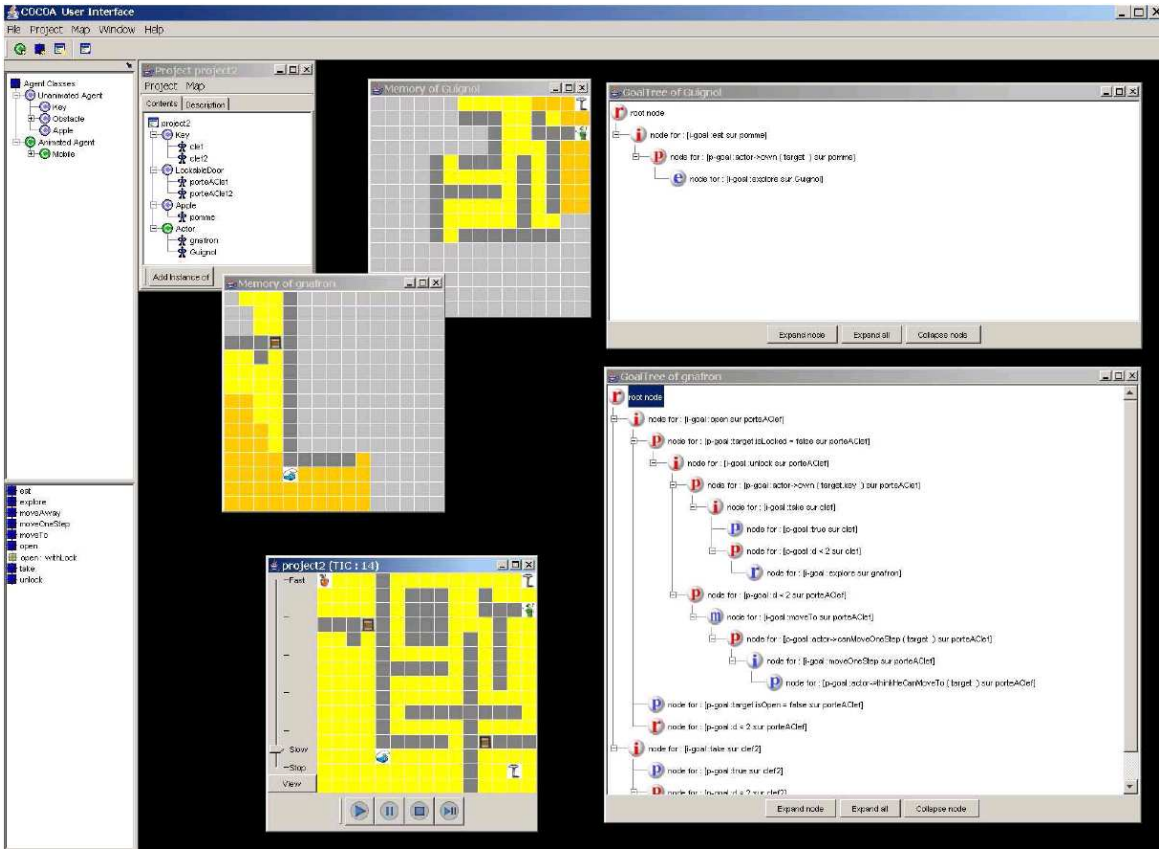


FIG. 2.11 – L'interface de simulation de CoCoA permet de voir l'environnement réel de la simulation, ainsi que les mémoires et les plans (les *Arbres – et/ou*) de chaque agent actif. Pour chaque nœud une lettre et une couleur sont associées, la couleur rouge représente un nœud non satisfait par le contexte actuel, un nœud bleu est satisfait. La lettre "i" représente une interaction à réaliser, "p" est une prémisse à satisfaire et "m" un déplacement à effectuer. À tout moment il est possible d'inspecter les valeurs de propriétés des agents de l'environnement.

Il est également possible dans cette fenêtre d'accéder aux propriétés de tous les agents de la simulation. Pour chaque agent actif, deux fenêtres sont accessibles, elles représentent l'état de la mémoire de l'agent ainsi que l'état de ses plans (avec un *Arbre – et/ou*). Dans les mémoires des agents, il est possible de voir l'état des autres agents et leur emplacement dans l'environnement (qui peuvent être différents de l'état réel des agents ou de leur emplacement réel dans l'environnement). La mémoire inclut également les informations perçues par l'agent. La perception des agents est représentée par un halo de vision, correspondant à un carré de taille fixe (la taille dépend de la propriété de vision de l'agent) centré sur l'agent (voir la figure 2.12).

Les plans présentent les nœuds satisfaits en bleu et les nœuds à satisfaire en rouge quelque soit le type du nœud. Il existe trois types de nœuds, les interactions (notés "i"), les prémisses (notés "p") et les déplacements (notés "m"). L'arbre de planification évolue au cours de la simulation en fonction des interactions effectuées et des connaissances de l'agent.

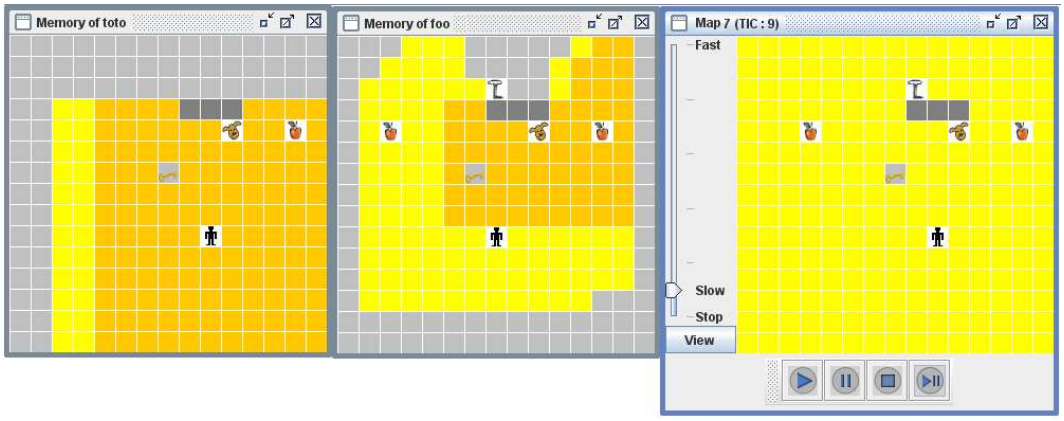


FIG. 2.12 – Les agents actifs CoCoA ont une perception limitée de leur environnement. Les deux fenêtres de gauche représentent la mémoire et la perception de l'agent `toto` et de l'agent `foo`, à droite c'est l'état réel de l'environnement. Dans cette simulation, l'agent `toto` (à gauche) possède une représentation graphique d'un bonhomme noir, l'agent `foo` (au centre) possède une représentation graphique d'un animal. Dans les deux mémoires sont indiquées en gris clair les zones inconnues de l'environnement, en orangé les zones dans le champ de perception de l'agent et le reste (les places en jaune et les obstacles en gris foncé qui sont de la même couleur que l'environnement) sont les zones déjà visitées et présentes dans la mémoire de l'agent. Dans cette illustration l'agent `toto` a un champ de perception de 5 cases, l'agent `foo` a un champ de perception de 4 cases. Nous pouvons remarquer que l'emplacement de `toto` dans la mémoire de `foo` n'est pas la bonne, il correspond au dernier emplacement mémorisé, car perçu, par `foo`.

2.4 Les fichiers de configuration

Un projet CoCoA est composé de 5 fichiers XML de configuration : Classes, Env, Interaction, Map, Project. Ces fichiers permettent de définir séparément les informations déclaratives et réutilisables (les classes d'agent, les interactions) et les informations liées à l'exécution de la simulation (l'environnement, la place de chaque agent, les instances d'agents avec leurs valeurs spécifiques pour la simulation).

Map définit la description de l'environnement sous la forme d'une graphe avec des places et d'arcs ;

Env définit l'emplacement de chaque agent dans l'environnement ;

Interaction définit l'ensemble des interactions pour la simulation ;

Classes définit les classes d'agents (les "types") ;

Project définit les instances d'agents.

Ainsi un agent est défini dans le fichier `Project.xml` (voir la figure 2.15), la définition de son type est effectuée dans le fichier `Classes.xml` (voir la figure 2.14) et la définition de ses interactions sont dans le fichier `Interaction.xml` (voir la figure 2.13).

```
<Interaction>
   <Name>moveAway</Name>
   <Garde />
   <Condition>actor.canMoveAwayFrom(target)</Condition>
   <Action>moveAway(target)</Action>
</Interaction>
<Interaction>
   <Name>moveTo</Name>
   </Information>
   <Garde />
   <Condition>actor.canMoveTo(target)</Condition>
   <Action>moveTo(target)</Action>
</Interaction>
```

FIG. 2.13 – Exemple de fichier contenant la définition des interactions. Les interactions `open`, `eat` et `take` ont déjà été présentés dans la figure 2.6.

```
<Animated>
   <Name>MyActor</Name>
   <Property>
      <Name_Property>inventory</Name_Property>
      <Value>collection</Value>
   </Property>
   <Heritage>Mobile</Heritage>
   <Performed_Interaction>
      <InteractionName>open</InteractionName>
   </Performed_Interaction>
   <Performed_Interaction>
      <InteractionName>take</InteractionName>
   </Performed_Interaction>
   <Performed_Interaction>
      <InteractionName>eat</InteractionName>
   </Performed_Interaction>
   <Icon>fr\lifl\smac\simulation\editor\icon\default_actor.jpg</Icon>
</Animated>
```

FIG. 2.14 – Exemple d'une classe d'agent nommé `MyActor`, cette classe peut effectuer les interactions `open, take` et `eat`, ainsi que les interactions `explore, moveTo` et `moveAway` (héritées de la classe `Mobile`).

2.5 Conclusion

Le projet CoCoA permet la création d'agents cognitifs situés dans un environnement en se basant sur l'approche centrée interaction. Les agents sont définis comme des entités qui peuvent subir ou effectuer des interactions et qui possèdent des propriétés permettant de caractériser leur état. Les agents sont cognitifs, proactifs et leur comportement est dirigé par des buts. Ils n'ont aucune connaissance a priori de leur environnement ou des autres agents de la simulation. Les agents sont situés dans un environnement et cet aspect a un impact sur la planification des résolution de leurs buts. Les interactions sont décrites de manière "universelle", elles sont réutilisables, tout comme les agents dont la description ne dépend pas

```
<Instance>
   <Name>tony</Name>
   <Name_Class>MyActor</Name_Class>
   <Goal>
      <InteractionGoal>
         <Interaction>take</Interaction>
         <Target>clef2</Target>
      </InteractionGoal>
   </Goal>
   <Property>
      <Name_Property>inventory</Name_Property>
      <Value>collection</Value>
   </Property>
   <Icon>fr\lifl\smac\simulation\editor\icon\Blesse.gif</Icon>
</Instance>
```

FIG. 2.15 – Exemple d'instance de la classe `MyActor`, l'agent `tony` a comme but d'effectuer l'interaction `take` sur l'agent `clef2`.

d'une simulation. Le projet CoCoA met en avant une conception particulière des systèmes multi-agents et permet de la mettre en œuvre au travers de simulations à l'aide sa plateforme de simulation éponyme.

Dans CoCoA, le choix de l'interaction à exécuter parmi l'ensemble des interactions exécutables était délégué à une heuristique prenant en compte l'aspect situé des simulations. Ainsi, le module de sélection d'interaction (présenté dans la (partie 2.2.5)) privilégiait l'interaction dont la distance entre la source et la cible de l'interaction est la plus faible. Cette heuristique bien que réduisant le surcoût en nombre d'aller-retours n'était pas suffisante pour définir le comportement d'un agent. En effet, comme je le présenterai dans le chapitre suivant, le choix des actions à exécuter fait partie intégrante de la définition du comportement des agents. Puisque l'heuristique de sélection d'interaction (basée uniquement sur la proximité des agents cibles) amène à un comportement réducteur de l'agent, seul le choix des interactions que l'agent peut-effectuer permettait réellement d'obtenir une différence de comportements entre les agents. Dans le chapitre suivant, j'expliquerai que le comportement d'un agent n'est pas uniquement défini par ses capacités (même si un agent ne pouvant effectuer que des interactions violentes comme casser ou tuer semblera avoir un comportement violent) mais également par le choix des interactions qu'il va effectuer, c'est-à-dire le mécanisme de sélection d'interaction.

Cette thèse se place dans un contexte déjà bien élaboré et dont l'influence sur mes travaux est perceptible. Par exemple, je me suis inspiré de l'aspect réutilisable et générique des interactions pour concevoir un système réutilisable et générique pour la partie individualité du comportement. L'intégration de mon travail dans CoCoA devant enrichir le projet, je me devais de garder ses points forts. Toutefois, bien que mes travaux apportent une contribution au projet CoCoA, ils sont utilisables en dehors, c'est pourquoi ils seront présentés en dehors de CoCoA et du modèle centré interaction.

Chapitre 3

Modélisation du comportement

« L'avantage d'être intelligent, c'est qu'on peut toujours faire l'imbécile, alors que l'inverse est totalement impossible. »
Woody Allen

Le but de mon travail est de proposer une approche pour la conception de comportements. Dans cette partie, je me focaliserai sur des agents cognitifs situés dont le comportement est dirigé par des buts préalablement définis. Néanmoins l'aspect situé n'est qu'un facteur à prendre en compte pour le comportement et n'est pas un élément déterminant de l'approche. De plus, cette approche n'est pas dédiée aux agents cognitifs et je monterai qu'il est également possible d'utiliser ma proposition pour des agents réactifs.

Une des cibles possibles de la conception de comportement est l'industrie de jeux vidéo, qui est la "killer application" selon J. Laird[LvL00]. Toutefois, dans les jeux vidéo, la conception de PNJ et les techniques d'Intelligence Artificielle utilisées sont en partie basées sur les langages de script ou sur des automates du type FSM (voir la partie 1.3). Ma proposition permet la conception d'un moteur comportemental basé sur les motivations, robuste et évolutif dont l'utilisation ne nécessite pas de grandes connaissances en intelligence artificielle. Le rôle de chaque partie étant défini, les calculs provenant des motivations sont interprétables et ne constituent pas une boite noire.

Dans un premier temps j'expliquerai que le modèle pour la conception de comportement que je propose est divisé en deux parties appelées le raisonnement et l'individualité. Puis je me focaliserai sur la partie individualité qui est gérée par un mécanisme de sélection d'action. Ce mécanisme est basé sur la notion de motivation et nous verrons qu'il prend également en compte la notion d'alternative afin d'assurer une certaine crédibilité dans le comportement. Je montrerai également qu'il est possible de personnaliser le comportement de l'agent à l'aide d'un profil d'individualité. Enfin, je présenterai la méthode de conception de comportement que je préconise et je proposerai l'illustration concrète d'une motivation en suivant les étapes de conception de cette méthode.

3.1 Le modèle de conception de comportement

3.1.1 Le raisonnement et l'individualité

Le comportement d'un agent (ou d'un personnage non joueur) est défini par l'observation des actions qu'il effectue dans l'environnement. De cette affirmation, nous pouvons extraire deux éléments participant à la définition du comportement. Premièrement, les actions exécutées sont les actions parmi les capacités des agents, pour lesquelles le contexte (l'environnement) était favorable à leur exécution. Si un agent n'est capable d'effectuer que des actions violentes, les actions qu'il exécutera dans l'environnement seront violentes et le comportement sera défini comme violent. J'appellerai **raisonnement**, la partie du comportement qui décrit les actions que pourra exécuter l'agent parmi ses capacités et en fonction de l'état de l'environnement connu (c'est-à-dire les connaissances sur l'état de l'environnement). Deuxièmement, si l'agent peut, à instant donné, effectuer plusieurs actions car il possède une palette d'actions suffisamment large pour que plusieurs actions soient exécutables, alors l'agent a un choix à effectuer. Ce choix doit être fait en fonction de son comportement et plus précisément des traits caractères qu'on lui a assigné, c'est-à-dire son **individualité**.

La séparation entre le raisonnement et l'individualité n'est pas courante dans la littérature. Dans les architectures classiques (réseaux de neurones, réseaux de Petri, etc.) la partie raisonnement prend la décision de l'action à exécuter, car la partie individualité est incluse dans le raisonnement. J'ai choisi de séparer les deux éléments car ce sont deux parties indépendantes qui définissent le comportement de manières différentes et leur construction ne repose pas sur les mêmes éléments. Le raisonnement permet de définir **comment un agent peut faire ce qu'il doit faire** (pour résoudre ses buts par exemple), selon les capacités de l'agent et l'état de son environnement. Cette procédure est commune à tous les agents et n'est pas suffisante pour définir un comportement "propre" à l'agent, dans le sens où il n'a pas le choix, le comportement est mécanique. Lorsque l'agent a plusieurs possibilités d'actions à exécuter, il faut définir **comment l'agent choisit de faire ce qu'il doit faire**. Ce choix se base sur des traits de sa personnalité qui rendent son comportement plus personnel exprimant ainsi une individualité.

Bien que mon travail vienne compléter le projet CoCoA et que l'approche centrée interaction assure la réutilisation et une représentation générique d'une partie du comportement (i.e. la partie raisonnement du comportement), mon modèle se veut également utilisable dans d'autres plateformes. C'est pourquoi, je présenterai les définitions des éléments importants de mon modèle afin qu'il puisse être exploiter ou adapter en dehors de CoCoA. Je parlerais donc d'action là où dans CoCoA on parlerait d'interaction.

3.1.2 Le raisonnement

Le raisonnement est la partie du comportement qui détermine comment l'agent **peut** résoudre ses buts. Cette partie du comportement dépend des connaissances de l'agent ainsi que de ses capacités (i.e. les interactions qu'il peut effectuer). Plus l'agent a une connaissance précise de l'environnement dans lequel il se trouve, mieux il est capable de déterminer comment il lui est possible de résoudre ses buts. Les capacités de l'agent vont déterminer quelles sont les actions qu'il peut effectuer afin de résoudre ses buts. Plus un agent a de capacités, plus il

est en mesure de résoudre ses buts de différentes façons. À l'inverse, moins un agent possède de capacités, plus il sera contraint dans sa résolution.

Les actions exécutables

À chaque étape de la simulation le raisonnement calcule l'ensemble des actions que l'agent peut effectuer immédiatement afin d'atteindre l'étape suivante de la résolution.

Ces actions sont possibles car le contexte, c'est-à-dire l'environnement, est dans un état qui permet l'exécution de l'action. Par exemple, pour ouvrir une porte, l'agent doit être proche de la porte et la porte doit être dans l'état fermé. De telles actions sont appelées dans la suite actions exécutables.

Définition 3.1 (Action exécutable) *Pour un environnement $\mathcal{E}$ donné, une action exécutable α est une action telle que $\mathcal{E}$ est dans un état valide pour l'exécution de cette action. Un état est valide pour l'exécution d'une action lorsqu'il satisfait toutes les conditions d'exécution de l'action.*

Les alternatives

Le raisonnement doit déterminer comment l'agent peut résoudre ses buts, c'est-à-dire déterminer la séquence d'actions à effectuer à partir d'un état initial jusqu'à l'état final recherché, c'est-à-dire le plan. Parfois les plans sont représentés en utilisant des disjonctions. Par exemple, le plan peut être "*pour résoudre le but g_i faire l'action a_1 ou l'action a_2*". Si nous considérons des plans sans disjonction, l'exemple précédent représente deux plans pour le même but :

Plan 1 : "*pour résoudre le but g_i faire l'action a_1*",

Plan 2 : "*pour résoudre le but g_i faire l'action a_2*".

Afin d'éviter la confusion, j'appelle ce genre de plan sans disjonction une **alternative**. Bien que le mot alternative désigne une possibilité entre deux situations, il est couramment utilisé pour désigner une seconde option. Dans notre cas, pour un but donné il peut y avoir plusieurs façons de le résoudre, une alternative désigne chacune de ces possibilités.

Définition 3.2 (Alternative,Plan) *Une **alternative** est un triplet $(g, \alpha, (a_i)_{i\in[1,n]})$ où*
- *g est un but,*
- *α est une action exécutable*
- *$\forall i \in [1, n], a_i$ est une action et $(\alpha, a_1, \ldots, a_n)$ est une séquence d'actions à effectuer pour résoudre g.*

Les trois parties de l'alternative peuvent être caractérisées comme cela (voir Figure 3.1) :

- Le but g est ce que l'agent **veut** : c'est l'état que l'agent cherche à atteindre une fois qu'il aura exécuté toutes les actions qu'il a planifiées.
- L'action exécutable α est ce que l'agent **peut faire**, immédiatement.
- La séquence d'actions (a_i) est ce que l'agent **prévoit de faire** : les actions que l'agent a planifiées de faire pour résoudre son but une fois que l'action α aura été exécutée.

$$\underbrace{\alpha}_{\text{peut faire}} \rightarrow \underbrace{a_1 \rightarrow \cdots \rightarrow a_n}_{\text{prévoit de faire}} \Rightarrow \underbrace{g}_{\text{veut}}$$

FIG. 3.1 – Une alternative est composé de trois parties : l'action exécutable α, les actions planifiées(a_i) et le but recherché g.

Planificateur

Bien que le but de ma proposition est de pouvoir construire des comportements variés sans avoir nécessairement de grandes connaissances en I.A, les comportements que je souhaite obtenir doivent être crédibles. Un élément de crédibilité est que mes agents se comportent de manière rationnelle. Suivant la définition de Russel et Norvig [RN03], je considère qu'un agent est rationnel s'il fait "la bonne chose à faire" en fonction de ce qu'il sait. Dans notre cas, cela veut dire que le plan de l'agent doit être cohérent avec les connaissances de l'agent et non pas avec l'état réel de l'environnement. En effet, nos agents ne sont pas omniscients, ils ont une perception locale de l'environnement (et des autres agents de l'environnement) et ils construisent leur base de connaissances à partir de ce qu'ils perçoivent. Cette base de connaissances est appelée mémoire (aussi parfois appelée base de croyances dans la littérature). À partir de ce constat, nous pouvons dire que le raisonnement est constitué d'un planificateur défini comme ceci :

Définition 3.3 (Planificateur) *Ètant donné un personnage (un agent) c, son ensemble de buts $\mathcal{G}$, son ensemble de capacités $\mathcal{A}$ et sa mémoire (base de connaissances) $\mathcal{KB}$, un* ***planificateur*** *est un processus qui selon les informations contenues dans $\mathcal{KB}$, calcule tous les alternatives $\mathcal{A}lt_c$ résolvant les buts $\mathcal{G}$ en utilisant les actions $\mathcal{A}$.*

Alors, si $\mathcal{C}$ désigne l'ensemble des personnages et $\mathcal{A}lt$ l'ensemble de toutes les alternatives possibles, un planificateur peut être vu comme l'application[6] *:*

$$\begin{array}{rccc} Planificateur : & \mathcal{C} & \rightarrow & \mathcal{P}(\mathcal{A}lt) \\ & c = (\mathcal{G}, \mathcal{A}, \mathcal{KB}) & \mapsto & \mathcal{A}lt_c \end{array}$$

Bien que le raisonnement utilise un planificateur, nous ne faisons pas de nouvelle proposition dans ce domaine. Il existe déjà de nombreux planificateurs, permettant notamment l'anticipation, ou la remise en cause de connaissances [Wel99]. N'importe quelle solution qui correspond à la définition 3.3, i.e. fournir un ensemble d'alternatives, peut convenir à mon modèle. Dans ma proposition, ce qui est important, c'est que le raisonnement fournisse à l'individualité les alternatives lui permettant de faire un choix sur des prévisions à moyen-terme. Nous verrons dans la suite comment nous avons fait évoluer le planificateur de CoCoA afin qu'il fournisse les alternatives nécessaires.

3.2 L'individualité

À partir du moment où l'agent a le choix parmi les actions qu'il peut effectuer, il faut déterminer un moyen de sélectionner l'action à exécuter à chaque instant. Nous appelons individualité la partie du comportement qui détermine **comment l'agent choisit** de résoudre ses

[6] où, si $\mathcal{S}$ est un ensemble, $\mathcal{P}(\mathcal{S})$ désigne l'ensemble des parties de $\mathcal{S}$: $\mathcal{P}(\mathcal{S}) = \{\sigma \mid \sigma \subseteq \mathcal{S}\}$.

buts. Ce choix permet à l'individualité d'exprimer des traits de la personnalité de l'agent. La personnalité ne se définit pas par rapport à l'environnement dans lequel l'agent se trouve. La personnalité peut s'exprimer différemment suivant l'environnement (notamment par la limite imposée par la partie raisonnement du comportement), mais sa définition ne change pas. C'est pourquoi, l'individualité d'un agent est définie sans connaissance a priori de l'environnement dans lequel l'agent sera utilisé. Si la définition de l'individualité ne dépend pas de l'environnement cela signifie qu'elle peut être utilisée quelque soit l'environnement. Je propose donc de définir l'individualité indépendamment de l'environnement d'exécution afin qu'elle puisse être réutilisable (entièrement ou en partie) pour d'autres agents et d'autres simulations.

L'individualité est décrite par les actions que l'agent a choisi d'exécuter dans l'environnement. Par définition, un mécanisme de sélection d'action choisit les actions à effectuer, c'est pourquoi j'affirme que c'est à travers ce mécanisme que l'individualité est définie. En effet, les choix effectués par le mécanisme permettent l'exécution d'actions dans l'environnement, ces dernières étant factuelles, elles forment la deuxième partie du comportement observable. La partie individualité du comportement suit le raisonnement et sa tâche est donc de sélectionner une action parmi les actions exécutables calculées par le raisonnement pour résoudre les buts de l'agent. Ainsi, deux agents dans le même environnement, qui ont les mêmes capacités, les mêmes buts à réaliser, les mêmes connaissances et le même moteur de raisonnement, peuvent se comporter différemment grâce au mécanisme de sélection d'action qui permet d'exprimer leur individualité. Pour cela j'utiliserai un mécanisme de sélection d'action basé sur les motivations et les alternatives. Les motivations définiront les traits de la personnalité de l'agent qui influenceront le choix de l'agent. Les alternatives permettront aux motivations d'avoir une prévision à moyen terme afin d'orienter au mieux le choix de l'agent. Les motivations seront définies de manière à être ajoutées et supprimées sans que cela ne perturbe l'agent, notamment grâce à leur indépendance à l'environnement. J'applique ainsi aux motivations, l'idée de "plug-and-play" que l'on peut retrouver avec les interactions.

3.2.1 Le mécanisme de sélection d'action

Définition

Le mécanisme de sélection d'action (ASM pour *Action Selection Mechanism*) a la charge de sélectionner une action parmi l'ensemble des actions exécutables $\mathcal{A}^R$ identifiées par le raisonnement. Classiquement, un ASM affecte à chaque action exécutable appartenant à $\mathcal{A}^R$ une valeur numérique et sélectionne l'action possédant la plus grande valeur. La valeur est calculée comme une combinaison de *critères d'évaluations* ou *évaluateurs*. Chaque évaluateur e_i est défini par une fonction γ_{e_i} :

$$\begin{array}{rccc} \gamma_{e_i}: & \mathcal{A}^R & \rightarrow & \mathbb{R} \\ & a & \mapsto & valeur \end{array}$$

Pour chaque action a dans $\mathcal{A}^R$, la note finale ϕ utilise une fonction de combinaison $Comb$ pour agréger les n-évaluateurs :

$$\begin{array}{rccc} Comb: & \mathbb{R}^n & \rightarrow & \mathbb{R} \\ \phi: & \mathcal{A}^R & \rightarrow & \mathbb{R} \\ & a & \mapsto & Comb(\gamma_{e_1}(a), \cdots, \gamma_{e_n}(a)) \end{array}$$

Classiquement, l'ASM sélectionne l'action exécutable α ayant obtenu la meilleure (la plus grande) valeur :

$$ASM(\mathcal{A}^R) = \alpha \text{ où } \alpha \in \mathcal{A}^R \mid \phi(\alpha) = \max_{a \in \mathcal{A}^R}\{\phi(a)\}$$

Voici la définition que nous donnons pour le mécanisme de sélection d'action.

Définition 3.4 (Mécanisme de sélection d'action) *Soit $\mathcal{A}$ l'ensemble des actions, et ϕ une fonction :*

$$\begin{array}{rccc} \phi: & \mathcal{A} & \rightarrow & \mathbb{R} \\ & a & \mapsto & valeur \end{array}$$

alors, le ***mécanisme de sélection d'action*** *est défini comme l'application :*

$$\begin{array}{rccc} ASM: & \mathcal{P}(\mathcal{A}) & \rightarrow & \mathcal{A} \\ & \mathcal{A}^R & \mapsto & \arg\max_{a \in \mathcal{A}^R}(\phi(a)) \end{array}$$

Soit $\mathcal{E}$ un environnement, $\mathcal{A}$ l'ensemble des actions possibles dans $\mathcal{E}$, c un agent (ou un personnage) situé dans $\mathcal{E}$, et $\mathcal{A}^R(\subset \mathcal{A})$ l'ensemble des actions exécutables pour c dans $\mathcal{E}$ au moment t, alors l'ASM fournit à c la prochaine action α à exécuter dans $\mathcal{E}$ au moment t.

3.2.2 Le mécanisme de sélection d'action basé sur les motivations

Le rôle de l'individualité est donc d'appliquer la fonction ϕ de la définition 3.4. L'individualité de l'agent est influencée par des tendances calculées à partir de motivations. Ainsi, ϕ est une fonction qui combine ces influences, chaque motivation étant exprimée à travers une fonction appelée *évaluateur*. Un évaluateur fournit une note (une évaluation) à chaque action exécutable. Cette note exprime l'influence de la motivation sur cette action exécutable. Néanmoins cette dernière affirmation doit être précisée.

L'importance des alternatives

L'ASM sélectionne la meilleure action, cette dernière ayant obtenu la meilleure évaluation. Cette évaluation résulte de la combinaison de toutes les évaluations des motivations (et de leur évaluateur). Cependant, il ne faut pas oublier qu'une action exécutable appartient à une ou plusieurs alternatives calculées par le planificateur. Les autres actions dans l'alternative correspondant à "ce que l'agent prévoit de faire". Ces actions contiennent la prévision (pas l'anticipation) sur les actions que l'agent aura à effectuer. Afin que l'agent ait un comportement plutôt réaliste, il est nécessaire de prendre en compte ces actions. En effet, le mécanisme de sélection d'*action* doit être considéré comme un mécanisme de sélection d'*alternative*[7], puisque le choix de l'action exécutable α détermine le prochain but et donc le prochain plan considéré par les agents. Évidemment, il est possible d'avoir plusieurs buts et donc plusieurs résolutions pour la même action exécutable mais cela ne change pas ce qui vient d'être dit.

Un critère important d'un bon mécanisme de sélection d'action défendu par Tyrrell [Tyr93b], est qu'il doit éviter les oscillations pour paraître réaliste. En effet si un agent

[7] l'acronyme en anglais ASM fonctionne également pour Alternative Selection Mechanism.

hésite entre deux résolutions sans pouvoir en terminer une seule, le comportement paraitrait incohérent ou trop indécis pour être réaliste. De même, une résolution ne doit pas être entamée pour ensuite être totalement écartée pour une autre sans autant que la première soit réalisée. Pour cela, les évaluateurs, ou au moins un des évaluateurs du mécanisme, ne doivent pas uniquement considérer α mais l'ensemble de l'alternative pour calculer la valeur à assigner à α.

Pour illustrer cela, considérons deux alternatives (ou plans) qui partagent le même but, mais chacune correspond à des actions exécutables différentes, $p_1 = (g, \alpha_1, (a_i^1)_{i\in[1,n_1]})$ et $p_2 = (g, \alpha_2, (a_i^2)_{i\in[1,n_2]})$. L'ASM doit tenir compte de l'ensemble des actions impliquées dans p_1, resp. p_2, pour promouvoir α_1, resp. α_2. Supposons, que l'ASM privilégie α_1 qui à court terme semble préférable, cela signifie que l'agent s'engage dans la résolution p_1 et en traite les actions a_i^1. Il ne faut pas qu'après quelques pas de résolution de p_1, cet agent se trouve confronté à une action de a_k^1 que l'ASM rejetterait pour réorienter l'agent sur p_2, suite à une inhibition sur cette action par une motivation, par exemple. Dans une telle situation, l'ASM doit immédiatement favoriser α_2, et donc p_2. Ceci peut être fait uniquement en prenant en compte l'ensemble de l'alternative pendant l'évaluation de l'action exécutable. L'utilisation de l'alternative dans l'évaluation des actions exécutables guide l'ASM vers a_2, et donc vers α_2, même si à court terme, α_1 semblait la meilleure action à exécuter. L'ASM doit prendre en compte le plan à moyen terme, en considérant toutes les actions qui apparaissent dans l'alternative. Il en résulte que la sélection d'action ne doit pas se focaliser sur le choix de l'action préférée à chaque pas, mais sur la meilleure résolution, c'est-à-dire la meilleure alternative.

De plus le mécanisme de sélection d'action combinant les évaluations des motivations sur les alternatives que je propose correspond aux critères de sélection de la meilleure action lorsqu'on considère l'ensemble des évaluations simultanément. L'action sélectionnée n'est donc pas nécessairement celle qui correspond le mieux à une motivation particulière, comme dans un "winner-takes-all". Notre sélection se rapproche plus à une sélection de la meilleure candidate du compromis comme le défini Tyrrell [Tyr93b]. En effet, les évaluations sont combinées pour ne donner qu'une seule valeur reflétant l'évaluation globale de l'action par l'ensemble des motivations. Nous verrons dans la partie 3.3.1, que la fonction de combinaison *Comb* a été pensée afin de respecter cette caractéristique du mécanisme de sélection d'action.

Pour assurer le rôle de la prise en compte de l'alternative, il faut que sur l'ensemble des motivations, chaque partie de l'alternative soit évaluée (voir la figure 3.1). Néanmoins le choix des parties à évaluer dépend en premier lieu de la définition même de la motivation. En effet, certaines motivations prennent en compte l'action exécutable, c'est-à-dire *ce que l'agent peut faire*, d'autres le but (*ce que l'agent doit faire*) et d'autres encore *ce que l'agent prévoit de faire* et *ce que l'agent peut faire*. Évidemment, chaque motivation ne doit pas prendre en compte la totalité de l'alternative, mais je préconise que l'ensemble des motivations utilisées le fasse.

Maintenant que l'importance de l'alternative est claire, je peux définir une motivation et son évaluateur comme une fonction qui évalue les alternatives :

Définition 3.5 (Motivation, Évaluateur) *Soit $\mathcal{A}lt$ un ensemble d'alternatives. Une **motivation** est définie par une fonction γ, appelée **évaluateur** :*

$$\begin{array}{rccc} \gamma : & \mathcal{A}lt & \rightarrow & \mathbb{R} \\ & alt = (g, \alpha, (a_i)_{i\in[1,p]}) & \mapsto & r \end{array}$$

Alors, un ASM (action/alternative selection mechanism) basé sur les motivations est défini comme (voir figure 3.2) :

Définition 3.6 (ASM basé sur les motivations) *Soit $\mathcal{A}$ un ensemble d'actions, $\mathcal{A}lt$ l'ensemble de toutes les alternatives possibles. Un* ***ASM basé sur les motivations*** *est une paire $(Comb, \Gamma)$ où $Comb$ est une fonction de $\mathbb{R}^n$ dans $\mathbb{R}$ et $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$ est un ensemble de motivations. Alors, la fonction ϕ peut être définie par :*

$$\begin{array}{rccl} \phi: & \mathcal{A}lt & \rightarrow & \mathbb{R} \\ & alt & \mapsto & Comb(\gamma_1(alt), \cdots, \gamma_n(alt)) \end{array}$$

et un ASM basé sur les motivations est défini comme l'application :

$$\begin{array}{rccl} ASM: & \mathcal{P}(\mathcal{A}lt) & \rightarrow & \mathcal{A} \\ & \mathcal{A}lt_i & \mapsto & \alpha \end{array}$$

où $alt = (g, \alpha, (a_i)_{[1,p]}) = \arg\max_{alt \in \mathcal{A}^R}(\phi(alt))$. $Comb$ est appelée la fonction de combinaison.

Donc, soit c un agent (ou un personnage), si $Planificateur(c) = \mathcal{A}lt_c$ alors $ASM(\mathcal{A}lt_c) = \alpha$ est la prochaine action que c va exécuter.

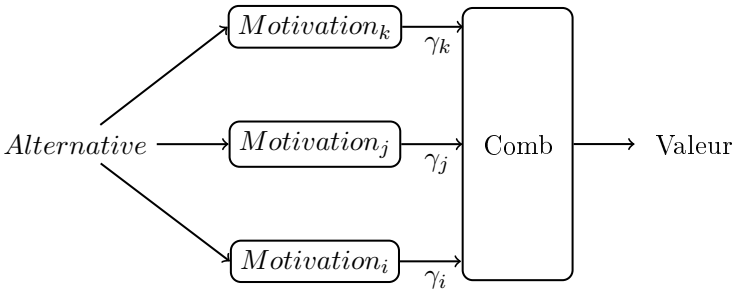


FIG. 3.2 – Un ASM basé sur les motivations utilise une combinaison des évaluateurs (motivations) pour sélectionner la meilleure action à partir des alternatives fournies par le planificateur.

Le mécanisme de sélection d'action permet donc, à partir des motivations, de choisir les actions à exécuter suivant son individualité. L'avantage de cette approche est que chaque évaluateur est une représentation d'une motivation. Il n'y a donc pas d'effet "boite noire" que l'on peut retrouver dans certaines architectures (comme les réseaux de neurones). Chaque valeur a un sens qui peut être interprété à l'aide de la définition de la motivation. C'est pourquoi dans la partie 3.3.1, je préconise de définir en premier les motivations puis à partir de cette définition de construire les évaluateurs.

3.2.3 Le profil d'individualité

Maintenant que ces définitions ont été établies, de la même manière que le raisonnement utilise le même planificateur pour chaque agent, l'individualité utilise le même mécanisme de

sélection d'action pour chaque agent. Ce qui veut dire que tous les agents utilisent la même fonction de combinaison *Comb* et le même ensemble de motivations Γ. Ceci implique que l'individualité résulte de la définition des motivations, leurs évaluateurs et de la fonction de combinaison *Comb*. Cette construction est la première tâche du concepteur de comportement, mais ce travail peut être effectué une fois pour chaque agent et quelque soit les simulations ou les jeux où ce comportement sera utilisé. En effet, ce qui change d'un agent à un autre c'est comment l'agent, son comportement pour être précis, est influencé par chaque motivation. Des différences dans ces influences constituent des différences dans les comportements. Alors d'un agent à l'autre, en dehors de la partie raisonnement, un changement de comportement est dû à la façon dont l'agent est affecté par les motivations.

Pour exprimer ces différences dans l'individualité des agents, les évaluateurs, γ_i sont définis comme étant des fonctions paramétrées, dont les valeurs de paramètre sont définies pour chaque agent. Ces paramètres expriment comment une motivation donnée influence le comportement de l'agent. Pour chaque γ_i, cet ensemble de paramètres est noté π_{γ_i}. Il est important de ne pas voir ce paramètre comme le poids de l'évaluateur. En effet chaque évaluateur est défini indépendamment et aucune motivation ne définit à elle seule le comportement de l'agent. De plus, ce paramètre n'a aucune influence sur l'expression des autres motivations, il n'a d'impact que sur sa motivation.

Ayant le même planificateur (raisonnement) et le même mécanisme de sélection d'action, les mêmes fonctions de combinaison et le même ensemble de motivations, des valeurs distinctes de ses paramètres permettent à deux agents ayant les mêmes capacités et les mêmes connaissances d'agir différemment dans le même environnement. Ainsi, le profil d'individualité d'un agent est défini par :

Définition 3.7 (Profil d'individualité) *Soit c un agent (ou un personnage),* $(Comb, \Gamma)$ *un ASM basé sur les motivations,* Π *l'ensemble* $\cup_{i\in[1,|\Gamma|]}\pi_{\gamma_i}$*, le* ***profil d'individualité*** *de c est l'ensemble des valeurs affectées pour c des éléments de* Π*.*

Il est possible d'avoir des fonctions qui fabriquent ces profils. Ces fonctions s'appliquent de l'ensemble des agents dans $\mathbb{R}^{|\Pi|}$ et donnent son profil d'individualité à un agent. La définition de ces fonctions est la seconde tâche du concepteur de comportement.

L'expression des motivations par le profil d'individualité rejoint l'idée défendu par J.P. Donckèle présentée dans la partie 1.1. L'auteur suppose que les humains expriment tous les six profils, mais dans des proportions différentes, certains profils étant plus dominants que d'autres. Cela rejoint l'idée du profil d'individualité où les mêmes motivations vont influencer le comportement de l'agent, mais selon l'agent dans des proportions différentes.

3.3 Construction et Illustration

3.3.1 Construction de l'ASM

Dans cette partie, je propose une méthode pour concevoir un ASM basé sur les motivations. La mise en place de notre ASM se décompose en trois étapes :

1. identifier toutes les motivations désirées et pertinentes pour le contexte applicatif, l'ensemble Γ,
2. choisir la fonction de combinaison $Comb$ qui sera utilisée,
3. définir un évaluateur γ_i pour chaque motivation dans Γ.

Les étapes 1 et 2 sont indépendantes l'une de l'autre. Mais la fonction de combinaison sélectionnée doit être prise en compte dans l'étape 3.

Étape 1, identifier les motivations

Cette première étape est plutôt conceptuelle, elle consiste à déterminer, et si possible nommer, les motivations qui vont influencer le comportement de l'agent. Pour cela, je préconise de séparer les motivations suivant le rôle de chacune, afin de les décrire plus facilement. Il est important d'identifier comment chaque motivation va agir, c'est-à-dire sa tendance et surtout qu'elle en est la cause. Ce dernier élément de la description de motivation permettra de déterminer la partie de l'alternative qu'il faudra prendre en compte. Une bonne description du rôle d'une motivation facilitera la définition de l'évaluateur à l'étape 3. Cette étape ne nécessite pas de programmation, typiquement dans la conception de comportement pour un jeu vidéo elle peut être faite par le game designer, comme un cahier des charges que les programmeurs doivent obtenir à l'étape 3. Dès cette étape, certaines notions doivent être prises en compte, notamment la notion d'alternative pour pouvoir décrire la ou les parties de l'alternative qui sont concernées par la motivation.

Étape 2, choisir la fonction de combinaison

Chaque motivation donne son "avis" (évaluation) sur les actions exécutables. Le rôle de la fonction de combinaison est d'agréger ces évaluations afin d'obtenir une évaluation globale. La fonction de combinaison joue un rôle similaire à un "arbitrator" de DAMN [Ros97] qui désigne l'action à exécuter une fois que chaque module de comportement à "voter". Il existe plusieurs méthodes pour combiner les motivations, telles que la prise en compte d'un choix social [Arr51] pour représenter les "avis" des motivations. Il existe également plusieurs fonctions mathématiques utilisables[8] comme fonction de combinaison, chacune ayant des propriétés pouvant influencer la sélection d'action.

Toutefois, parmi l'ensemble des propriétés de ces fonctions mathématiques, afin de préserver l'indépendance des motivations les unes des autres ainsi que de marquer l'impact de chaque motivation sur l'évaluation finale, j'ai identifié une propriété essentielle.

Propriété 3.1 (Fonction de combinaison pour CoCoA) *Soit alt une alternative et ϕ une fonction de combinaison avec n évaluateurs, tels que $\phi(alt) = Comb(\{\gamma_i(alt)\})$ avec $i \in [1, n]$. Soit ϕ' la fonction de combinaison et γ_m un évaluateur, tels que $\phi'(alt) = Comb\{\gamma_1(alt), ..., \gamma_n(alt), \gamma_m(alt)\}$. Alors une fonction de combinaison sera acceptable, si elle répond à ces critères*[9] *:*

[8] Dans la partie 4.3.3 certaines fonctions mathématiques que j'ai testées seront détaillées et le choix que j'ai fait dans CoCoA sera expliqué.

[9] Ces critères sont applicables dans le cas où le mécanisme de sélection d'action choisit l'interaction ayant obtenu l'évaluation la plus importante.

- *elle permet aux motivations d'exprimer la neutralité, la répulsion, l'attraction et l'inhibition.*
- *elle permet l'ajout et la suppression de motivations en gardant la cohérence de l'agrégation par rapport à l'individualité désirée.*

À partir de ces deux critères, je définis quatre conditions que devra respecter la fonction de combinaison :

- *Si γ_m donne une évaluation neutre, alors $\phi'(alt) = \phi(alt)$, une évaluation neutre n'ayant pas d'impact sur la sélection d'action.*
- *Si γ_m donne une évaluation attractive, alors $\phi'(alt) \geq \phi(alt)$, l'ajout d'une attraction doit augmenter l'évaluation globale de l'alternative (sauf en cas d'inhibition).*
- *Si γ_m donne une évaluation répulsive, alors $\phi'(alt) \leq \phi(alt)$, l'ajout d'une répulsion doit diminuer l'évaluation globale de l'alternative (sauf en cas d'inhibition).*
- *Si γ_m donne une évaluation inhibitrice, alors $(\phi'(alt) = \gamma_m(alt)) \leq \phi(alt)$, l'ajout d'une telle évaluation inhibite l'ensemble de l'alternative.*

Nous pouvons remarquer que la sélection d'action se basant sur la valeur maximale de la combinaison des motivations ne permet pas aux motivations, avec la propriété de la fonction de combinaison ci-dessus, d'être en concurrence direct les unes avec les autres. En effet, mon mécanisme de sélection permet d'obtenir l'action correspondant à la candidate du compromis (voir les critères de Tyrrell dans le chapitre 1), c'est-à-dire celle qui a reçu la meilleure évaluation lorsqu'on considère l'ensemble des motivations simultanément. Ainsi, l'action sélectionnée n'est pas celle qui a reçu le plus grand nombre de meilleures évaluations (lorsqu'on considère chaque évaluation une par une et qu'on somme le nombre de fois où l'action a reçu la valeur maximale).

De plus, le second des deux critères implique qu'il est possible de construire incrémentalement l'ASM. Dans ce cas, des motivations requises a posteri peuvent être ajoutées sans remettre en cause ce qui a été fait précédemment, surtout au niveau de l'individualité de l'agent. Ceci est une propriété importante qui augmente la robustesse de l'ASM.

Enfin, du choix de la fonction de combinaison et de l'intervalle de valeurs utilisées dépend la construction des évaluateurs. C'est pourquoi cette fonction doit être définie avant la construction de ces évaluateurs.

Étape 3, la construction des évaluateurs

La difficulté principale réside probablement dans cette troisième étape où les motivations choisies doivent être traduites en une fonction d'évaluation. Toutefois, il ne faut pas oublier quelques points qui permettent éventuellement de rendre cette difficulté un peu plus relative. Premièrement, l'évaluation du comportement dépendra de divers observateurs et l'unanimité est impossible. Nous avons déjà dit (dans la partie 1.1) que la désignation d'un comportement est subjective, et donc le respect du nom initial (dans la première étape) reste incertain. Deuxièmement, il y aura plusieurs motivations en concurrence qui vont évaluer la même action exécutable (alternative), la notion de " tendance" est donc importante, l'évaluateur doit faire pencher la sélection de l'action vers sa tendance, et non pas de définir précisément l'action à choisir.

De par la grande variété de motivations possibles, il n'existe pas de squelette ou de modèle pour esquisser ce à quoi l'évaluateur devrait ressembler. Toutefois, comme il a été indiqué dans la partie 3.2.2, le concepteur doit être vigilant pendant cette phase de construction afin que les motivations prennent en compte les trois parties de l'alternative : les buts, l'action exécutable et les autres actions de la séquence. Encore une fois, chaque motivation doit prendre en compte les parties de l'alternative en fonction de sa définition et pas forcément l'ensemble de l'alternative. Ce qui compte c'est que les trois parties de l'alternative soient considérées par l'ensemble des motivations.

Lors de cette étape, le concepteur définit l'ensemble des paramètres π_{γ_i} pour chaque évaluateur γ_i. Ce sont ces paramètres qui une fois instanciés seront utilisés afin d'établir l'individualité de l'agent.

3.3.2 Exemple de conception de motivation

Afin d'illustrer les différentes étapes de conception de l'ASM, nous allons donner un exemple de conception d'une motivation en suivant les trois étapes précédemment décrites. Cette motivation est une motivation que j'ai implémentée dans CoCoA. Le but est également de présenter l'approche que le concepteur de comportement pourra développer. Je présenterai également une approche possible pour la conception du profil d'individualité.

Les trois étapes de conception

À l'étape 1, nous décidons que le comportement des agents doit être influencé par une motivation que j'ai nommée *opportunisme*.

Concrètement, dans le jeu The Sims (voir la partie 1.3), les ordres aux personnages sont donnés par l'utilisateur et organisés dans une file (FIFO). Dans ce jeu il existe un cas qui nous servira à décrire la motivation d'opportunisme. Ce cas est un exemple bien connu dans ce jeu, qui est celui du journal et du courrier. Le Sims pour prendre son courrier et son journal a ce comportement : il sortira de la maison, prendra le courrier, le déposera sur la table la plus proche à l'intérieur de la maison, ressortira, prendra le journal et le déposera à l'intérieur. Ce comportement est très coûteux en déplacement et peu rationnel. Un comportement plus réaliste aurait été de sortir, de prendre le courrier et le journal et de les déposer à l'intérieur comme l'explique le schéma de la figure 3.3.

Un opportuniste est défini comme "*une personne qui adapte ses actions, réponses, etc., pour prendre avantage d'opportunités, de circonstances, etc.*". Nous décidons que cette motivation s'exprimera en favorisant les actions dont la cible des actions est proche de l'agent. Un observateur devrait voir que lorsque l'agent se déplace dans l'environnement, il semble privilégier les éléments qui aident à la résolution de ses buts.

Par exemple, un agent doit manger pour survivre et s'il se déplace à proximité d'une pomme, l'agent devrait être enclin à prendre la pomme et à la manger. L'opportunisme privilégiant les actions dont les cibles sont proches, l'agent peut temporairement arrêter une résolution afin de profiter de la proximité d'une cible pour réaliser une action et reprendre la résolution précédente. Privilégier prendre une pomme et la manger peut être interprété comme

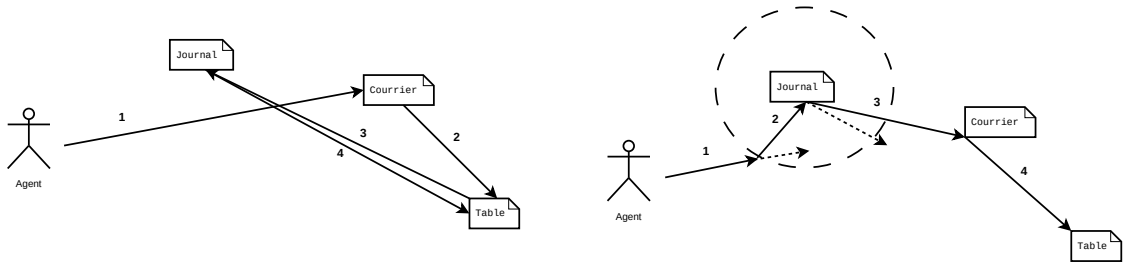


FIG. 3.3 – Schéma de l'étape 1 décrivant l'influence de l'opportunisme. Dans cet exemple, l'agent a reçu deux ordres, l'ordre de prendre et déposer le journal sur la table, l'ordre de prendre et déposer le courrier sur la table, le premier étant plus prioritaire que le second (pour reprendre l'organisation en FIFO des Sims). **À gauche,** un mécanisme de sélection ne prenant pas en compte l'opportunisme, l'agent exécute les actions dans un ordre très coûteux en déplacement (c'est ce qui se passe dans The Sims). **À droite,** L'opportunisme influence le comportement de l'agent, étant proche du journal (la notion de proche est représentée par le cercle) l'agent se détourne du courrier pour aller prendre le journal par opportunisme, pour ensuite prendre le courrier (en respectant la priorité des buts). L'agent finit donc par prendre le journal, puis le courrier et les déposer sur la table.

un comportement réactif. La notion de "proche de la cible" doit être personnelle à l'agent et donc paramétrable et dépendre également de la distance entre l'agent et la cible de l'action.

À l'étape 2, il faut définir la fonction de combinaison *Comb*. Utiliser une fonction de combinaison pour une seule motivation est un peu superflu, néanmoins comme il est toujours possible d'ajouter de nouvelles motivations, nous devons en définir une. Nous choisissons une fonction de combinaison telle que la valeur 0 puisse signifier une inhibition, une valeur entre $]0,1[$ puisse signifier une répulsion, 1 puisse signifier une neutralité et une valeur au dessus de 1 puisse signifier une attraction.

À l'étape 3, la fonction d'évaluation doit être définie. L'opportunisme peut être vu comme une attraction provenant d'éléments de la simulation. Ceci représente une tendance à court terme et très contextuelle. De ce fait seule l'action exécutable de l'alternative sera prise en compte pour cette motivation. Nous décidons que cette motivation a un effet dans une zone restreinte (le voisinage) et qu'elle n'en a pas en dehors (la tendance sera neutre). À l'intérieur, plus la cible est proche de l'agent, plus l'attraction sera importante. Ainsi nous définissons l'évaluateur de l'opportunisme γ_{opp} comme cela :

Soit $alt = (g, \alpha, (a_i)_{i\in[1,n]})$ une alternative,

$$\gamma_{opp}(alt) = \begin{cases} 1 \text{ if } \theta_{opp} \leq 1 \\ 2 \text{ if } dist(target(\alpha)) = 0 \\ \max\left(1, 1 + \log_{\theta_{opp}}\left(\frac{\theta_{opp}}{dist(target(\alpha))}\right)\right) \text{ else} \end{cases}$$

où
- *target* est une fonction qui fournit la cible de l'action α,
- *dist* est une fonction qui calcule la distance en nombre de mouvement entre l'agent, acteur de l'action, et un élément,
- θ_{opp} est la portée de l'influence de l'opportunisme.

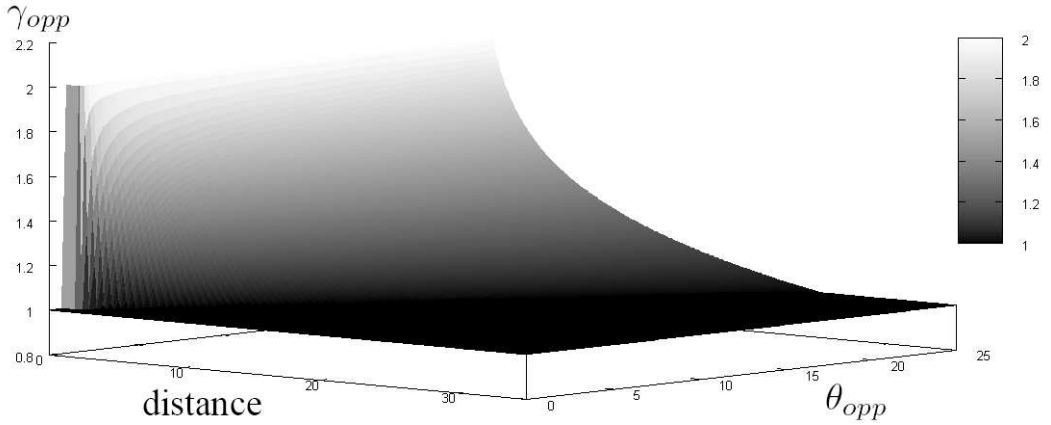


FIG. 3.4 – Les valeurs γ_{opp} selon la distance (entre l'agent et la cible) et la valeur de la portée d'influence de l'opportunisme θ_{opp}. La zone noir représente la neutralité (la cible est trop loin), ce qui signifie que l'opportunisme n'a pas d'influence sur le comportement de l'agent.

Dans cette fonction nous introduisons un logarithme afin de limiter la force d'attraction d'une cible proche, ainsi $\gamma_{opp}(\alpha) \in [1, 2]$ (voir la figure 3.4).

θ_{opp} est le seul paramètre de cet évaluateur, donc $\pi_{\gamma_{opp}} = \{\theta_{opp}\}$. Cette valeur peut être changée d'un agent à un autre pour personnaliser le profil d'individualité. Plus cette valeur est grande, plus l'agent sera opportuniste (selon notre définition d'opportunisme).

L'opportunisme est un exemple de conception de motivation. D'autres motivations seront présentées dans la partie 4.3.2, parmi ces autres motivations certaines exprimeront la répulsion ou l'inhibition et ensemble elles utiliseront toutes les parties de l'alternative pour leurs évaluations.

Conception du profil d'individualité : prototypes

Une fois défini l'ASM qui sera partagé par tous les agents, la seconde tâche du concepteur est de définir les profils d'individualité des agents.

Étant donné le nombre d'évaluateurs différents qu'il est possible de créer et les caractéristiques différentes de chaque évaluateur et leurs paramètres, il n'est pas possible de proposer une méthode universelle pour la conception du profil d'individualité. Cependant, en prévention de la lourdeur que peut représenter cette tâche répétitive, le concepteur peut définir des instances de profils d'individualité, en regroupant des ensembles de valeurs pour le même sous-ensemble de paramètres d'évaluateurs. Ces instances représentent un ensemble de valeurs possédant une sémantique exprimant un ou plusieurs traits de personnalité. Ainsi, le concepteur peut paramétrer les individualités d'agent en combinant des prototypes différents. De telles instances peuvent être considérées comme des **prototypes de profil d'individualité**. Le but est d'éviter de définir de façon répétitive les paramètres un par un pour chaque agent.

Le concepteur n'aurait qu'à choisir plusieurs prototypes afin de construire le profil d'individualité de l'agent. Pour cela, il est nécessaire que le concepteur caractérise les intervalles de valeurs des paramètres (ces intervalles ne devant pas nécessairement couvrir toute la gamme

des valeurs possibles). Pour un personnage donné, la valeur assignée peut alors être choisie dans cet intervalle, la diversité des profils obtenus est alors accrue.

Par exemple, prenons l'évaluateur de l'opportunisme défini précédemment, nous choisissons arbitrairement de définir les instances de prototypes pour $\{\theta_{opp}\}$:

- $\theta_{opp} = 1$ (ou $\theta_{opp} \in [1, 1]$) correspond à *pas opportuniste*
- $\theta_{opp} \in [2, 3]$ correspond à *faiblement opportuniste*
- $\theta_{opp} \in [5, 8]$ correspond à *moyennement opportuniste*
- $\theta_{opp} \in [12, 20]$ correspond à *très opportuniste*

Ainsi, selon l'importance de la motivation pour l'agent que le concepteur veut exprimer, ce dernier peut choisir entre *pas opportuniste*, *faiblement opportuniste*, *moyennement opportuniste* ou *très opportuniste*.

Les limites de l'intervalle de valeurs peuvent également être fixées à une valeur de propriétés de l'agent afin d'être spécifique à ce dernier. Dans le cas de $\{\theta_{opp}\}$, cette valeur peut correspondre à une portion de la valeur du champ de perception (la portée de vision par exemple) de l'agent, sans jamais être supérieure pour rester réaliste. Ainsi, *faiblement opportuniste* pourrait correspondre à un θ_{opp} plus faible que le quart de la portée, *moyennement opportuniste* du quart à la moitié et *très opportuniste* de la moitié jusqu'à la portée de perception. Ceci peut rendre la conception des profils d'individualité plus accessible, plus compréhensible et moins répétitive.

Il est alors possible de considérer deux niveaux de conception suivant les connaissances du modèle nécessaire. Le premier correspond au travail d'une sorte d'administrateur. Ce dernier construit le mécanisme de sélection en suivant les trois étapes de conception, en définissant les motivations, la fonction de combinaison et les évaluateurs. Puis pour chaque évaluateur, le concepteur construit les prototypes de profils d'individualité. Il définit donc des intervalles de valeurs et les nomme suivant les intervalles choisis afin de les différencier et permettre à un autre utilisateur de choisir plus facilement les prototypes pour le profil qu'il désire concevoir. Ce rôle nécessite une bonne connaissance de notre modèle et de la programmation pour construire le mécanisme de sélection d'action dans sa totalité. Il faut aussi essayer de définir les prototypes de manière à faciliter leur utilisation. Le second niveau correspond au travail d'un concepteur d'individualités qui n'aurait pas connaissance des fonctions d'évaluations, de leur intervalle de valeurs ni de la fonction combinaison utilisée. Ce concepteur n'aurait qu'à piocher parmi les prototypes définis pour construire un profil d'individualité de chaque personnage qu'il désire concevoir. Basée sur la sémantique des noms des prototypes, cette tâche ne nécessite pas de connaissance en programmation ou sur le mécanisme de sélection d'action qui a été construit au niveau précédent. Cette tâche peut être effectuée, par exemple, par un game designer (souvent les game designers ne rentrent pas dans le code du jeu).

Conclusion

Dans ce chapitre, j'ai présenté ma proposition pour la conception de comportement. Pour cela, j'ai défini une séparation entre la partie raisonnement et individualité du comportement dont le rôle et le type d'influence sur le comportement sont différents. Puis, j'ai proposé un mécanisme de sélection d'action basé sur les motivations et les alternatives, pour gérer la par-

tie individualité du comportement. Comme les motivations sont définies indépendamment de l'environnement dans lequel l'agent sera utilisé, elles sont réutilisables pour d'autres agents et d'autres simulations (d'autres environnements). Ce mécanisme sélectionne la meilleure action candidate du compromis, c'est-à-dire la meilleure action lorsqu'on considère l'ensemble des motivations simultanément. Enfin, j'ai défini une méthodologie de conception pour ce mécanisme de sélection d'action et j'ai illustré ma proposition par une motivation qui a été implémentée dans CoCoA.

Dans la présentation de ce chapitre, j'ai supposé travailler avec des agents cognitifs situés dont le comportement était dirigé par des buts. Mais, du point de vue du comportement, la différence entre agents réactifs et agents cognitifs n'est pas très importante, car au final tout dépend de la connaissance qui est manipulée. S'il est possible de retrouver la notion d'alternative dans un système dit réactif, alors il est possible d'utiliser la partie individualité du comportement. Par exemple, dans un système réactif à base de règles, pour une simulation donnée, les alternatives peuvent être construites en retrouvant l'enchainement des règles qui sera à réaliser. La perception, la mémorisation et la planification ne permettent que d'obtenir des comportements plus rationnels. Ma proposition ne dépend pas de l'architecture de nos agents, ni de l'environnement dans lequel la simulation se déroule. Le comportement n'est pas non plus contraint par l'architecture, les motivations sont indépendantes les unes des autres et aucun poids n'est attribué aux motivations. Ainsi, l'expression de chaque motivation influence le comportement quelque soit leurs natures où le type de comportement observable qu'elles peuvent engendrer. Ceci est vrai, que le comportement soit jugé comme réactif ou cognitif, qu'il soit issu d'une planification à moyen terme ou d'une action immédiate (par exemple dans [And03] les actions planifiées sont moins prioritaires).

J'ai également présenté mon modèle en dehors de CoCoA. Pourtant ce projet vient compléter mes travaux et notamment la réutilisation de la partie raisonnement du comportement. En effet, l'approche centrée interaction et le projet CoCoA permettent déjà de définir la partie raisonnement de manière déclarative à l'aide des interactions. Dans CoCoA, le processus de planification définit comment l'agent peut résoudre ses buts selon les connaissances que l'agent possède dans sa mémoire et ses capacités (les interactions peut-effectuer). Cette séparation déclarative des interactions et du processus procédural de planification, permet l'utilisation du même moteur pour des agents ayant des capacités différentes. Les interactions, comme les motivations, sont définies indépendamment du contexte dans lequel elles seront utilisées. Les interactions, comme les motivations, sont réutilisables pour d'autres agents et d'autres simulations. Par l'utilisation conjointe de l'approche centrée interaction et du mécanisme de sélection d'action basé sur les motivations, il est possible de définir les deux parties du comportement de manière "universelle" indépendamment du contexte, permettant la réutilisation du comportement (tout ou partie) pour d'autres agents et d'autres simulations.

Dans le chapitre suivant nous verrons comment j'ai mis en place ma proposition sur la conception de comportement pour la plateforme de simulation CoCoA. La partie raisonnement assurée par l'approche centrée interaction et la planification nécessitait quelques adaptations pour être conforme au modèle, notamment sur la construction de l'alternative à partir des arbres de planification. La partie individualité se résumait à une heuristique de sélection d'action, la construction d'un mécanisme de sélection d'action basée sur les motivations a donc été réalisée.

Chapitre 4

Contributions à CoCoA

« Pour la tâche que nous aimons, nous nous levons de bonne heure, Et nous y mettons avec joie. »
William Shakespeare, Antoine et Cléopâtre

Comme présenté précédemment, le projet CoCoA possédait un mécanisme de sélection d'action rudimentaire qui ne permettait pas la construction de différents comportements. J'ai donc mis en place la proposition du chapitre précédent dans CoCoA.

CoCoA présente déjà une séparation entre la partie raisonnement et la partie individualité du comportement qui sont gérés par deux modules différents. Néanmoins, le raisonnement ne manipule pas les alternatives et les plans sont construits à l'aide d'un $Arbre - et/ou$. Le mécanisme de sélection reçoit du plan les interactions exécutables et choisit l'interaction dont la cible est la plus proche de l'agent. Enfin les propriétés des agents sont des valeurs statiques modifiables uniquement par l'intervention d'une interaction.

Afin de mettre en place ma proposition et de permettre de définir des comportements variés et plus complexes, j'ai dû mettre en place trois éléments : les propriétés dynamiques, les alternatives et les motivations.

La notion de propriétés qui évoluent pendant la simulation, sans intervention d'interaction, n'est pas présente et est pourtant importante pour simuler certains comportements. En effet, si l'on veut pouvoir concevoir des agents dont les propriétés homéostatiques influencent leur comportement (comme la soif ou la faim) il est utile de mettre en place cette notion. De plus, sachant que les interactions et les motivations se basent sur les propriétés des agents, cette première étape permet d'élargir les possibilités dans la création de comportement dans CoCoA.

Je me suis également focalisé sur la partie raisonnement du comportement. Cette partie correspond au module de planification dans CoCoA. L'approche centrée interaction utilisée permet déjà de définir des interactions génériques et de les réutiliser. Toutefois, pour correspondre à ma proposition, le planificateur doit fournir des alternatives, ce qui n'était pas le cas.

Nous verrons donc comment il est possible de construire des alternatives à partir des plans fournis par le planificateur de CoCoA.

Une fois les alternatives construites à partir de l'arbre de planification, je présenterai le mécanisme de sélection d'action basé sur les motivations mis en place pour gérer la partie individualité du comportement des agents CoCoA. Pour la construction de ce mécanisme, je me suis fixé comme contrainte de définir des motivations correspondantes aux critères de Tyrrell d'un bon mécanisme de sélection d'action (à l'étape 1 de la conception). De plus, pour la création de la fonction de combinaison et des évaluateurs, l'objectif a été de concevoir ces éléments afin qu'ils répondent aux critères de l'étape 1 et qu'ils ne soient pas trop compliqués à comprendre. Ceci afin de montrer qu'il est possible d'obtenir des comportements variés avec un mécanisme simple.

4.1 Des propriétés qui évoluent

Comme nous l'avons vu précédemment, dans CoCoA tous les agents possèdent des propriétés. Certaines propriétés sont communes à tous les agents de la même classe, d'autres sont uniques ou spécifiques à l'agent. Une propriété est un couple symbole-valeur. Le symbole représente le nom de la propriété (son identifiant), la valeur correspond à la valeur de la propriété. La valeur d'une propriété est fixée au début de la simulation et elle ne peut être modifiée au cours de la simulation que par l'exécution d'une interaction. En effet, la partie action d'une interaction permet de modifier les propriétés de l'agent cible et de l'agent source de l'interaction. Néanmoins, lorsque l'on souhaite représenter des propriétés homéostatiques de l'agent comme la soif ou la faim, l'agent n'exécutant qu'une action à la fois, il ne semble pas possible de le faire par l'intermédiaire d'une interaction et le sens de l'utilisation d'une interaction resterait encore à justifier.

Les propriétés homéostatiques ont un impact sur le comportement de l'agent et donc sur les actions qu'il va exécuter. Ainsi, lorsqu'un agent a soif (i.e. sa propriété soif est au dessus d'un seuil limite), il doit exécuter l'action boire et lorsque il a faim, il doit exécuter l'action manger. C'est pourquoi il est indispensable de mettre en place des propriétés qui évoluent pendant une simulation sans l'intervention d'interaction. Ainsi, par l'utilisation de propriétés dont la valeur évolue dynamiquement au cours de la simulation (sans que cette évolution ne soit due à l'exécution d'une interaction), nous pouvons introduire de nouveaux facteurs influençant le comportement.

4.1.1 Propriétés constantes et dynamiques

Afin de mettre en place des propriétés dont la valeur évolue au cours de la simulation, je distingue deux types de propriétés : les propriétés constantes et les propriétés dynamiques.

Une propriété constante est une propriété dont la valeur n'est modifiable que via l'utilisation d'une interaction, par exemple l'état d'un agent `porte` qui change lorsqu'il subit l'interaction `ouvrir`.

Une propriété dynamique est une propriété dont la valeur peut changer au cours de la simulation sans que cela soit l'effet d'une interaction (valeur d'une propriété dynamique

peut quand même être modifiée par l'exécution d'une interaction). Parmi les propriétés dynamiques nous distinguons deux cas suivant le mode de calcul de la valeur de la propriété : les propriétés dynamiques évolutives (ou propriétés évolutives) et les propriétés dynamique dépendantes (ou propriétés dépendantes).

Les propriétés évolutives sont les propriétés dont l'évolution de la valeur ne dépend pas d'une autre propriété. À chaque étape de l'évolution, la valeur des propriétés évolutives est recalculée suivant leur règle d'évolution. La valeur d'une propriété évolutive à une étape $t+1$ est déterminée par rapport à sa valeur à l'étape t. Par exemple, supposons que nous utilisons une propriété évolutive pour représenter l'`énergie` de l'agent. La valeur de l'`énergie` décroit à chaque pas de la simulation. À partir de la valeur de l'énergie à l'étape t, il est possible de déterminer la valeur à l'étape suivante $t+1$. Ceci implique d'avoir l'unicité des étapes d'évolution. Les règles d'évolution doivent par conséquent être injectives par rapport à l'étape d'évolution.

Les propriétés dépendantes sont les propriétés dont la valeur dépend de la valeur d'une autre propriété. L'évolution de la propriété n'est donc prévisible qu'en connaissant l'évolution de la propriété dont elle est dépendante. Par exemple, si un agent a la propriété `fatigue` qui représente le niveau de fatigue de l'agent, nous pouvons définir que plus un agent a d'énergie et moins il est fatigué. Ainsi, le niveau de fatigue (la valeur de la propriété fatigue) d'un agent est inversement proportionnelle à son niveau d'énergie (à la valeur de sa propriété énergie). La valeur d'une propriété dépendante peut dépendre de la valeur d'une propriété constante, évolutive ou dépendante.

4.1.2 Implémentation des propriétés

Afin d'unifier les différentes formes de propriétés tout en gardant leur différences, toutes les propriétés sont définies par un couple symbole-fonction. Le symbole représente le nom de la propriété (son identifiant) et la fonction sert à déterminer à chaque instant la valeur de la propriété.

Les propriétés constantes possèdent une fonction constante, les propriétés dynamiques possèdent une fonction définissant la règle d'évolution de la valeur de la propriété. Pour représenter ces propriétés, nous avons défini des fonctions mathématiques décrivant des règles d'évolution, comme les constantes, les fonctions linéaires, les fonctions linéaires bornées (max ou min-et-max), les sigmoïdes, etc, auxquelles nous avons rajouté une classe représentant les fonctions évolutives et une autre pour les fonctions dépendantes.

Par exemple, dans le cadre de valeurs numériques nous pouvons avoir :

- Le `champ de perception`, qui est une propriété numérique dont la fonction est constante représentant le nombre de cases (le rayon) au maximum que l'agent peut percevoir.
- L'`énergie` de l'agent est une propriété numérique dont la fonction est linéaire et dont le paramètre t représente une étape dans l'évolution de la valeur et $t+1$ représente l'étape suivante.
- La `fatigue` de l'agent est une propriété numérique dont la fonction est linéaire et dont le paramètre λ représente la valeur de la propriété `énergie` de l'agent.

Pour nos trois propriétés (voir la figure 4.1), cela se traduit concrètement par :

```
Function cst = new ConstantFunction(4); // cst(x) = 4
Property perception = new Property("perception", cst);

Function f = new LinearFunction(-1, 50); // f(x) = -x + 50
// évolution suivant la fonction f avec 30 comme valeur de départ.
Function evo = new EvolutiveFunction(30, f);
Property energie = new Property("energie", evo);

Function g = new LowBoundedLinearFunction(-1, 30, 0); // g(x) = min (0, -x + 30)
Function dep = new DependantFunction(energie, g); // dep(x) = g(energie) = g(f(x))
Property fatigue = new Property("fatigue", dep);
```

Une fonction linéaire `LinearFunction` est construite à partir des valeurs `a` et `b` de son équation `y=ax+b`. Une fonction linéaire inférieure bornée `LowBoundedLinearFunction` est construite à partir de la borne minimale `bm` et des valeurs `a` et `b` de son équation `y= max(bm,ax+b)`. Une fonction évolutive `EvolutiveFunction` est construite à partir d'une valeur de départ et d'une fonction mathématique fixant sa règle d'évolution. Une fonction dépendante `DependantFunction` est construite à partir d'une propriété dont elle dépend et d'une fonction mathématique fixant le calcul à effectuer en prenant en compte cette propriété.

Pour déterminer la valeur d'une propriété dépendante, il faut calculer l'image de la valeur de la propriété dont elle dépend par la fonction mathématique fixant le calcul à effectuer.

Pour déterminer la valeur d'une propriété évolutive, il faut dans un premier temps déterminer l'étape d'évolution e de la valeur actuelle de la propriété. En effet, la valeur de l'énergie peut être modifiée par une interaction subie ou effectuée, modifiant ainsi l'étape d'évolution de la propriété. Ainsi, la valeur d'une propriété évolutive pour l'étape $t+1$ de la simulation, se calcule à partir de la valeur actuelle de la propriété à l'instant t. Pour cela, à partir de la valeur actuelle de la propriété, il faut calculer son antécédent x_e (la valeur x à l'étape d'évolution e de la propriété) par rapport à la fonction d'évolution[10]. Cet antécédent est unique (la fonction d'évolution est injective) et il représente l'étape d'évolution de la valeur de la propriété. Puis, il faut passer à l'étape suivante x_{e+1} pour calculer la nouvelle valeur de la propriété. Cette nouvelle valeur correspond à l'image de l'étape suivante $f(x_{e+1})$ par la fonction évolutive f de la propriété.

Prenons l'exemple de l'énergie qui est une propriété dynamique évolutive. La valeur de l'énergie évolue à chaque pas de temps et suivant certaines interactions subies ou effectuées par l'agent. Lorsqu'un agent effectue l'interaction `manger`, la valeur de sa propriété énergie est immédiatement modifiée :

```
manger
condition : actor.own(target)
garde : -
action : actor.energie = actor.energie + target.energie
         destroy(target)
```

Prenons le cas précis de la figure 4.1 où l'axe des abscisses représentent les étapes de la simulation. Dans cette figure nous pouvons voir qu'à l'étape $t = 0$ de la simulation, la

[10] J'utilise t pour parler des étapes de la simulation et e pour les étapes d'évolution d'une propriété

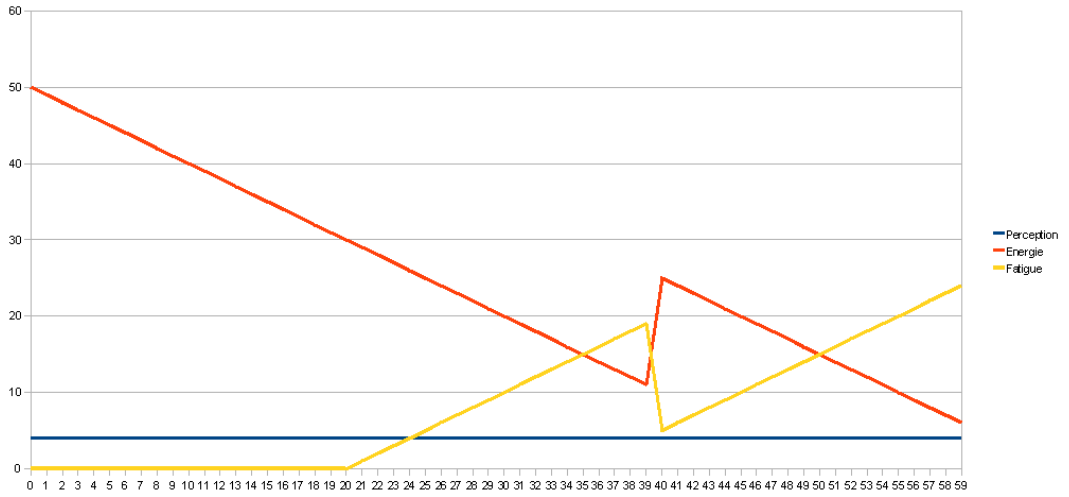


FIG. 4.1 – Évolution des valeurs des propriétés énergie, fatigue et perception pendant la simulation. À l'instant 40, l'agent effectue l'interaction *manger* sur une pomme, ce qui a pour effet d'augmenter l'énergie de l'agent de 15. La valeur de la propriété `fatigue`, qui est dépendante de l'`énergie`, a été changée en conséquence.

valeur de l'énergie vaut 50. Jusqu'à l'étape $t = 40$ de simulation, nous pouvons confondre l'étape de simulation et l'étape d'évolution de la propriété énergie, puisque aucune interaction n'a modifié la valeur de la propriété énergie($e = t$). En effet, si la valeur de l'énergie est définie par la fonction $evo(x) = -x + 50$, dans ce cas $evo(0) = 50$. À l'étape suivante de la simulation, la valeur de l'énergie est calculée par rapport à l'évolution de son antécédent ($x + 1 = 0 + 1 = 1$ et $evo(1) = -1 + 50 = 49$). Pour l'étape $t = 39$ de la simulation, l'énergie vaut 11 ($evo(39) = -39 + 50 = 11$). À l'étape $t = 40$ l'agent effectue l'interaction *manger*, qui augmente son énergie de 15, la portant à 25. Pour déterminer la valeur à l'étape $t = 41$ de la simulation, il faut calculer la valeur de l'antécédent à l'étape de la simulation précédente. À l'étape $t = 40$ de la simulation, la propriété énergie est à l'étape $e = 25$ d'évolution ($evo(x) = 25 \rightarrow x = 25$). Donc, à l'étape l'étape $t = 41$ de la simulation, la propriété énergie est à l'étape $e = 26$ et sa valeur vaut 24 ($evo(26) = -26 + 50 = 24$).

4.1.3 Mise à jour des propriétés

Les propriétés évoluant à chaque tour de la simulation, il est important d'intégrer dans le cycle d'exécution de la simulation la mise à jour des propriétés. Cette mise à jour concerne tous les agents (actifs ou non) possédant des propriétés dynamiques. Dans CoCoA, à chaque pas de temps, les agents n'effectuent qu'une seule action. Le moteur de simulation signale à chaque agent actif qu'un nouveau pas de temps est démarré et qu'il peut donc agir à nouveau. Ceci déclenche chez l'agent le calcul des actions exécutables et l'exécution d'une action. Une fois qu'un pas de temps de la simulation est terminé, j'ai ajouté un envoi d'appel à tous les agents ayant des propriétés dynamiques (via un système d'abonnement) afin qu'ils mettent à jour leurs propriétés. Les agents peuvent subir ou effectuer des interactions pouvant modifier leurs propriétés, les plans construits dépendent de propriétés qui peuvent appartenir à d'autres agents et être dynamiques. Afin de préserver une cohérence entre l'état du monde et les plans

des agents, la mise à jour des propriétés dynamiques n'est effectuée qu'une fois que tous les agents actifs ont agi.

4.1.4 Modifier le profil comportemental des agents avec des propriétés

La valeur d'une propriété peut évoluer en fonction des interactions, des pas de temps de la simulation ou d'une autre propriété de l'agent, mais elle peut également évoluer en fonction d'autres agents (et de leurs propriétés).

Les évaluateurs des motivations sont paramétrés par le profil comportemental. Ces paramètres sont des valeurs numériques permettant d'exprimer la manière dont les motivations vont affecter le comportement. Nous avons déjà vu qu'afin de simplifier le paramétrage, ce profil comportemental pouvait être défini en fonction de la valeur d'une propriété. Si ces propriétés évoluent pendant la simulation, les valeurs du profil comportemental évolueront également ainsi que l'influence des motivations et donc le comportement de l'agent. Il est donc possible d'obtenir un comportement qui évolue au cours de la simulation à l'aide de propriétés dynamiques. De même, les interactions effectuées ou subies peuvent changer la valeur des propriétés d'un agent, changeant ainsi le profil comportemental et donc le comportement de l'agent. Nous pouvons également imaginer qu'un agent qui subit (respectivement effectue) une interaction peut changer de comportement ou changer le comportement de l'agent qui effectue (respectivement subit) cette interaction.

Par exemple, si l'on souhaite créer des objets qui peuvent changer le profil comportemental d'un agent, il suffit de définir une interaction à effectuer pour prendre ces objets. Cette interaction possède, dans la partie `action` de sa définition, la modification de propriétés de l'agent et donc de son profil d'individualité afin que son comportement change.

Nous voyons donc que la mise en place de ces propriétés nous offre des possibilités supplémentaire de modélisation et d'évolution du comportement de l'agent.

4.2 La construction des alternatives

Nous avons vu que la partie raisonnement du comportement était principalement gérée par un moteur de planification. Dans CoCoA, ce moteur manipule un $Arbre-et/ou$ qui présente en racine les buts à résoudre, en feuilles les interactions exécutables et dont les branches alternent les nœuds-et représentant les conditions (ou les états à atteindre) et les nœuds-ou représentant des interactions. Or, dans notre proposition le raisonnement doit fournir des alternatives au mécanisme de sélection d'action. J'expliquerai donc dans cette partie comment il est possible de retrouver les alternatives dans un $Arbre - et/ou$.

Avant d'aller plus loin dans les explications définissons quelques éléments de l'arbre de planification. Pour définir un plan, nous nous appuyons sur deux éléments : les états et les interactions.

Un état E_i est une description d'une partie de l'environnement dans lequel évolue l'agent. L'environnement peut ainsi être entièrement décrit sous la forme d'une conjonction d'états.

Une interaction A_i est une interaction que peut effectuer l'agent dans l'environnement suivant certaines conditions, cette interaction peut provoquer des changements dans l'environnement. Nous pouvons donc associer à chaque A_i un ensemble d'états nécessaires à son exécution et un ensemble d'états résultants de son exécution. A_i : {états requis} {états résultants}.

Une situation est un ensemble d'états satisfaits.

Un but est une situation à satisfaire, c'est-à-dire un ensemble d'états qui doivent être satisfaits.

4.2.1 La notion d'alternative dans un $Arbre - et/ou$

Avant de présenter l'algorithme que nous avons utilisé pour construire les alternatives à partir d'un $Arbre - et/ou$, prenons un exemple $Arbre - et/ou$ afin de mieux comprendre ce que représente les alternatives.

Soient $Ea, ..., Ez$ l'ensemble des états que peut prendre l'environnement.

La situation initiale : $\{Ea, Er\}$.

Le but : $\{Ez, Ew, Ey\}$

$\{A1, ..., A10\}$ l'ensemble des interactions que l'agent peut effectuer.

$A1 : \{Eb, Ec\}\{Ez, Ex\}$.

$A2 : \{Er\}\{Ez, Ep\}$.

$A3 : \{Ee, Ef\}\{Ez\}$.

$A4 : \{Ea, Er\}\{Ew, Es, Et\}$.

$A5 : \{Eh\}\{Ey\}$.

$A6 : \{Ea\}\{Eb\}$.

$A7 : \{Ea\}\{Eh, Em\}$.

$A8 : \{Ea, Er\}\{Ee\}$.

$A9 : \{Ea\}\{Ef\}$.

$A10 : \{Er\}\{Ef\}$.

À partir de cet exemple, nous pouvons construire par chaînage arrière l'$Arbre - et/ou$ de planification présent en figure 4.2. Les interactions $A2, A4, A6, A7, A8, A9, A10$ sont dites exécutables au début de la simulation car leurs conditions sont satisfaites par la situation courante (qui est la situation initiale au début de la simulation). Ces actions seront présentes en feuille de l'arbre de planification.

Dans cet arbre, nous pouvons extraire toutes les alternatives possibles afin d'obtenir un $Arbre - ou$ (figure 4.3) où chaque branche représente une alternative. Dans les $Arbres - ou$ de planification, la racine est le but à atteindre, les autres nœuds sont des interactions et les feuilles sont les interactions exécutables. On voit rapidement que le passage d'un $Arbre - et/ou$ à un $Arbre - ou$ implique un nombre important de branches, chaque branche correspondant à une alternative. Ces alternatives sont créées à partir de chaque $Noeud - ou$ de l'$Arbre - et/ou$ d'origine et à partir de chaque permutation dans l'ordre d'exécution des $Noeuds - et$ (A et B est équivalent à l'interaction A suivie de l'interaction B ou l'interaction B suivie de l'interaction A).

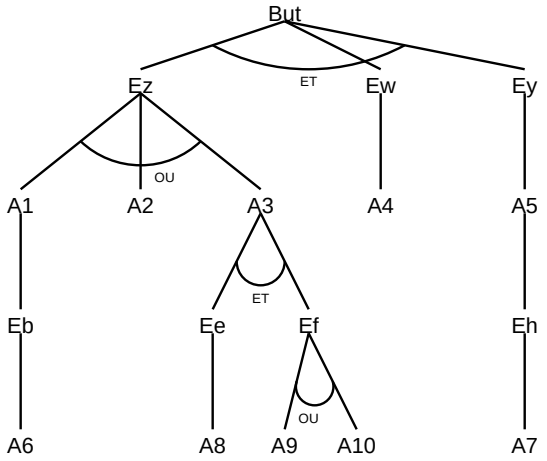


FIG. 4.2 – Exemple d'un $Arbre - et/ou$, la racine But est le but que doit réaliser l'agent, les feuilles (des nœuds interactions) sont {A2, A4, A6, A7, A8, A9, A10}. Chaque état Ei est placé dans un $Noeud - ou$, car il y a plusieurs manières d'atteindre un état. Chaque interaction Ai est placée sur un $Noeud - et$ car il faut remplir toutes les conditions (les états) pour pouvoir effectuer une action.

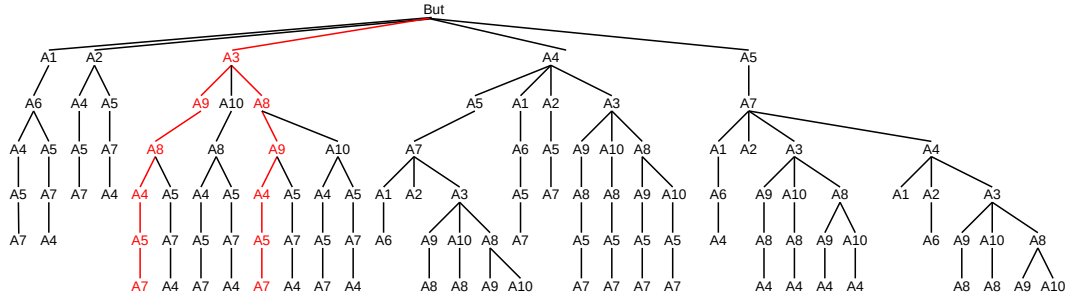


FIG. 4.3 – $Arbre - ou$ équivalent à l'$Arbre - et/ou$ en figure 4.2. Dans cette arbre, nous ne représentons que les actions pour simplifier la visualisation. L'$Arbre - ou$ représente la liste des alternatives possibles pour résoudre un but (à une branche correspond une alternative).

Dans l'$Arbre - ou$ il existe plusieurs alternatives pour une même interaction exécutable (i.e, il existe plusieurs branches pour une feuille identique). Pour un but, deux alternatives correspondant à des permutations d'actions réalisent les mêmes interactions mais dans des ordres différents. Dans le mécanisme de sélection d'action que j'ai choisi de mettre en place dans CoCoA, l'ordre d'exécution des interactions n'a pas beaucoup d'importance car ces interactions correspondent à une prévision du planificateur des interactions à effectuer qui se base sur les connaissances (potentiellement inexactes) de l'agent. Nous pouvons donc simplifier l'$Arbre - ou$ en fusionnant ces deux alternatives équivalentes. Seuls le But et l'interaction en feuille gardent leur place (le But est réalisé en dernier, l'action feuille est exécutée en première). Dans notre exemple, la séquence A7-A5-A4-A8-A9-A3-But et la séquence A7-A5-A4-A9-A8-A3-But (en rouge dans la figure 4.3), correspondent à la séquence A7-{A3, A4, A5, A8, A9}-But (en rouge dans la figure 4.4).

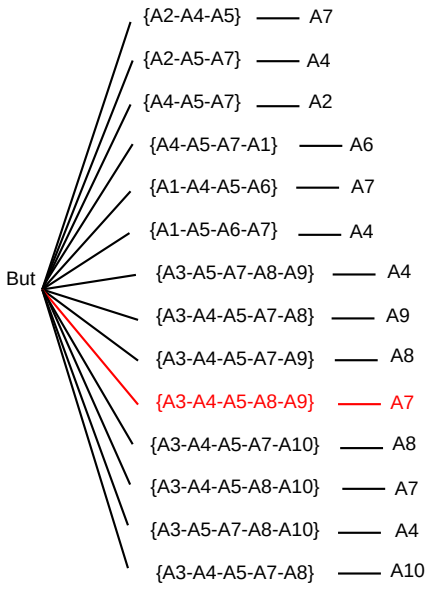


FIG. 4.4 – *Arbre – ou* ne prenant pas en compte toutes les permutations possibles, puisque ce qui nous intéresse ce sont les actions présentes dans l'arbre pas l'ordre d'exécution des *Noeuds – et*.

Si nous reprenons la définition d'un *Arbre – et/ou* et d'une alternative, nous pouvons en déduire la définition d'une alternative dans un *Arbre – et/ou*.

Définition 4.1 (Alternative dans un *Arbre – et/ou***)** *Une alternative est une séquence d'actions permettant de résoudre un but à partir d'une action exécutable. Dans un Arbre – et/ou, une alternative est composée de toutes les actions issues d'un nœud-et et d'un choix parmi les actions issues d'un nœud-ou.*

Nous discuterons de la notion de choix dans la partie sur l'élagage de l'*Arbre – et/ou* (voir la partie 4.2.3).

4.2.2 La construction des alternatives

La démarche qui semblerait la plus naturelle pour construire les alternatives serait de partir d'un but puis de descendre dans l'arbre jusqu'à l'interaction exécutable. Mais, dans CoCoA le planificateur fournit les interactions exécutables. Si nous ne voulons pas modifier le processus de planification et donc la construction de l'arbre, nous devons calculer à partir des interactions exécutables, les alternatives qui y sont associées. Ce n'est donc pas une construction par la racine (le but) que nous avons effectuée mais bien à partir des feuilles.

Afin de ne pas modifier le processus de planification, nous avons donc conçu un algorithme pour collecter les alternatives à partir des interactions exécutables. Ainsi cet algorithme se déroule en deux temps[11] (voir la figure 4.5). Le premier temps, *la phase montante* : l'algo-

[11] L'arbre de planification dans CoCoA est un *Arbre – et/ou* avec des nœuds répétés. Le principe de cet arbre est d'optimiser l'arbre *Arbre – et/ou* en ne présentant qu'une seule fois chaque occurrence d'un nœud. Toutes

rithme va collecter les interactions en remontant dans l'arbre jusqu'à atteindre le but. Puis dans un second temps, *la phase descendante* : l'algorithme va redescendre dans les nœuds fils (en évitant le nœud par lequel il est monté) pour collecter les interactions manquantes. Nous verrons dans la phase descendante que la collecte parmi les fils d'un $Noeud - et$ n'est pas ordonnée. Néanmoins, dans le code que nous avons mis en place dans CoCoA pour la collecte dans les $Noeud - et$, nous distinguons les interactions issues d'une garde de distance des autres interactions afin qu'elles soient collectées en dernières. En effet, comme nous l'avons précédemment souligné, les interactions issues des gardes de distances (les déplacements) doivent être effectuées en dernier pour effectuer une interaction.

Pour construire les alternatives correspondant à l'$Arbre - et/ou$, nous partons donc de l'interaction exécutable (en feuille), pour sélectionner les nœuds correspondant à son alternative. Dans un premier temps, nous prenons l'ensemble des interactions de la branche allant de la feuille jusqu'à la racine (phase montante). Ces interactions, ne nécessitant pas de choix de la part de l'agent, font partie de l'alternative. Dans la phase descendante, l'algorithme va devoir faire des choix exprimés par les $Noeuds - ou$ de l'alternative. Afin de simplifier les calculs l'algorithme de construction d'alternative effectue une présélection ou élagage à ce niveau pour effectuer un choix. En effet, les fils de ces nœuds peuvent représenter un choix que l'agent aura à effectuer. Je présenterai comment prendre en compte ces nœuds dans la partie suivante.

4.2.3 L'élagage de l'$Arbre - et/ou$

Comme nous l'avons présenté précédemment, nous ne prenons pas en compte l'ordre d'exécution des fils issus d'un $Noeud - et$(sauf pour le cas spécifique de la garde de distance). Ainsi notre collecte ressemble à l'$Arbre - ou$ optimisé (voir la figure 4.4).

De plus, la collecte d'une alternative peut fournir plusieurs alternatives. En effet, lorsqu'on cherche à construire une alternative, à chaque $Noeud - ou$ correspond un choix que l'agent doit faire et donc une nouvelle alternative. Afin de simplifier la collecte des alternatives et d'optimiser le temps de calcul nécessaire à la sélection d'action, j'ai mis en place un algorithme permettant de faire un choix dans le parcours de l'arbre en l'élaguant. Les branches issues d'un $Noeud - ou$ forment un sous-arbre dont la racine est le $Noeud - ou$. Ce sous-arbre contient des portions d'alternatives que je nommerai sous-alternative. Faire un choix dans un $Noeud - ou$ revient à sélectionner la meilleure sous-alternative du sous-arbre dont la racine est le $Noeud - ou$. Pour cela, chaque sous-alternative est évaluée par le mécanisme de sélection d'action.

Sachant que l'élagage est effectué pour construire les alternatives nécessaires au mécanisme de sélection d'action et que cette sélection choisit l'interaction (ou l'alternative correspondante) ayant obtenu la meilleure évaluation, j'ai décidé d'utiliser le mécanisme de sélection d'action pour l'élagage de l'arbre en appliquant cette heuristique : toutes les portions d'alternatives (ou sous-alternatives) issues du sous-arbre issu d'un $Noeud - ou$ (d'un choix) sont évaluées

les autres occurrences sont présentées comme des répétitions compactant les nœuds identiques (et donc des sous-arbres issus de ces nœuds). Pour cela, les nœuds présents plusieurs fois font référence à la seule version du nœud entièrement développé dans l'arbre, les autres sont des répétitions. L'algorithme que j'ai mis en place dans CoCoA fonctionne pour un $Arbre - et/ou$ avec des nœuds répétés, toutefois j'en présente une version simplifiée pour les $Arbres - et/ou$.

```
Algorithme de collecte d'une alternative dans un arbre-et/ou

// Donne le père du noeud n
Début Pere (n: Noeud) -> Noeud
  retourner le noeud pere du noeud n.
Fin Pere

// Donne les fils du noeud n
Début Fils (n: Noeud) -> Ensemble <Noeud>
  retourner les noeuds fils du noeud n.
Fin Pere

// Récupère les noeuds interactions de l'alternative de l'interaction x.
Début N (x: NoeudInteraction) -> Ensemble <NoeudInteraction>
  alt : Ensemble <NoeudInteraction> = vide
  Si x n'est pas la racine Alors
    alt.ajouter(x)
    alt.ajouterEnsemble(P (Pere(x))
  FinSi
  retourner alt
Fin N

// Collecte les noeuds interactions parmi les ancêtres du noeud courant.
Début P (x: NoeudEtat) -> Ensemble <NoeudInteraction>
  alt : Ensemble <NoeudInteraction> = vide
  alt.ajouterEnsemble(N (Pere(x))
  Pour chaque Noeud s de Fils(Pere(x)) faire
    Si s est différent de x Alors
      alt.ajouterEnsemble(Et(s))
    FinSi
  FinPour
  retourner alt
Fin P

// Sélectionne la meilleure sous-alternative des noeuds-et.
Début Et (x: NoeudEtat) -> Ensemble <NoeudInteraction>
  choisi : NoeudInteraction = Elagage (Fils(x))
  retourner A(choisi)
Fin Et

//Récolte les noeuds interaction parmi les fils issus d'un choix.
Début A (x: NoeudInteraction) -> Ensemble <NoeudInteraction>
  alt : Ensemble <NoeudInteraction> = vide
  alt.ajouter(x)
  Pour chaque Noeud s de Fils(x) faire
    alt.ajouterEnsemble(Et(s))
  FinPour
  retourner alt
Fin A
```

FIG. 4.5 – Algorithme récursif pour la collecte d'une alternative pour une interaction exécutable. L'algorithme se lance en appelant N avec en paramètre l'interaction exécutable (le nœud interaction exécutable de l'$Arbre - et/ou$.

par le mécanisme de sélection d'action. La sous-alternative gardée sera celle ayant obtenu la meilleure évaluation.

Cette heuristique permet de ne collecter qu'une seule alternative pour chaque interaction exécutable en effectuant des choix imposés par les $Noeuds - ou$. L'évaluation des sous-alternatives issues d'un $Noeud - ou$ est effectuée à l'aide d'un mécanisme de sélection d'action qui n'est pas basé sur l'ensemble des motivations utilisées pour l'évaluation des interactions exécutables. En effet, toutes les interactions d'une sous-alternative doivent être prises en compte pour l'élagage. C'est pourquoi, le mécanisme de sélection d'action utilisé pour l'élagage est composé uniquement des motivations évaluant les trois parties de l'alternative. De plus afin que l'évaluation d'une sous-alternative reflète l'évaluation de son alternative (par les mêmes motivations), j'impose aux évaluateurs sélectionnés pour l'élagage de respecter la règle d'additivité suivante : Soient I_1, I_2 et I_3 des interactions, I_1 et I_2 appartiennent à la même sous-alternative et I_1 et I_3 appartiennent à la même sous-alternative, une évaluation ϕ est *additive* si $\max(\phi(\{I_1, I_2\}), \phi(\{I_1, I_3\})) = \phi(\{I_1\}) + \max(\phi(\{I_2\}), \phi(\{I_3\})$.

Soient α une interaction exécutable, $(alt_i)_{i\in[1,n]}$ l'ensemble des alternatives de l'$Arbre - ou$ dont l'interaction exécutable est α et $ssalt$ la sous-alternative sélectionnée par l'élagage. Si les mêmes motivations sont utilisées pour la sélection d'action et pour l'élagage, nous pouvons dire que $ssalt \in \arg\max_{alt\in(alt_i)_{i\in[1,n]}}(\phi(alt))$. Puisque les évaluateurs utilisés pour l'élagage donnent des évaluations additives, la sous-alternative choisie pendant l'élagage correspond à l'alternative la mieux évaluée (avec les mêmes évaluateurs que l'élagage) parmi l'ensemble des alternatives issues de la même interaction exécutable.

Nous permettons à l'utilisateur de choisir le sous-ensemble de motivations à utiliser pour l'élagage lors de la construction des alternatives.

4.3 Mise en œuvre du mécanisme de sélection d'action

Dans CoCoA, il n'y avait pas réellement de mécanisme de sélection permettant de définir différents comportements. J'ai donc mis en place un mécanisme de sélection d'action basé sur les motivations. Dans cette partie nous allons reprendre les différentes étapes de conception que je préconise (dans la partie 3.3.1) afin de définir le mécanisme de sélection d'action que j'ai mises en place dans CoCoA. Je commencerai donc par une analyse des besoins qui débouchera sur l'identification des motivations. Puis je choisirai une fonction de combinaison pour enfin définir les évaluateurs pour chaque motivation. Évidemment, je propose un ensemble de motivations qui n'est pas clos, il est toujours possible de trouver d'autres motivations et de les combiner avec les motivations déjà présentes ou de sélectionner un sous-ensemble de ces motivations.

4.3.1 Analyse

Afin de dégager les motivations que j'ai mis en place dans CoCoA, nous allons considérer un contexte applicatif particulier qui est les jeux vidéo et notamment la modélisation de personnage non joueur (PNJ) dans un jeu vidéo du type RPG ou MMORPG. De plus, j'ai également pris en compte les critères que propose Toby Tyrrell[Tyr93b] dans sa thèse pour

évaluer un bon mécanisme de sélection d'action. Notre environnement étant virtuel, j'ai écarté les critères liés au senseurs et aux actionneurs.

Un personnage non joueur, comme un joueur, a le plus souvent plusieurs buts à gérer en concurrence. Ceux-ci vont de la résolution de simples "missions" à la gestion de ses propriétés "vitales". La granularité de ces buts et leur importance sont variables. Ils nécessitent la mise en place de comportements qui pourront être perçus tantôt comme cognitifs, lorsqu'il s'agira de construire un plan pour résoudre un but à moyen ou long terme comme c'est le cas des "missions", tantôt comme réactifs, lorsqu'il s'agira de prendre immédiatement un élément de l'environnement comme fuir un ennemi puissant qui vient d'apparaître. On comprend bien qu'une réorganisation des *priorités relatives des buts* doit donc être possible.

De plus, les agents évoluent dans un environnement géographique dont l'impact sur le comportement est nécessairement important. Ainsi la situation (ou position) de l'agent dans cet environnement a une influence sur ses choix, ne serait-ce que parce qu'il doit se déplacer dans cet environnement pour exécuter telle ou telle action à un endroit précis de cet environnement. La crédibilité du comportement perçu du PNJ sera en partie jugée sur ces déplacements, une gestion qui paraîtrait "déraisonnable" des déplacements pénaliserait le jugement, tant d'un point de vue local à un moment donné que sur la totalité des déplacements prévus pour résoudre ses buts. Ainsi, localement, lorsque deux actions équivalentes peuvent être exécutées, mais la cible de l'une est très éloignée alors que celle de l'autre est proche (dans son voisinage), on peut s'attendre à ce qu'un personnage choisisse celle dont *la cible est la plus proche*. De même entre deux résolutions de buts, la résolution dont *les déplacements prévus* sont moindre doit être privilégiée. Le mécanisme de sélection d'action doit donc prendre en compte l'environnement, ses déplacements et la notion de voisinage de l'agent.

Souvent, les actions d'un joueur ou d'un PNJ prennent un certain temps. Couper du bois ou ouvrir une porte sont deux actions dont les durées devraient être différentes pour paraitre réalistes. Le temps que prend chaque action doit avoir une influence sur le choix des actions à effectuer. Prendre en compte *le temps d'exécution* d'une action permettrait à l'agent de résoudre ses buts le plus rapidement possible peut être compris comme de l'efficacité ou parce qu'il veut se débarrasser le plus rapidement possible de son fardeau.

Nous souhaitons également pouvoir exprimer des traits de caractère différents pour les différents personnages. Les traits de caractère s'expriment à travers de nombreuses facettes et, d'une certaine manière, on peut considérer que tout facteur influençant la sélection d'action est un élément constituant ces traits de caractère. Cependant, pour des PNJ de jeux vidéo, ce que l'on veut à travers ses traits de caractère c'est également exprimer *des préférences*. Il faut donc considérer donc que les choix de l'action exécutée doivent exprimer ces préférences. Ou, inversement, que l'on peut vouloir transcrire des préférences en favorisant l'exécution de telle ou telle action. Notons que, comme pour des personnages de séries télévisées par exemple, les personnages de jeux vidéo sont assez stéréotypés car ils doivent être facilement "identifiés" par le joueur, et qu'en conséquence les traits de caractère sont le plus souvent très tranchés.

Toujours dans le but de paraitre crédible, un personnage doit éviter de tenter de faire plusieurs choses en même temps. Typiquement, si un personnage oscille entre deux déplacements (une fois à gauche pour effectuer une action, une fois à droite pour effectuer une autre action) son comportement ne sera pas jugé comme réaliste. Il faut donc éviter d'obtenir ce genre d'oscillation. En effet, il est assez rare de voir un joueur tenter de résoudre plusieurs

buts en même temps, surtout si ces buts n'ont absolument rien en commun (ni l'emplacement géographique, ni leur direction par rapport à l'emplacement de l'agent, ni même dans leur pré-requis). Toutefois, si plusieurs buts partagent des éléments communs, il semble réaliste de privilégier *les actions pouvant entamer plusieurs résolutions* en même temps en faisant "d'une pierre deux coups".

4.3.2 Le choix des motivations

À partir de l'analyse précédente, nous pouvons identifier sept motivations. Si on reprend les catégories de motivations définies par Ferber[Fer95], deux motivations sont liées à l'environnement (l'opportunisme et l'accomplissement en temps) et cinq sont liées à la personnalité de l'agent (la priorité des buts, l'accomplissement en temps, les préférences de l'agent, la revalorisation multi-buts et l'inertie). La motivation d'*opportunisme* privilégiant les cibles proches des interactions exécutables, l'*accomplissement en espace* prenant en compte les déplacements dans les résolutions. La *priorité des buts* comme son nom l'indique prend en compte les différentes priorités des buts de l'agent, l'*accomplissement en temps* favorise les résolutions courte en temps d'exécution, les *préférences de l'agent* favorise les interactions préférées par l'agent, la *revalorisation multi-buts* privilégie les actions pouvant entamer plusieurs résolutions en même temps et l'*inertie* avantage la continuité dans les résolutions. Dans CoCoA, je ne proposerai pas de motivation liée au comportement de groupe, la catégorie relationnelle n'a donc pas été exploitée.

L'opportunisme

La motivation d'opportunisme (ou *opportunisme*) a été décrite dans l'exemple de conception de motivation (voir la partie 3.3.1). Cette motivation est liée à l'environnement, elle privilégie les interactions dont la cible se trouve dans le voisinage proche de l'agent. Cette notion de voisinage dépend du profil d'individualité de l'agent, un seuil permet de définir si une cible est dans le voisinage de l'agent ou pas. Cette motivation s'intéresse donc à ce que l'agent **peut faire** immédiatement, c'est-à-dire l'interaction exécutable d'une alternative.

Nom de la motivation	Opportunisme
Type de la motivation	Environnementale
Champ de l'évaluation	Attraction et neutralité
Partie de l'alternative concernée	Action exécutable
Paramètre du profil d'individualité	Un seuil qui définit le voisinage de l'agent

L'accomplissement en espace

Contrairement à l'opportunisme qui privilégie l'aspect local de l'environnement, l'accomplissement en espace privilégie les résolutions ne demandant pas beaucoup de déplacements. Cette motivation liée à l'environnement, prend en compte toutes les parties d'une alternative pour en déduire le déplacement total en nombre de cases de cette alternative. La notion de "peu de déplacements" est subjective, c'est pourquoi cette notion dépendra du profil d'individualité

de l'agent. Un seuil en nombre de cases permettra de définir si une distance totale est courte (si elle est inférieure au seuil) ou non (si elle est supérieure au seuil). Contrairement à l'opportunisme qui ne peut que revaloriser l'évaluation globale d'une interaction exécutable (et de son alternative), l'accomplissement en espace privilégiera les résolutions courtes et dévalorisera l'évaluation globale des résolutions longues en nombre de déplacements.

Nom de la motivation	Accomplissement en espace
Type de la motivation	Environnementale
Champ de l'évaluation	Attraction, neutralité et répulsion
Partie de l'alternative concernée	Toute l'alternative
Paramètre du profil d'individualité	Un seuil qui définit une résolution courte en nombre de déplacements

La priorité des buts

Dans une simulation, un agent a des buts à résoudre. Parmi l'ensemble de ses buts, certains peuvent être plus ou moins prioritaires. Un but préservant la "survie" de l'agent peut être plus prioritaire qu'un but correspondant à la résolution d'une quête. De plus, la priorité d'un but peut également évoluer au cours de la simulation. Basées sur le même principe que les propriétés nous avons mis en place les priorités dynamiques des buts. La valeur d'une priorité d'un but est calculée à l'aide d'une fonction mathématique. La fonction utilisée pour la priorité peut également être constante, évolutive ou dépendante.

Il est alors possible de définir une priorité de buts qui concerne une propriété d'un agent. Ainsi la valeur de priorité du but dépendra de la valeur de la propriété et si la valeur de la propriété évolue au cours de la simulation (propriété évolutive) alors la valeur de la priorité du but évoluera au cours de la simulation.

Par exemple, nous pouvons définir un seuil à partir duquel l'agent doit prendre en compte son énergie dans ses choix. Si la valeur de l'énergie est au dessus du seuil, la priorité est de 0, en dessous moins l'agent a d'énergie, plus cette priorité est importante. Prenons arbitrairement un seuil d'énergie fixé à 20, le but de type *condition* de l'agent sera défini par `actor.energy > 20` et la fonction associée pourrait être une sigmoïde (voir la figure 4.6).

Ce qu'il faut retenir, c'est que pour un agent, ces priorités sont intrinsèquement variables, cette importance peut varier pendant la simulation et peut également dépendre d'une propriété de l'agent. Suivant la priorité du but, toutes les configurations d'évaluations sont possibles allant de l'attraction à l'inhibition. Cette motivation prend en compte la priorité du but dans le choix des actions à exécuter.

Nom de la motivation	Priorité des buts
Type de la motivation	Personnelle
Champ de l'évaluation	Attraction, neutralité, répulsion et inhibition
Partie de l'alternative concernée	Le but
Paramètre du profil d'individualité	Aucune, la priorité d'un but est déjà individuelle pour chaque agent

$$f(x) = \begin{cases} 0 & \text{si } x \geq 20 \\ MAX & \text{si } x \leq 0 \\ \frac{MAX}{1+\exp\frac{x-m}{\lambda}} & sinon \end{cases}$$

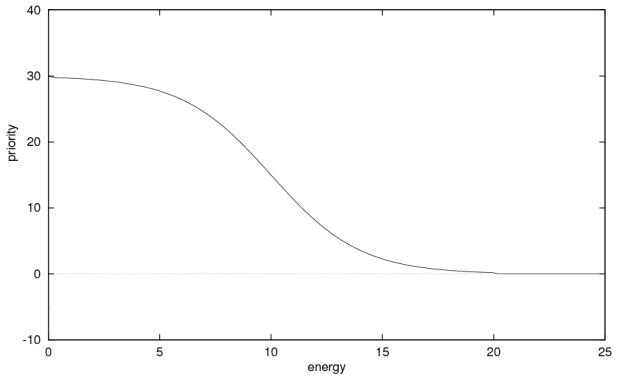


FIG. 4.6 – Exemple de fonction pour calculer la priorité du but "prendre en compte son énergie à partir de 20". La fonction f utilisée est basée sur une sigmoïd (avec MAX qui fixe la valeur maximale de la priorité du but , m peut modifier le centre de symétrie et λ contrôle la pente. Dans cet exemple, nous avons fixé $MAX = 30$, $m = 10$ et $\lambda = 2$.

L'accomplissement en temps

Dans un jeu, il est possible que certaines actions prennent plus de temps que d'autre à se réaliser. Cela peut dépendre de la nature de l'action, des caractéristiques du personnage effectuant l'action ou des compétences de se personnage dans un domaine spécifique (par exemple son niveau dans un métier). Par cette motivation, le coût en temps de chaque action est pris en compte afin de favoriser les résolutions prenant le moins de temps et de dévaloriser les résolutions prenant trop de temps. La notion de "peu de temps" est subjective, c'est pourquoi cette notion dépendra du profil d'individualité de l'agent. Un seuil en temps permettra de définir si un coût total en temps est court (s'il est inférieur au seuil) ou non (si il est supérieur au seuil). De plus, nous devons également inclure dans le profil d'individualité les coûts des interactions que l'agent peut effectuer, car ces coûts peuvent dépendre de l'agent lui-même.

Nom de la motivation	Accomplissement en temps
Type de la motivation	Personnelle
Champ de l'évaluation	Attraction, neutralité, répulsion et inhibition
Partie de l'alternative concernée	Toute l'alternative
Paramètre du profil d'individualité	Un seuil qui définit une résolution courte en nombre d'interactions à exécuter et le coût de chaque interaction que l'agent peut effectuer

Les préférences de l'agent

Nous voulons attribuer des traits de personnalité à notre agent, il peut par exemple être brutal, pacifique ou gourmand. Pour cela nous attribuons à chaque agent des préférences différentes sur les actions qu'il peut effectuer[12]. Ainsi, suivant ces préférences, cette motivation poussera l'agent à faire certains choix permettant à l'observateur d'identifier différents traits

[12] Un trait de personnalité est associé à un ensemble d'interactions.

de caractère. Par exemple, en attribuant une préférence plus forte pour *casser* que pour *ouvrir*, nous poussons l'agent à faire le choix de casser une porte plutôt que de l'ouvrir. Ainsi, l'observateur peut interpréter ce comportement, comme étant une conséquence du trait de caractère "brutal" de l'agent. Une préférence peut donc être une attraction, une répulsion, une inhibition ou une indifférence sur l'interaction.

L'évaluation de l'alternative par cette motivation se base sur les préférences de toutes les actions contenues dans l'alternative. Ces préférences sont combinées pour produire une valeur de préférence globale pour l'alternative.

Nom de la motivation	Accomplissement en temps
Type de la motivation	Personnelle
Champ de l'évaluation	Attraction, neutralité, répulsion et inhibition
Partie de l'alternative concernée	Toute l'alternative
Paramètre du profil d'individualité	Les préférences de l'agent pour chaque interaction qu'il peut effectuer

La revalorisation multi-buts

Une action exécutable qui apparaît dans plusieurs alternatives fait progresser simultanément plusieurs résolutions lorsqu'elle est exécutée. Si plusieurs résolutions amènent l'agent à aller dans une même direction ou à effectuer une même interaction, il semble rationnel que l'agent effectue cette interaction et faire d'une pierre n coups (n étant le nombre de résolutions ayant la même interaction exécutable). Cette motivation a pour but de favoriser ce type d'interactions. Plus l'interaction fait progresser un grand nombre d'alternatives, plus cette action est favorisée. Il faut donc définir dans le profil d'individualité de l'agent le bonus qu'apporte cette motivation et que ce dernier soit fonction du nombre d'alternatives présentant la même interaction exécutable.

Nom de la motivation	Revalorisation multi-buts
Type de la motivation	Personnelle
Champ de l'évaluation	Attraction et neutralité
Partie de l'alternative concernée	L'interaction exécutable
Paramètre du profil d'individualité	Le bonus de revalorisation en fonction du nombre de fois où l'interaction est présente comme interaction exécutable.

L'inertie

Lorsqu'on regarde dans la littérature sur la modélisation de comportement, une précaution revient systématiquement. Cette précaution a pour but d'éviter qu'un agent soit pris entre deux feux et n'arrive pas à se décider et oscille entre plusieurs résolutions. En effet, cette oscillation est un comportement non réaliste facilement décelable par un observateur. Une oscillation se produit lorsque les deux évaluations de deux interactions exécutables ont des valeurs très proches et qu'à chaque étape, de petits changements des conditions d'évaluation

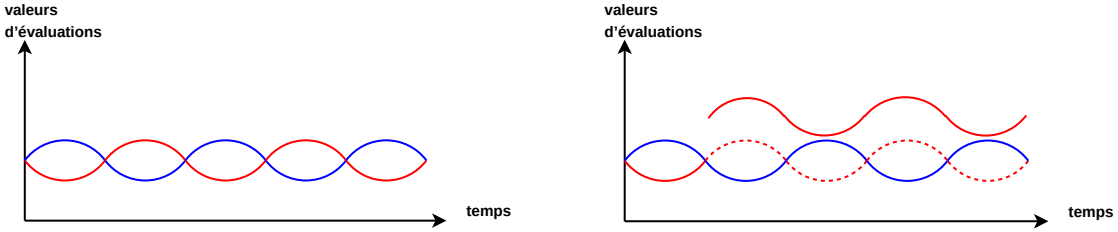


FIG. 4.7 – Les deux schémas représentent les courbes d'évaluations de deux alternatives (une rouge et une bleu). À gauche sans l'inertie les deux alternatives deviennent successivement celle avec la même évaluation. À droite, une fois que l'alternative rouge a été sélectionnée, elle obtient un bonus d'inertie qui lui permet d'améliorer son évaluation et permet d'éviter les oscillations dans la sélection de la meilleure alternative entre la bleu et la rouge.

(par exemple le changement de l'état de l'environnement) mène l'agent à échanger l'action à exécuter entre les deux meilleures. C'est pourquoi, quand un agent s'engage dans une résolution, la motivation d'inertie exprime la tendance de l'agent à poursuivre dans la même résolution. Pour cela cette motivation favorise uniquement l'interaction appartenant à la même alternative que la précédente interaction exécutée.

Le bonus apporté à l'alternative ayant reçu précédemment la meilleure évaluation permet d'accroître son évaluation et de créer un écart plus important entre la meilleure alternative et les autres (voir la figure 4.7). De plus, si une alternative alt_a dont l'évaluation profite du bonus d'inertie ne reçoit pas la meilleure évaluation (la meilleure étant reçue par l'alternative alt_b), au pas de simulation suivant alt_a perdra le bonus d'inertie au profit de l'alternative alt_b renforçant ainsi l'écart entre les deux alternatives.

Il faut noter que plus la valeur du bonus d'inertie est importante, moins les phénomènes d'oscillation apparaissent mais également moins l'agent changera de résolution. Ainsi, il est possible d'empêcher (ou de rendre difficile) le changement d'alternative pour un agent "borné" qui ne change jamais d'avis même temporairement en donnant une forte valeur au bonus d'inertie.

Nom de la motivation	Inertie
Type de la motivation	Personnelle
Champ de l'évaluation	Attraction et neutralité
Partie de l'alternative concernée	L'interaction exécutable
Paramètre du profil d'individualité	Le bonus d'inertie en fonction de la précédente interaction exécutée

4.3.3 Les fonctions de combinaison

Afin d'obtenir l'évaluation de chaque action exécutable, il faut pouvoir combiner les évaluations de chaque motivation. La fonction de combinaison choisie sera déterminante pour la construction des évaluateurs de chaque motivation. En effet, pour que les évaluations puissent avoir un sens, il est nécessaire de définir préalablement, un intervalle de valeurs dans lequel les évaluations pourront exprimer l'attraction, la neutralité, la répulsion et l'inhibition.

Dans CoCoA la fonction de combinaison utilisée répond à tous les critères définis dans la partie 3.3.1. La fonction sélectionnée a également l'avantage d'être "simple" montrant qu'il est possible d'obtenir des comportements facilement sans avoir de grandes connaissances en intelligence artificielle ou en mathématiques. Les choix que j'ai fait dans CoCoA pour la fonction de combinaison et pour les motivations ne sont qu'une illustration possible d'implémentation de ma proposition.

La fonction de combinaison, par définition, permet l'ajout et la suppression de motivations en gardant la cohérence de l'agrégation par rapport à l'individualité désirée. Elle doit donc être additive pour permettre la construction incrémentale du mécanisme de sélection d'action et l'élagage de l'*Arbre − et/ou* lors de la collecte des alternatives. Enfin, afin de marquer l'indépendance des motivations les unes par rapport aux autres, pendant leur évaluation ainsi que l'impact de chaque motivation sur l'évaluation finale, elle doit vérifier les critères définis par la propriété 3.1.

De ces critères nous pouvons exclure les moyennes comme fonction de combinaison. En effet, si une attraction doit toujours améliorer la moyenne, il est donc impossible de pouvoir définir une attraction sans connaître la moyenne courante (afin de pouvoir l'améliorer). Ceci est également vrai pour une répulsion et dans le cas général, pour une évaluation non neutre et non inhibitrice, l'ajout d'une évaluation doit avoir un impact sur l'évaluation globale sans en connaître l'évaluation courante a priori. Pour les motivations, il ne s'agit donc pas de moyenner les évaluations mais bien de les cumuler afin d'être certain qu'une attraction (resp. une répulsion) augmentera (resp. diminuera) toujours la valeur finale de *Comb*. De plus, le calcul d'une moyenne se base généralement sur le nombre d'éléments (la cardinalité) à moyenner. Pour la fonction de combinaison, ce nombre ne devrait pas être utilisé, car l'ajout et la suppression d'une motivation neutre aurait un impact sur la moyenne finale. Ce point peut néanmoins être réglé en écartant les évaluations neutres du calcul final. Toutefois, il existe un autre point qui ne permet pas d'utiliser ce nombre qui est la construction incrémentale de l'ASM où par définition le nombre total de motivations n'est pas fixé.

J'ai donc commencé par rechercher différentes fonctions de combinaison "simples". Mais choisir une fonction combinaison n'est pas suffisant. Pour pouvoir satisfaire les critères précédents, il faut en même temps définir, le domaine de définition de la fonction et un intervalle de valeurs possibles pour les évaluations. Une fois cet intervalle défini, il faut déterminer quelles sont les valeurs permettant d'exprimer les évaluations attractives, répulsives, neutres et inhibitrices. Pour le domaine, j'ai choisi de travailler dans $\mathbb{R}$. Ensuite, suivant la fonction de combinaison traitée, il faut définir un sous-domaine de $\mathbb{R}$.

Le choix de la fonction de combinaison

La description et mon évaluation de chaque fonction combinaison sont présentées dans l'annexe A et résumées dans le tableau 4.1.

J'ai choisi d'utiliser dans CoCoA le produit dans l'intervalle $[0, N]$ comme fonction de combinaison parmi les fonctions testées. Cette fonction est simple, les éléments neutre et absorbant de la multiplication forment respectivement les valeurs de l'évaluation neutre et de l'inhibition. Pour chaque évaluation attractive, il est possible d'obtenir une évaluation répulsive annulant l'attraction. La multiplication permet l'ajout et la suppression de motivations sans perturba-

méthode de composition	notation	description
la fonction puissance	$\phi'(alt) = (\phi(alt))^{\gamma_m(alt)}$ avec $\phi_0(alt) > 1$	Inhibition pour $\gamma_m(alt) = 0$ et évaluation neutre pour $\gamma_m(alt) = 1$, attraction pour $\gamma_m(alt) > 1$ et répulsion pour $\gamma_m(alt) \in]0, 1[$. Les valeurs manipulées deviennent très importantes avec le nombre d'évaluateurs ce qui ne facilite pas leur manipulation.
la fonction racine	$\phi'(alt) = \sqrt[\gamma_m(alt)]{(\phi(alt))}$	Il n'existe pas de valeur d'inhibition, comme la fonction puissance la manipulation de grands nombres n'est pas justifiée. Le sens donné aux évaluations n'est pas évident à comprendre
la somme	$\phi'(alt) = \phi(alt) + \gamma_m(alt)$	Il n'existe pas de valeur d'inhibition, l'évaluation neutre pour l'élément neutre de l'addition ($\gamma_m(alt) = 0$), l'attraction pour ($\gamma_m(alt) > 0$) et répulsion pour ($\gamma_m(alt) < 0$)
la somme probabiliste	$\phi'(alt) = (\phi(alt) + \gamma_m(alt)) - (\phi(alt) * \gamma_m(alt))$	Il n'existe pas de valeur d'inhibition, l'évaluation neutre $\gamma_m(alt) = 0$, l'attraction pour ($\gamma_m(alt) \in]0, 1[$) et répulsion pour ($\gamma_m(alt) \in]-1, 0[$)
le produit	$\phi'(alt) = \phi(alt) * \gamma_m(alt)$	Inhibition pour l'élément absorbant ($\gamma_m(alt) = 0$), l'évaluation neutre pour l'élément neutre ($\gamma_m(alt) = 1$), l'attraction pour ($\gamma_m(alt) > 1$) et répulsion pour ($\gamma_m(alt) \in]0, 1[$)

TAB. 4.1 – Comparatif des fonctions de composition *Comb* (détail en annexe A).

tion (le nombre de motivations n'entrant pas dans le calcul de la fonction de combinaison) et elle préserve l'indépendance des motivations. Pour finir, la multiplication respecte les critères de la définition d'une fonction de combinaison :

- La multiplication est additive et l'attraction, la répulsion, la neutralité et l'inhibition sont exprimables.
- L'attraction correspond à $\gamma_m(alt) \in]1, N]$, dans ce cas l'évaluation globale est augmentée, $\phi'(alt) = \phi(alt) * \gamma_m(alt) > \phi(alt)$.
- La répulsion correspond à $\gamma_m(alt) \in]0, 1[$, dans ce cas l'évaluation globale est diminuée, $\phi'(alt) = \phi(alt) * \gamma_m(alt) < \phi(alt)$.
- La neutralité correspond à $\gamma_m(alt) = 1$, dans ce cas l'évaluation globale est pas affectée par la motivation, $\phi'(alt) = \phi(alt) * 1 = \phi(alt)$.
- L'inhibition correspond à $\gamma_m(alt) = 0$, dans ce cas l'évaluation globale est inhibée, $\phi'(alt) = \phi(alt) * 0 = \gamma_m(alt) = 0$.

4.3.4 Les évaluateurs

Maintenant, que la fonction de combinaison et l'intervalle de valeurs pour CoCoA ont été choisis, je vais vous présenter les évaluateurs correspondant à chaque motivation sélectionnée. Encore une fois, ces évaluateurs ne sont qu'une illustration possible de notre modèle que j'ai mis en place dans CoCoA. Les évaluateurs sont des fonctions qui pour un agent et une alternative donnés fournissent une évaluation. Cette évaluation peut exprimer une attraction, une neutralité, une répulsion ou une inhibition suivant l'interprétation que l'on peut faire de la valeur avec la fonction de combinaison *produit*.

Pour chaque évaluateur, considérons une alternative $alt = (g, \alpha, (a_i)_{i\in[1,n]})$, où g le but de l'alternative, α l'interaction exécutable et a_i les autres actions à exécuter de l'alternative. $\mathcal{A}^R$ est l'ensemble des interactions exécutables au moment de la sélection d'action.

L'opportunisme

L'opportunisme est la motivation illustrant le principe de conception du mécanisme de sélection d'action basé sur les motivations. Pour rappel :

$$\gamma_{opp}(alt) = \begin{cases} 1 \text{ si } \theta_{opp} \leq 1 \\ 2 \text{ si } dist(target(\alpha)) = 0 \\ \max\left(1, 1 + \log_{\theta_{opp}}\left(\frac{\theta_{opp}}{dist(target(\alpha))}\right)\right) \text{ sinon} \end{cases}$$

où
- *target* est une fonction qui fournit la cible de l'action α,
- *dist* est une fonction qui calcule la distance en nombre de mouvements entre l'agent, acteur de l'interaction, et un élément (ici la cible de l'interaction),
- θ_{opp} est la portée de l'influence de l'opportunisme.

Le paramètre θ_{opp} représente la notion de voisinage de l'agent afin d'identifier les autres agents qui sont proches de lui pour favoriser les interactions sur ceux-ci. Alors les paramètres π_{γ_opp} du profil comportemental pour l'opportunisme correspondent au paramètre θ_{opp}, $\pi_{\gamma_opp} = \{\theta_{opp}\}$.

L'accomplissement en espace

Cette fonction évalue les déplacements nécessaires à la résolution d'une alternative. L'accomplissement en espace est paramétrée par un seuil en nombre de déplacements $\pi_{\gamma_{accS}} = \{\theta_{accS}\}$. Si l'estimation de la résolution de l'alternative est supérieure que ce seuil, l'évaluation sera répulsive, en-dessous elle sera attractive. Comme pour l'opportunisme, j'utilise un logarithme afin borner la valeur maximale de l'évaluation.

$$\gamma_{accS}(alt) = \begin{cases} 1 \text{ si } \theta_{accS} \leq 1 \\ 2 \text{ si } nbSteps(alt) = 0 \\ \log_{\theta_{accS}}(1 + \frac{(\theta_{accS} * (\theta_{accS} - 1))}{nbSteps(alt)}) \text{ sinon} \end{cases}$$

La fonction *nbSteps* a été créée pour calculer le nombre de déplacements nécessaires pour résoudre une alternative. Ce nombre de déplacements est une estimation des pas de dépla-

cement basée sur les emplacements des cibles des interactions d'après les connaissances de l'agent au moment de l'évaluation. Ainsi elle effectue la somme en nombre de cases des distances entre l'agent et la cible de la première interaction, avec la distance entre la cible de la première interaction avec la cible de la deuxième interaction et ainsi de suite jusqu'à atteindre la cible de la dernière interaction de l'alternative.

Dans CoCoA, les plans ne présentent pas les séquence de déplacements unitaires pour atteindre une cible. En effet, un déplacement vers une cible qui se trouve à n cases, ne va pas être représenté par n interactions de déplacement. Indépendamment du nombre de cases que nécessite un déplacement, ce dernier sera décomposé en deux interactions $MoveTo$ et $MoveOneSteep$. Cette décomposition peut se comprendre comme ceci, pour se rendre à un emplacement ou rejoindre un autre agent, il faut d'abord effectuer un premier pas. Il ne s'agit donc pas pour cette motivation de compter le nombre d'interactions de déplacement mais de faire *une estimation* des déplacements suivant les connaissances de l'agent au moment de l'évaluation.

Comme nous l'avons vu précédemment, la collecte des actions de l'alternative ne prend pas en compte la notion d'ordre des actions. En effet, dans un $Arbre-et/ou$, pour un $Noeud-et$ nous ne pouvons pas toujours savoir dans quel ordre les fils de ce nœud seront traités. C'est pourquoi, le calcul des distances permettant l'évaluation de l'accomplissement en espace est une estimation afin de donner un ordre d'idée sur le coût en déplacements de la réalisation de l'alternative.

De plus, le calcul même de la distance entre l'acteur et la cible de l'interaction est une estimation, puisqu'elle est basée sur une prévision. En effet, ce calcul est basé sur les informations contenues dans la mémoire au moment de l'évaluation de l'alternative (comme une photo instantanée de la mémoire). La mémoire étant une version dégradée de l'environnement, elle peut contenir des erreurs par rapport à l'état réel de l'environnement, ainsi les emplacements des différents agents cible, s'ils ne sont pas dans le champ de perception de l'agent, peuvent être erronés.

Enfin, il faut savoir que les cibles des interactions peuvent être des agents mobiles (ils ont la capacité de se déplacer dans l'environnement). Or au moment où l'évaluateur demande le calcul des distances, ces dernières seront les distances entre les emplacements des cibles dans la mémoire et au moment du calcul. Par exemple, si un agent c doit effectuer les interactions i_1, i_2 et i_3 d'une alternative, dont les cibles prévues par le plan sont respectivement c_1, c_2 et c_3. Le calcul de la distance à parcourir pour résoudre cette alternative est la somme de la distance entre l'emplacement c et l'emplacement de c_1, la distance entre l'emplacement c_1 et l'emplacement de c_2 et la distance entre l'emplacement c_2 et l'emplacement de c_3. Si un agent cible est mobile, il a donc la capacité de changer d'emplacement pendant l'exécution des différentes actions qui précèdent l'action dont il est la cible, rendant erronée l'estimation de la distance totale. Notons toutefois, que pour ce dernier cas, l'estimation des distances peut être améliorée en ajoutant une prédiction sur les emplacements des agents cible, notamment comme les travaux de Cyril Septseault[Sep07] qui permettent la mise à jour de la mémoire de manière introspective en se basant sur la connaissance de la dynamique du comportement des autres agents.

La priorité des buts

Dans une simulation, l'agent doit résoudre des buts de priorités différentes. Ces priorités sont mises en place en fonction de l'agent et de l'importance des différents buts. Elles définissent donc une partie du comportement de l'agent, puisque nos agents sont dirigés par les buts. Cette priorité doit donc être prise en compte. Une fois cette priorité définie dans CoCoA, la fonction d'évaluation est assez triviale.

$$\gamma_{goal}(alt) = priority(g)$$

où *priority* représente la fonction donnant la valeur de la priorité du but de l'alternative et $\pi_{\gamma_{goal}} = \emptyset$.

L'accomplissement en temps

Cette motivation évalue le coût d'exécution de toutes les interactions présentes dans l'alternative. Dans un contexte de jeu vidéo, ce coût peut représenter le temps d'exécution de l'action (ou de l'animation). Comme l'accomplissement en espace, cette fonction est paramétrée par un seuil (θ_{accT}). Si le coût total de toutes les interactions de l'alternative est inférieur à ce seuil, l'évaluation sera attractive, si le coût est supérieur, l'évaluation sera répulsive. Cette motivation nécessite également de définir pour l'agent et pour chaque interaction son coût d'exécution. Par défaut, nous définissons le coût minimal comme étant 1, suivant l'application de la simulation, cette valeur peut représenter un coût en temps ou en ressource et pourquoi pas en argent. La somme des coûts des interactions de l'alternative *alt* étant calculée par la fonction $cost(alt)$.

$$\gamma_{accT}(alt) = \begin{cases} 1 \text{ si } \theta_{accT} \leq 1 \\ \log_{\theta_{accT}}(1 + \frac{(\theta_{accT} * (\theta_{accT} - 1))}{cost(alt)}) \text{ sinon} \end{cases}$$

et $\pi_{\gamma_{accT}} = \{\theta_{accT}\}$.

Le coût d'une interaction n'est jamais nul (i.e. $cost(alt) = 0$ est impossible), mais selon le contexte d'utilisation, ce coût ne doit pas forcément être défini par l'utilisateur pendant la création du profil d'individualité. Prenons comme exemple le MMORPG Dofus, dans ce jeu les joueurs ont des métiers, chaque métier a un niveau et plus le joueur exécute des actions liées à son métier plus son niveau dans ce métier augmente. Dans les métiers dit de "récolte" comme bûcheron, mineur ou paysan, le joueur doit attendre un laps de temps spécifique pour qu'une récolte soit effectuée, ce temps dépend du niveau du joueur dans le métier (voir tableau 4.2). Par exemple, lorsque le joueur est bûcheron niveau 1, le temps de coupe du frêne est de 11.9 secondes, lorsque pour le joueur qui est bûcheron de niveau 100, le temps de coupe est de 2 secondes. Ainsi le joueur possède une propriété niveau dans chaque métier qu'il pratique. Selon son niveau, la durée de la récolte est automatiquement calculée. Dans d'autres jeux vidéo ce principe existe également, par exemple dans les jeux du type RPG médiéval-fantastique la durée d'incantation (de chargement) d'un sort dépend du niveau du sort possédé par le joueur, ainsi que de ses caractéristiques comme ses valeurs en intelligence et en esprit. Sur ce même principe, le profil d'individualité de l'accomplissement en temps peut être réduit au seuil de la motivation, si les propriétés de l'agent permettent de calculer automatiquement le temps d'exécution des actions. Ceci allégerait la tâche du concepteur en automatisant une partie du paramétrage du profil d'individualité.

Niveau du métier bucheron	1	10	20	30	40	50	60	70	80	90	100
Temps de coupe (en secondes)	11.9	11	10	9	8	7	6	5	4	3	2

TAB. 4.2 – Temps de coupe du bois (du Frêne) en secondes requis en fonction du niveau dans le métier de bûcheron du personnage dans Dofus.

Les préférences de l'agent

Cet évaluateur prend en compte les préférences de l'agent sur les interactions contenues dans l'alternative. Comme pour l'accomplissement en temps, il faut agréger les différentes valeurs pour chaque alternative en utilisant une fonction de combinaison respectant la définition de la motivation. Pour l'accomplissement en temps la somme est une évidence, mais les préférences nécessitent une fonction de combinaison un peu plus complexe. De plus, contrairement à la combinaison des évaluations (la fonction $Comb$), les préférences ne sont pas indépendantes, car elles peuvent se compenser. Ainsi un agent peut effectuer une interaction qu'il n'aime pas s'il l'effectue après des interactions qu'il aime bien. Une préférence est exprimée par rapport à une autre. Il faut donc pouvoir exprimer les interactions que l'agent aime faire et celles qu'il n'aime pas. Nous retrouvons donc l'idée d'attraction et de répulsion que peut exprimer cette motivation. La neutralité et l'inhibition ont également un sens. La neutralité ou l'indifférence est une préférence exprimant que l'exécution ou non de l'interaction n'a pas d'importance pour l'agent (cette préférence ne doit donc pas influencer l'évaluation de l'alternative). L'inhibition est une préférence bloquant toute l'alternative, car cette interaction (et par conséquent l'alternative) ne doit pas être exécutée. Afin d'anticiper sur le moyen terme, les préférences de l'agent sur les actions à exécuter, toutes les actions a_i l'alternative alt sont considérées.

Maintenant que le sens des valeurs des préférences a été défini et que l'impact de ces préférences sur l'évaluation de l'alternative, il faut choisir une fonction pour combiner les préférences des actions d'une alternative. La présentation des différentes fonctions pour combiner les préférences est effectuée dans l'annexe B. La préférence pour une alternative représente une préférence globale pour toutes les interactions de l'alternative. Les préférences pouvant se compenser et n'étant pas indépendantes, j'ai privilégié le fait d'obtenir une valeur "moyenne" pour la préférence pour chaque alternative. Toutefois, une action inhibitrice doit toujours inhiber l'alternative et une action neutre ne doit pas influencer la valeur finale.

Ainsi, pour un agent c, la fonction π_c donne pour chaque interaction a, la valeur de la préférence de l'agent $\pi_c(a)$ et ses valeurs sont incluses dans $\mathbb{R}^+$:

- $\pi_c(a) = 0$ exprime une inhibition pour l'interaction a : l'agent ne veut absolument pas effectuer l'interaction a,
- $0 < \pi_c(a) < 1$ désigne une répulsion qu'a l'agent c à exécuter a,
- $\pi_c(a) = 1$ dénote l'indifférence de l'agent c quand à l'exécution de l'interaction a,
- $\pi_c(a) > 1$ marque l'attirance de l'agent c pour l'exécution de a.

Dans CoCoA, je laisse la possibilité à un utilisateur averti de choisir entre la moyenne harmonique et la moyenne géométrique (via une configuration d'un fichier xml). Par défaut ou par l'utilisation de l'interface graphique (voir la partie 5.1), la moyenne harmonique est choisie. Que ce soit pour la moyenne harmonique ou pour la moyenne géométrique, les moyennes se basent sur le nombre d'éléments à évaluer pour définir leur valeur. Ainsi, dans le but de garder le fait qu'une préférence neutre qui n'influence pas l'évaluation des préférences de l'agent, nous définissons l'ensemble $pref$ des interactions avec une préférence non neutre :

$$pref(alt) = \{a_i \in alt \mid \pi_c(a_i) \neq 1\}$$

Ainsi, si la moyenne harmonique Moy_{harm} est définie comme ceci :

$$Moy_{harm}(X) = \frac{\|X\|}{\sum_{x_i \in X} \left(\frac{1}{x_i}\right)}$$

Alors la préférence de l'agent γ_{pref} est défini comme :

$$\gamma_{pref}(\alpha) = \begin{cases} 0 \text{ if } \exists a_i \mid \pi_c(a_i) = 0 \\ Moy_{harm}(pref(alt)) \text{sinon} \end{cases}$$

Notons qu'il est possible que l'utilisateur veuille aller plus loin dans le détail des préférences de son agent, en spécifiant des préférences sur les actions et sur les cibles des interactions. Par exemple, l'agent peut préférer manger des pommes, plutôt que de manger des épinards (même si les épinards sont bons pour la santé). Pour cela il existe plusieurs façons de faire. Soit spécifier pour chaque couple (interaction, agent) une valeur de préférence, soit ajouter une nouvelle motivation de préférence qui se base uniquement sur les cibles et qui viendrait compléter la motivation actuelle. Nous préférons la deuxième solution qui rend moins fastidieux le paramétrage du profil comportemental (s'il y a N interactions et M cible, la première solution demande $N * M$ opérations alors que la deuxième en demande $N + M$) et exploite notre perception de notre modèle qui se base sur la combinaison de motivations indépendantes pour sélectionner la meilleure action à exécuter. En effet, si nous reprenons l'exemple précédent, si l'interaction manger est favorisée, et si la cible pomme est plus favorisée que la cible épinard, alors la combinaison des deux motivations, favorisera naturellement l'interaction manger(pomme) plutôt que l'interaction manger(épinard). Dans CoCoA, j'ai décidé d'assigner à chaque interaction une préférence sans préciser la cible de l'interaction et je me limiterai aux motivations déjà présentées. Toutefois, comme la construction est incrémentale il est possible d'ajouter une motivation de préférence de l'agent sur les cibles ou de redéfinir les préférences pour chaque couple (interaction, agent).

La revalorisation multi-buts

La revalorisation multi-buts favorise les interactions exécutables présentes dans plusieurs alternatives. Dans CoCoA, la collecte des alternatives permet de connaître toutes les alternatives présentes pour un tour donné. Cette évaluation nécessite de compter le nombre d'occurrences d'une interaction exécutable α de l'alternative alt avec la même cible et présente comme interaction exécutable dans les autres alternatives. Ce nombre est calculé en comparant les interactions exécutables. Ce calcul est effectué par $nbmulti(\alpha)$ et la revalorisation applique comme bonus :

$$\gamma_{multi}(alt) = \theta_{mrg} * (1 + \log(nbmulti(\alpha)))$$

Avec $\pi_{\gamma_{rmg}} = \{\theta_{rmg}\}$, le paramètre θ_{rmg} permet de définir l'impact du bonus de la revalorisation multi-buts. Dans CoCoA, j'ai fixé par défaut, la valeur de θ_{rmg} à 1, ce qui limite le bonus maximal à 2 (comme l'opportunisme et l'accomplissement en temps et en espace).

L'inertie

L'inertie favorise la continuité dans une résolution. Une interaction exécutable doit être favorisée par l'inertie si elle fait partie de l'alternative de l'interaction exécutée au tour suivant. Il est donc nécessaire de mémoriser la dernière alternative sélectionnée par le mécanisme de sélection d'action (noté *lsa* pour last selected alternative). Néanmoins à chaque tour l'agent met à jour les informations perçues de son environnement dans sa mémoire. Les plans peuvent donc être modifiés au cours de la simulation. Si un plan est modifié par l'apport de nouvelles informations perçues par l'agent, les alternatives correspondantes au plan peuvent donc être modifiées également. J'ai donc défini une heuristique afin de déterminer la similitude entre deux alternatives (voir la figure 4.9).

La modification d'un plan peut entrainer l'ajout et/ou la suppression d'interactions à exécuter. Afin de ne pas complexifier la recherche d'inertie et de ne pas effectuer trop de comparaisons entre deux alternatives pour déterminer la similitude, je gère l'ajout et la suppression d'interactions par l'algorithme en figure 4.8. Dans le cas 1 où l'alternative a perdu des interactions à exécuter (c'est le cas général puisqu'une interaction a été exécutée), l'algorithme recherche si l'interaction exécutable courante appartient à la *lsa* qui a été mémorisée. Dans le cas 2, où l'alternative a gagné des interactions à exécuter, l'algorithme recherche si l'interaction père de la précédente interaction exécutée (c'est-à-dire l'interaction qui aurait dû être exécutée) appartient à l'alternative courante. Cet algorithme permet de découvrir si l'alternative courante est similaire à l'alternative précédente ou non avec seulement 2 parcours d'alternative (le parcours de la *lsa* pour le cas 1 et le parcours de l'alternative courante pour le cas 2). Tous les cas ne sont pas gérés, mais cette recherche a une complexité linéaire (2 parcours d'alternative) et dans le cas où l'alternative change trop, on peut présumer que les alternatives ne sont pas similaires.

$$\gamma_{inertia}(alt) = \begin{cases} 1 + \theta_{inertia} & \text{si } inertie \\ 1 & \text{sinon} \end{cases}$$

Le paramétrage $\pi_{\gamma_{inertia}} = \{\theta_{inertia}\}$. Ce paramétrage n'est pas limité, dans les simulations que j'ai effectuées dans CoCoA j'utilise principalement un bonus d'inertie de 0.10 qui permet de valoriser l'alternative assez pour éviter les oscillations sans pour autant bloquer le changement d'alternative. Toutefois, il est possible de définir volontairement un bonus d'inertie très important afin de concevoir le comportement buté d'un agent qui ne veut pas changer d'alternative et qui reste dans une résolution une fois qu'elle a été choisie.

4.3.5 Réutilisation de l'ASM en dehors de CoCoA

Comme je l'ai présenté dans ce chapitre, certains éléments ont dû être créés dans CoCoA afin de pouvoir mettre en place mon mécanisme de sélection d'action. Il serait sûrement de même pour une utilisation en dehors de CoCoA. En effet, afin de pouvoir utiliser pleinement mon mécanisme de sélection d'action, certains pré-requis sont nécessaires. Premièrement, il faut pouvoir extraire les alternatives de la partie raisonnement du comportement. J'ai présenté un algorithme permettant d'extraire ses alternatives à partir d'un $Arbre - et/ou$. Suivant la structure de données utilisées pour la planification cet algorithme doit être adapté. Ces alternatives doivent être constituées du but, de l'action exécutable et des autres interactions

```
Algorithme de recherche d'inertie

LSA : la précédente alternative sélectionnée
pere-a : l'interaction père de la précédente interaction exécutable appartenant à LSA
alt : l'alternative courante
alpha : l'interaction exécutable courante appartenant à alt


Si la LSA existe et est définie alors

  // CAS 1
  Pour chaque interaction i de LSA
    Si i = alpha alors
      retourner VRAI.
    FinSi
  FinPour

  Si pere-a existe et est défini alors

    // CAS 2
    Pour chaque interaction i de alt
      Si i = pere-a alors
        retourner VRAI.
      FinSi
    FinPour
  FinSi

FinSi

// Les cas négatifs
retourner FAUX.
```

FIG. 4.8 – Algorithme de recherche d'inertie. Cet algorithme prend en compte les deux cas de la figure 4.9.

permettant de résoudre le but. La notion d'action exécutable doit être fournie par le raisonnement. Ensuite, il est important de pouvoir combiner les motivations, pour cela il faut implémenter une fonction de combinaison *Comb*. Dans l'annexe A, je teste différentes fonctions de combinaison et je détermine deux fonctions que je juge être de bonnes candidates. Enfin, il faut pouvoir mettre en place chaque motivation en tenant compte de leur besoin. Ses besoins sont définis dans le tableau 4.3.

Conclusion

Dans cette partie, j'ai présenté l'implémentation de ma proposition dans CoCoA. Dans un premier temps, j'ai mis en place les propriétés dynamiques permettant notamment de représenter des variables homéostatiques. Ces propriétés offrant des possibilités supplémentaires de modélisation et d'évolution du comportement des agents. Puis, nous avons vu que la partie raisonnement nécessitait la mise en place d'un algorithme collectant les alternatives dans

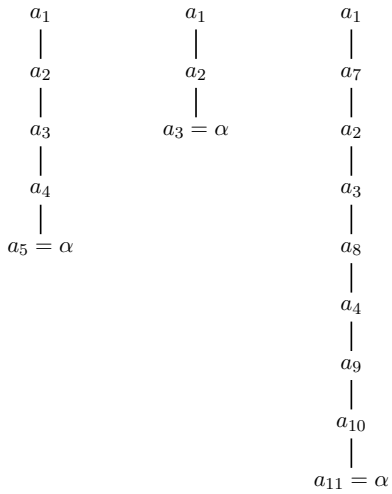


FIG. 4.9 – Prise en compte de l'inertie. À gauche, l'alternative initiale qui sera exécutée. Au tour suivant, deux cas sont possibles. Le cas 1, au centre, représente la même alternative au tour suivant mais avec moins d'interactions à exécuter que prévu. Le cas 2, à droite, représente l'alternative de gauche au tour suivant dans le cas où elle contient plus d'interactions à exécuter.

l'*Arbre – et/ou* de planification. Cet algorithme permet, à partir des interactions exécutables calculées par le planificateur, de retrouver les alternatives correspondantes. Enfin j'ai présenté le mécanisme de sélection d'action basé sur les motivations qui a été mis en place dans CoCoA. La construction de l'ASM est incrémentale, les motivations sont indépendantes et le mécanisme utilisé dans CoCoA n'est pas clos. Ce mécanisme permet de montrer qu'il est possible de construire un mécanisme de sélection d'action basé sur les motivations simples tout en définissant différents comportements. De plus, ce mécanisme répond aux critères de Tyrrell définissant un bon mécanisme de sélection d'action (voir le tableau 4.4).

Ces implémentations modifient l'architecture d'un agent CoCoA (voir la figure 4.10), où le raisonnement géré par le planificateur est une partie du comportement et l'individualité gérée par le mécanisme de sélection d'action en est l'autre. Ces deux modules forment une continuité (via la notion d'alternative) et permettent d'identifier l'interaction à exécuter et donc de définir le comportement observable de l'agent.

Nom de la motivation	Prérequis
L'opportunisme	Calculer la distance entre l'acteur et la cible d'une action exécutable
L'accomplissement en espace	Définir pour chaque action que l'agent peut effectuer un coût en temps
La priorité des buts	Créer des priorités donc la valeur est évolutive ou/et dépendante d'une priorité de l'agent
L'accomplissement en temps	Pouvoir estimer la distance totale nécessaire à l'exécution d'une alternative en fonction des connaissances courantes de l'agent
Les préférences de l'agent	Définir pour chaque action que l'agent peut effectuer une préférence qui est une attirance, une répulsion, une inhibition ou une indifférence (neutralité)
La revalorisation multi-buts	Déterminer si une action exécutable est présente dans plusieurs alternatives. Ceci est effectué pendant l'extraction de l'alternative correspondant à l'action exécutable
L'inertie	Définir si une action exécutable appartient à la même alternative que la précédente action exécutée. Il faut pour cela mémoriser la précédente alternative dont l'action exécutable a été sélectionnée et déterminer une notion de similitude avec l'alternative correspondant à l'action exécutable courante.

TAB. 4.3 – Tableau récapitulatif des pré-requis à la mise en place des motivations pour le mécanisme de sélection d'action.

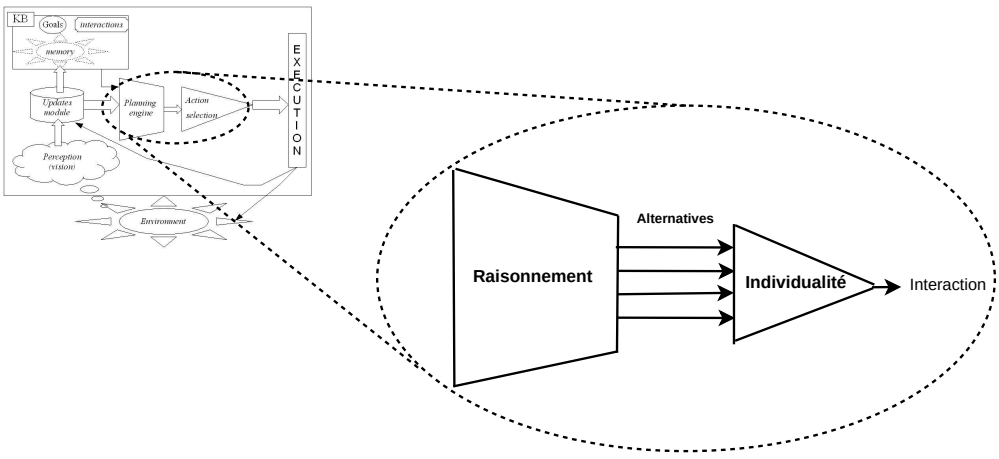


FIG. 4.10 – Architecture d'un agent actif CoCoA après l'implémentation de ma proposition pour la conception de comportements.

Critères de Tyrrell	L'ASM de CoCoA
Persistance jusqu'à l'achèvement d'une action	La prise en compte de l'alternative dans l'évaluation d'une action exécutable et l'inertie permettent d'éviter les oscillations entre différentes alternatives pendant les déplacements
Activation proportionnelle à l'état courant	L'influence des buts est une fonction variable ou constante dont la valeur peut être dépendante d'une propriété de l'agent qui peut définir une variable homéostatique
Concurrence équilibrée entre les actions	Pas de discrimination pour les interactions achevant un seul but, mais revalorisation multi-buts
Continuité des séquences d'actions	L'inertie est transmise au père de l'interaction exécutée, l'évaluation des interactions exécutables se fait sur l'ensemble des actions de l'alternative permettant de ne pas osciller
Candidat du compromis	L'interaction sélectionnée est la meilleure solution de la combinaison de toutes les motivations, aucune motivation n'impose son choix (la meilleure évaluation d'une motivation ne donne pas forcement la meilleure évaluation de toutes motivations, pas de winner-take-all entre les motivations)
Interruptibilité si nécessaire	L'ASM est défini dans son ensemble, la fonction de combinaison permet de définir pour chaque attraction une répulsion permettant l'annulation de l'attraction. De plus je préconise l'utilisation d'une inertie modérée afin de permettre l'interruptibilité notamment par les motivations liée à l'environnement comme l'opportunisme
Profiter de l'avantage qu'offrent des opportunités	L'opportunisme que je propose a les mêmes propriétés
Combinaison des préférences	La notion d'alternative permet d'évaluer toutes les interactions d'une alternative. Les "préférences de l'agent" est une motivation prenant en compte les trois parties d'une alternative.
Combinaison flexible des stimuli	Plusieurs combinaisons sont possibles pour la personnalité, la fonction de combinaison *Comb* permet l'ajout et la suppression de motivations sans perturber le calcul tout en préservant l'indépendance des motivations

TAB. 4.4 – Critères de Tyrrell respectés par l'ASM de CoCoA.

Chapitre 5

Concevoir des comportements

« J'ai toujours été frappé par le comportement d'ivrogne des enfants en bas âge : ils bégaient, titubent, trébuchent, passent sans transition du rire aux larmes et réciproquement. Qu'est-ce que ce serait si, en plus, ils buvaient de l'alcool ! »
Roland Topor, Pense-bêtes

Dans ce chapitre nous aborderons la conception de comportement du point de vue utilisateur. Je présenterai d'abord, l'atelier de conception qui a été mis en place afin de permettre à l'utilisateur de concevoir des comportements indépendamment des simulations. Puis, je me focaliserai sur la partie simulation de ce chapitre. Cette partie débute par la description du système de monitoring qui a été mis en place dans CoCoA afin de mieux déterminer l'influence du mécanisme de sélection d'action sur le comportement des agents durant la simulation. Puis, je présenterai des simulations simples ("toys") qui ont été faites afin de montrer l'influence de chaque motivation. Une fois l'influence de chaque motivation comprise, nous passerons à des simulations prenant en compte plusieurs motivations afin d'illustrer leurs actions complémentaires. Je décrirai également des simulations illustrant une utilisation des motivations veillant à la stabilité de propriétés internes de l'agent. La dernière expérimentation se focalisera sur les prototypes et les profils d'individualité. Enfin, je présenterai l'évaluation d'un profiler sur ces simulations afin d'en extraire l'impact du mécanisme de sélection d'action sur le fonctionnement d'un agent.

5.1 L'atelier de conception de comportements

L'atelier de conception de comportements est un outil mis en place afin de concevoir des comportements indépendamment des simulations. Cet atelier a été réalisé par Jean-Baptiste Leroy et Raphaël Prud'Homme, deux étudiants de cinquième année d'IMA (Informatique Microélectronique Automatique de l'école Polytech'Lille avec la spécialité informatique) que j'ai supervisé dans le cadre de leur projet de fin d'études. Le but de cet atelier est de proposer une interface pour un utilisateur expérimenté, fournissant assez d'informations pour avoir

une vision large pendant la conception, tout en n'ayant pas de connaissances a priori sur la simulation où le comportement sera appliqué.

5.1.1 Les principes de l'atelier

Pour construire le comportement d'un agent indépendamment du contexte d'exécution, il faut définir son raisonnement et son individualité. Pour un agent CoCoA muni du mécanisme de sélection d'action basé sur les motivations cela revient à définir trois éléments :

1. Les **interactions** que l'agent peut effectuer, c'est-à-dire ses capacités. Plus un agent à de possibilités d'atteindre un état particulier de l'environnement, plus il sera capable d'exprimer des traits de caractère différents.
2. Les **propriétés** qui, comme nous l'avons vu, peuvent jouer un rôle dans le comportement de l'agent.
3. Les **motivations** qui vont influencer la partie individualité de l'agent.

Les buts de l'agent sont liés à une simulation particulière, c'est pourquoi ils ne sont pas définis lors de la construction du comportement (bien qu'ils soient pris en compte par la motivation d'influence des buts). Notre atelier de conception repose donc sur la définition des interactions que l'agent peut effectuer, de ses propriétés internes (dynamiques ou non) et des motivations qui vont influencer ses choix.

Les interactions

Les interactions que peut effectuer l'agent influencent comme nous l'avons vu, le comportement qu'il sera en mesure d'exprimer. Plus un agent a de capacités différentes, plus il sera confronté à des choix et plus il sera en mesure d'exprimer certains traits de son caractère.

Les propriétés

Dans CoCoA, chaque valeur peut être obtenue par une propriété. Il est donc possible de déterminer un but ou un paramètre d'une motivation par rapport à une propriété. Les propriétés peuvent évoluer sans exécution d'une interaction, de deux manières différentes : selon une dépendance à une autre propriété ou selon l'évolution de la simulation.

Les motivations

Les motivations influencent le choix de l'action à exécuter parmi l'ensemble des alternatives fournies par la partie raisonnement du comportement. Cette individualité de l'agent dépend des motivations et de l'expression de ces motivations (le profil d'individualité). Les motivations sont paramétrables soit par une valeur fixe, soit par la valeur d'une propriété, soit par rapport aux capacités de l'agent (les actions qu'il peut effectuer).

5.1.2 Les groupes de comportement

Regrouper les valeurs

Parmi les motivations présentées, les préférences de l'agent et les coûts des interactions (de l'accomplissement en temps) sont deux valeurs à affecter à chaque interaction pour un agent. Or, plus le nombre d'interactions que l'agent peut effectuer est important et plus l'affectation des valeurs de préférences et de coûts est une opération longue.

Évidemment, lors de la construction d'un comportement, il est possible de réutiliser une partie ou la totalité d'un comportement déjà construit afin de réduire le travail à effectuer. Pour simplifier encore le travail du concepteur, nous avons recommandé de regrouper les interactions par thématique. Nous avons mis en place ce regroupement pour les interactions afin de définir les préférences et les coûts. Ainsi, dans notre atelier, il est possible de construire des groupes de préférences et des groupes de coûts pour les interactions que l'agent peut effectuer. Pour chaque groupe, l'utilisateur peut définir une valeur. Cette valeur sera répercutée sur l'ensemble des interactions appartenant à ce groupe.

Ainsi, un utilisateur peut affecter des valeurs de préférences et de coûts des interactions soit une par une, soit en affectant une même valeur pour l'ensemble des interactions d'un groupe.

De plus, les groupes permettent d'exprimer des similitudes entre les interactions. Les groupes de préférences peuvent définir des traits de caractère et les groupes de coûts un domaine où le paramètre exprimerait une notion de qualité des compétences de l'agent dans ce domaine. Un agent qui aura le trait de caractère *bon-vivant* aura certainement des valeurs fortes pour les interactions `manger`, `boire`, `chanter` et `danser`. Un agent *bûcheron* aura un coût pour les interactions comme `scier` ou `tronçonner` inférieur à un agent *boucher*.

Gestion des conflits

Si une interaction appartient à plusieurs groupes, deux cas de figure se présentent.

1. La valeur à donner à cette interaction est la même valeur que l'une des valeurs d'un des groupes auxquels appartient l'interaction.
2. La valeur à donner à cette interaction est différente des valeurs de ces groupes.

Pour régler ces deux cas de figure, l'atelier de conception fixe une règle pour la définition des valeurs des paramètres : la valeur d'une interaction est la dernière valeur qu'elle a reçue directement ou via un de ses groupes.

Ainsi, si une interaction reçoit une valeur de 3 et que l'un de ces groupes est modifié avec la valeur 1, l'interaction aura comme valeur 1. Ceci permet à la fois de ne pas bloquer l'utilisateur en lui demandant de confirmer à chaque fois une modification car elle a un impact sur d'autres interactions, tout en fixant une règle qui gère les problèmes de conflits qui pourraient se présenter.

5.1.3 Le choix de l'utilisateur

Lors de la conception de l'atelier, nous nous sommes retrouvés face à un choix. Nous devions déterminer quel type d'utilisateur allait utiliser l'atelier de conception pour en déterminer les caractéristiques de l'interface graphique. En effet, suivant les aptitudes de l'utilisateur visé, l'interface présenterait certaines différences.

Pour un utilisateur novice, l'interface doit être intuitive, compréhensible et surtout simple, quitte à réduire les possibilités qu'offre l'interface par rapport au modèle. Nous avons également envisagé, pour un utilisateur novice, d'avoir une conception très guidée, présentant différentes étapes de conception afin de limiter les erreurs venant de l'utilisateur.

Pour un utilisateur expérimenté, l'interface doit présenter le plus possible, les richesses qu'offre le modèle. Notamment sur la possibilité d'utiliser des paramètres dynamiques comme paramètre des motivations. De plus, pour ce type d'utilisateur, il me semblait évident qu'il fallait éviter l'aspect séquentiel et dirigiste qu'on pouvait trouver dans une interface destinée à un utilisateur novice. En effet, si l'utilisateur est expérimenté, il doit pouvoir travailler sur les différentes parties du comportement, sans devoir respecter un ordre prédéfini.

L'atelier de conception que nous avons conçu est à destination d'un utilisateur expérimenté. Ainsi l'interface permet d'exploiter en profondeur notre modèle. Dans cette interface, nous nous sommes efforcés de fournir assez d'informations à l'utilisateur pour déterminer les conséquences des différents paramétrages effectués (tout en n'ayant aucune connaissance sur le contexte d'application du comportement). Enfin, l'atelier offre une interface paramétrable et permet une conception modulaire des comportements.

5.1.4 Vue générale

L'atelier se présente dans une fenêtre composée de trois parties :
- le menu permettant de charger et de sauvegarder le comportement
- l'arbre récapitulatif du comportement
- la fenêtre de gestion qui offre trois vues : les propriétés, les interactions et les motivations

Le menu et l'arbre récapitulatif du comportement sont toujours visibles quelque soit la partie du comportement qui est gérée. Les fenêtres de gestion des propriétés, interactions et motivations correspondent aux trois parties que l'utilisateur doit définir. Ces fenêtres de gestion sont affichables une à une (dans le même espace) par simple clic sur l'onglet correspondant (voir la figure 5.1).

5.1.5 Bilan

L'atelier de conception mis en place, a été conçu pour définir les comportements des agents indépendamment d'une simulation ou de CoCoA. Cet atelier permet de construire des propriétés dynamiques et dépendantes, de définir les capacités de l'agent, ainsi que les préférences et les coûts de ses interactions. Il permet également de choisir les motivations qui vont influencer le mécanisme de sélection d'action, ainsi que de concevoir le profil d'individualité de l'agent. Enfin, cet atelier de conception génère différents fichiers correspondant aux différentes parties de la définition du comportement. Les données de ces fichiers sont chargeables par l'atelier (et donc réutilisable) ce qui rend la conception modulaire. Il est ainsi possible de ne construire

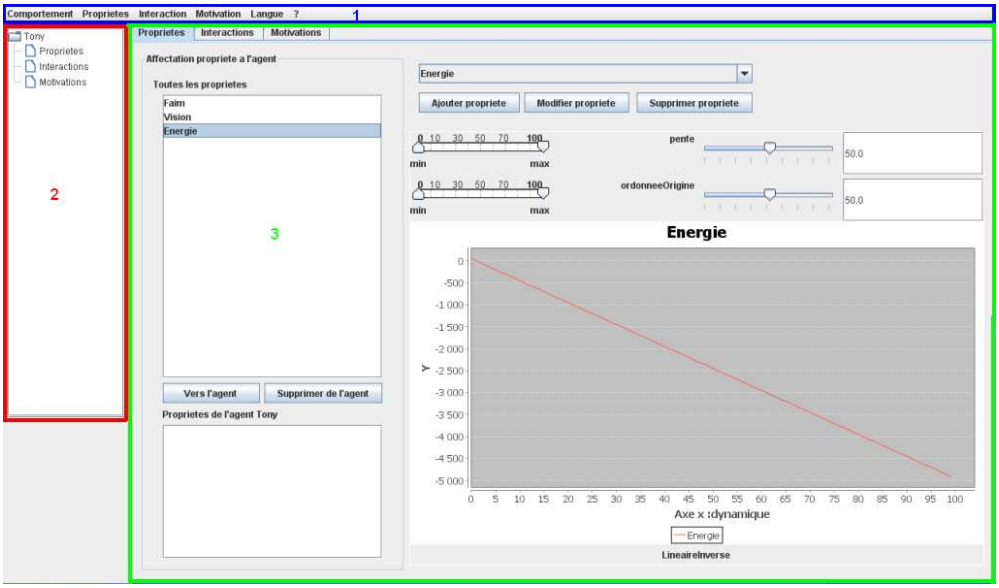


FIG. 5.1 – L'interface se décompose en trois parties. Le menu (partie 1 en bleu), l'arbre récapitulatif (partie 2 en rouge) et la fenêtre de gestion (partie 3 en vert) qui gère soit les propriétés, soit les interactions, soit les motivations. Les parties 1 et 2 sont fixes, la partie 3 dépend de l'onglet sélectionné.

qu'une partie du comportement ou même simplement de définir les groupes d'interactions qui vont simplifier une future conception. L'utilisation de l'atelier est présenté en détails en annexe C.

5.2 Les simulations

5.2.1 Le monitoring dans CoCoA

Afin de comprendre le comportement de nos agents, nous avons mis en place une interface graphique permettant de suivre l'évolution des notations des alternatives dans CoCoA. En effet, à partir du moment où l'on souhaite étudier plusieurs alternatives, il devient difficile de simplement suivre la simulation (observer le comportement) pour comprendre le comportement. C'est pourquoi nous avons mis en place un système de "monitoring" nous permettant de connaître pour chaque agent et à chaque étape de la simulation, l'évaluation de chaque interaction exécutable et la priorité de chaque but.

Le monitoring se présente sous la forme d'un graphe pour chaque motivation et d'un graphe pour l'ensemble des évaluations combinées (voir figure 5.2). Généralement, l'évaluation globale des interactions et la valeur des buts suffisent à expliquer le comportement de l'agent. Le principe de ce monitoring est de suivre l'évolution des évaluations et des interactions exécutables. Néanmoins, une interaction exécutable appartient à une ou plusieurs alternatives. Il me semblait donc judicieux de pouvoir suivre également l'évolution des alternatives tout au long de la simulation. Pour reconstruire les alternatives graphiquement, j'utilise l'algorithme

de recherche d'inertie (présenté dans la figure 4.8) dans une version simplifiée[13]. Si deux actions appartiennent à deux alternatives similaires, on considère que ces actions appartiennent à la même alternative. Pour différencier les actions d'une alternative, une génération aléatoire de couleur a été mise en place (en évitant certaines couleurs comme la couleur de fond). Chaque nouvelle action d'une alternative reçoit une couleur différente de la précédente action de la même alternative. De plus, comme nous n'avons aucune information a priori sur les valeurs des évaluations, l'échelle du graphique s'adapte automatiquement à la valeur la plus importante. Cette adaptation de l'échelle est également appliquée pour la durée de la simulation qui n'est pas limitée.

5.2.2 Les simulations "Toys" : les motivations

Pour chaque motivation, nous avons créé deux projets (ou simulations) afin de voir l'influence de la motivation sur le comportement. Ces projets sont composés du même environnement, les agents sont placés au même endroit. L'agent étudié possède les mêmes capacités dans les deux projets, seules les motivations changent. Généralement, le premier projet montre la simulation sans l'influence de la motivation, comme une simulation étalon. Le second projet montre l'influence de la motivation et l'impact de la motivation sur le comportement de l'agent.

Nous allons présenter ici des exemples simples, exécutés avec CoCoA (voir l'interface de simulation en figure 5.3), permettant d'illustrer tour à tour un certain nombre de motivations.

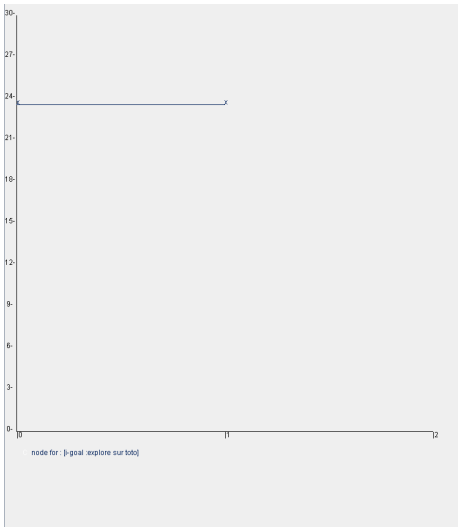
D'une expérience à l'autre nous n'utiliserons que l'une des motivations, celle que nous voulons mettre en évidence.

Priorité des buts et opportunisme

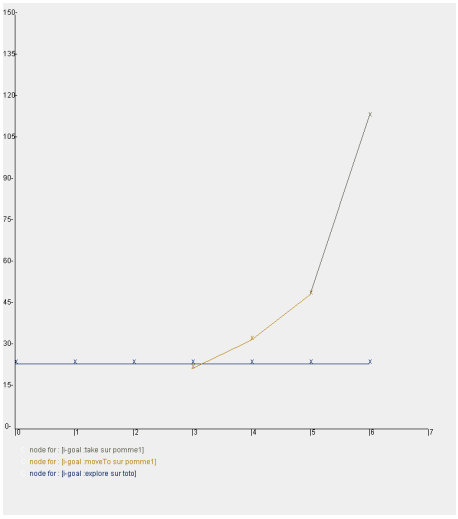
Nous commençons par illustrer la motivation qui paraît certainement la plus naturelle : la priorité des buts. Pour des raisons qui apparaîtront immédiatement évidentes, nous couplons cette présentation avec celle de la motivation d'opportunisme. Dans les deux expériences ci-dessous seules ces deux motivations sont activées.

Dans cette expérience notre agent doit récupérer quatre objets : la pomme 1, la pomme 2, la clef et la hache, ayant chacun leur propre priorité de but : *prendre on pomme1* $= 10$, *prendre pomme2* $= 9$, *prendre clef* $= 8$ et *prendre hache* $= 5$. La hache n'apparaît pas dans le champ de vision de l'agent au début de l'expérience, il doit donc explorer l'environnement pour la trouver. L'image de gauche de la figure 5.4 présente comment le plan est exécuté par le personnage si l'on ne prend en compte que la priorité des buts. On constate que les différents buts sont exécutés, et donc sélectionnés, dans l'ordre décroissant de leur importance pour l'agent. Cependant, la seule prise en compte de la priorité des buts se révèle clairement beaucoup trop naïve dans notre contexte situé dans lequel les déplacements doivent être pris en

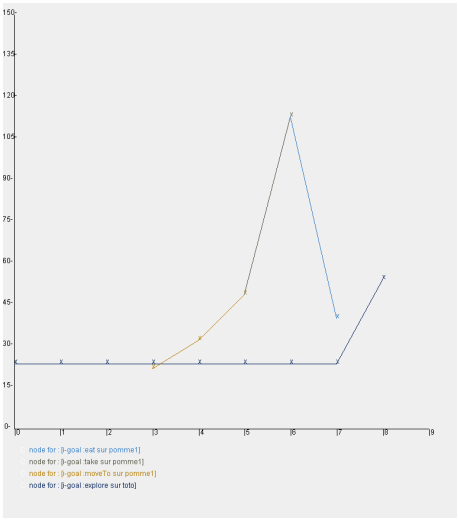
[13]Le calcul de similitude utilisé pour l'inertie sauvegarde la précédente alternative sélectionnée. Dans le cadre du monitoring, il aurait donc fallu garder en mémoire toutes les alternatives de la sélection précédente. Dans sa version simplifiée, le monitoring ne sauvegarde que les interactions exécutables du tour précédent. Le calcul de similitude est donc moins performant (notamment lors de l'ajout d'informations dans l'arbre de planification), mais le monitoring n'est qu'un indicateur, il n'a aucun impact sur le comportement de l'agent, cette simplification est donc acceptable



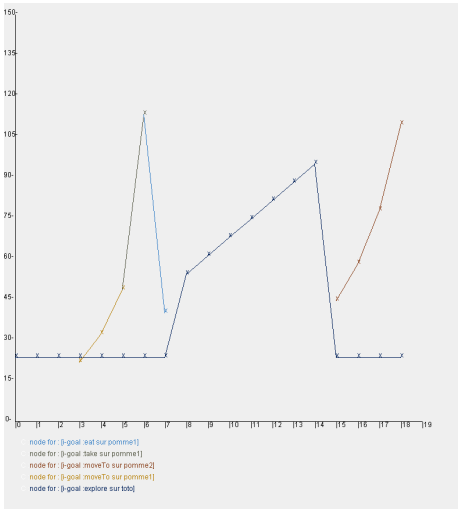
(a) Au départ une seule interaction à exécuter, l'agent "toto" doit explorer son environnement. Afin de faciliter la compréhension de l'exemple nous avons volontairement évalué l'action bien qu'elle soit seule (normalement, lorsqu'il n'y a qu'une seule action à choisir, le mécanisme de sélection d'action n'intervient pas, car il n'y a pas de choix à faire).



(b) La taille de la fenêtre de monitoring est fixe, les courbes sont adaptées pour tenir dans cette fenêtre par un facteur de réduction (facteur commun à toutes les courbes pour préserver la compréhension du graphique).



(c) En dessous du graphe, se trouve la légende présentant pour chaque couleur l'interaction correspondante. Dans cet exemple, nous avons deux alternatives dont l'une d'elles a changé d'action exécutable deux fois (sa courbe est donc composée de trois couleurs différentes).



(d) Au pas 15 de cette simulation, l'agent a de nouveau faim, le but se réactive mais il s'agit d'une nouvelle alternative, cette fois l'interaction s'effectue sur l'agent pomme2.

FIG. 5.2 – Graphes de monitoring qui présente en abscisse le repère de temps de la simulation, et les valeurs d'évaluations de l'ensemble des motivations pour les actions exécutables en ordonnée. Ce système a été mis en place pour suivre l'évolution des évaluations des alternatives. Chaque couleur représente une interaction exécutable et une courbe représente une alternative. Dans cet exemple, l'agent "toto" va explorer l'environnement pour trouver, prendre et manger des pommes dès qu'il a faim.

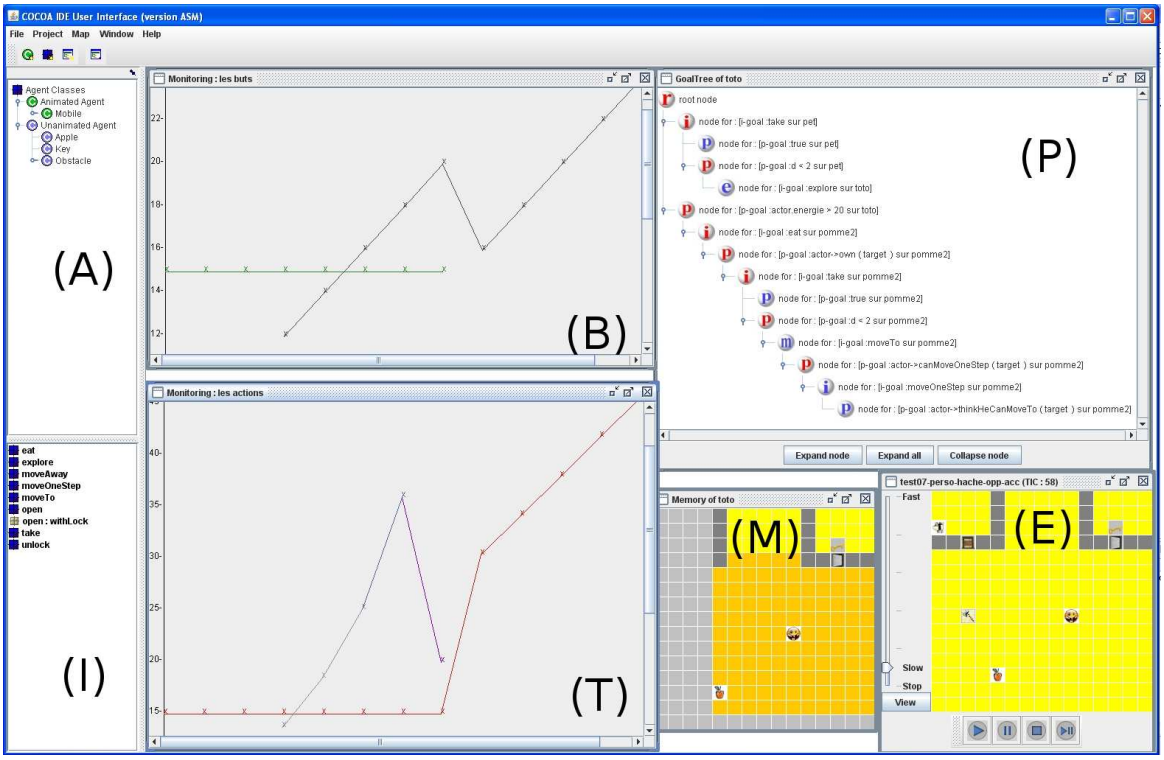


FIG. 5.3 – L'interface de simulation de CoCoA permet de définir des interactions (I), des agents (A), des environnements (E) et de construire des simulations à l'aide de ces éléments. On peut alors observer l'environnement (E), la mémoire (M), le plan (P) et les évaluations des actions (T) et des buts (B) de chaque agent.

compte. Dans le cadre de simulation de comportements que l'on veut crédibles, la réalisation des déplacements est sans conteste un élément important de perception de cette crédibilité. C'est le cas en particulier pour les personnages de jeux vidéo. La séquence de déplacements générée ici peut amener l'observateur à désapprouver l'enchaînement des actions exécutées malgré la prise en compte des importances relatives des buts. C'est cet effet que la motivation d'opportunisme doit atténuer en favorisant les actions dont la cible est proche.

C'est ce que l'on observe dans l'image de droite de la figure 5.4. Il s'agit du même contexte mais nous avons ajouté à notre ASM la prise en compte de la motivation d'opportunisme. Par l'influence de l'opportunisme, l'exécution du plan (voir figure 5.6) change. Bien que le but *prendre pomme1* soit le plus important, la proximité de la *pomme2* (au pas de temps 0) amène l'agent à choisir ce but en premier. Ce choix est expliqué par la figure 5.5 qui présente les valeurs attribuées par l'ASM aux actions exécutables à chaque pas de temps. L'action *se déplacer vers pomme2* a une évaluation supérieure à l'action *se déplacer vers pomme1* grâce à l'opportunisme. Ensuite, alors qu'il se dirige vers la *pomme1*, l'agent est à nouveau temporairement amené à se détourner de son but principal (au pas 5 de la simulation) pour prendre la *clef* dont il passe à proximité (voir la figure 5.5 où la courbe de l'action *se déplacer vers clef* "passe au-dessus" de la courbe *se déplacer vers pomme1*). Le compromis entre priorité des buts et opportunisme a ainsi amené un enchaînement des actions plus "raisonnables".

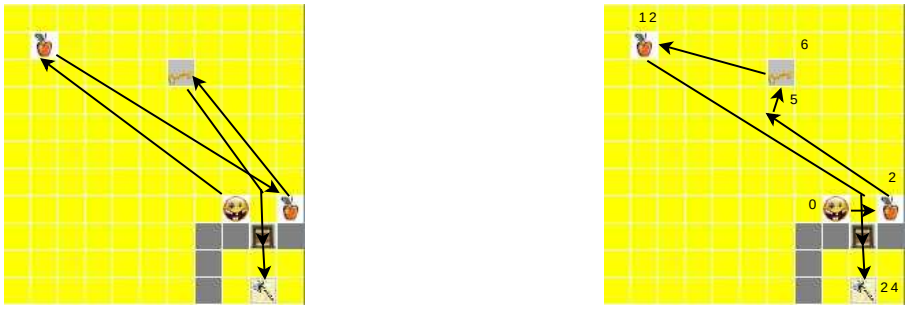


FIG. 5.4 – L'agent a 4 buts, ayant chacun leur propre priorité : *prendre pomme1* = 10, *prendre pomme2* = 9, *prendre clef* = 8 et *prendre hache* = 5. La *pomme1* est celle en haut à gauche. La version de droite possède les annotations de pas de temps que l'on retrouve dans la figure 5.5. À gauche, l'ASM prend en compte uniquement la priorité des buts, à droite, il prend en compte la priorité des buts et la motivation de l'opportunisme.

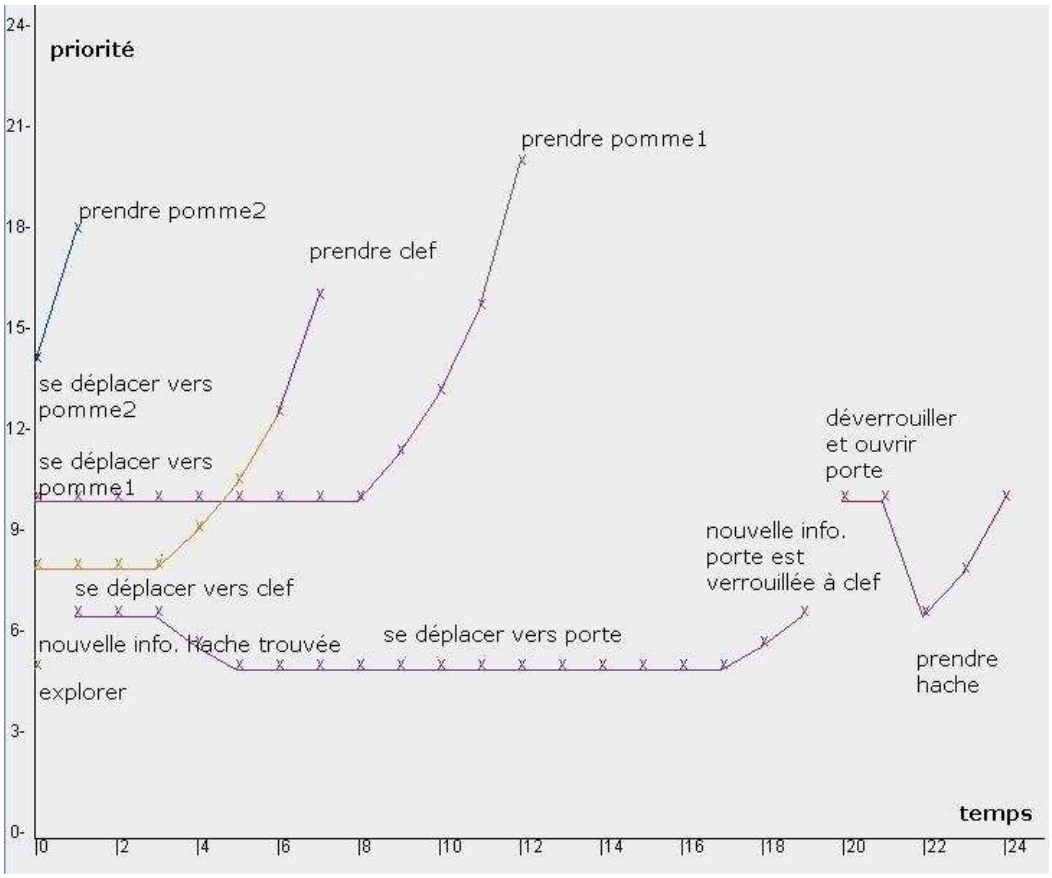


FIG. 5.5 – Valeurs des actions exécutables calculées par l'ASM en fonction du temps, correspondant au déroulement de l'expérience de droite de la figure 5.4. Les valeurs prennent en compte la priorité du but associée et la valeur de l'opportunisme. Le pas de temps 0 correspond au temps avant de faire la première action. Les courbes correspondent aux évaluations accordées aux actions successives d'une même alternative.

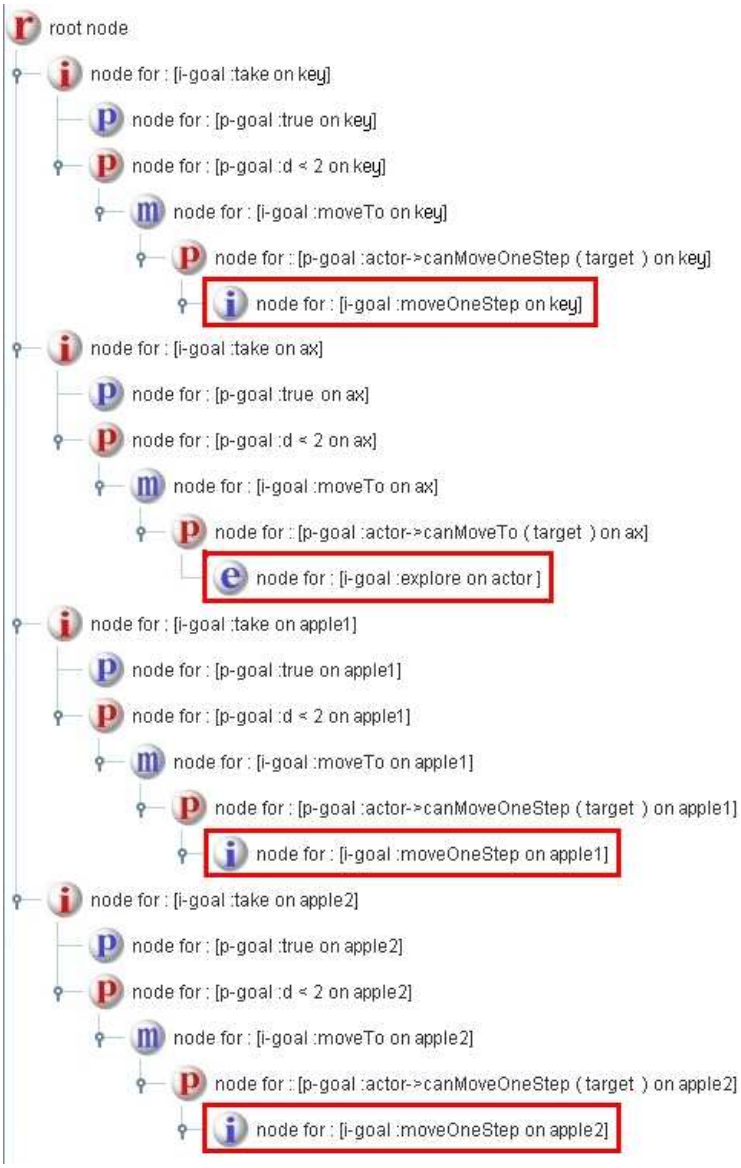


FIG. 5.6 – Plan construit par le raisonnement, à l'aide d'un arbre et-ou, au début des deux simulations de la figure 5.4. Les actions exécutables sont encadrées en rouge. L'action *explore* s'explique par le fait qu'initialement l'agent ne sait pas où se trouve la hache.

Notons que la priorité du but *prendre hache* n'était pas assez importante pour que, même renforcé par l'opportunisme, cette action soit sélectionnée en début de simulation.

Signalons rapidement que le genre d'effet indésirable obtenu sans la prise en compte de l'opportunisme, est bien connu dans les jeux vidéo. Dans le fameux jeu *les Sims*, un personnage devant prendre le courrier dans la boîte aux lettres pour le déposer sur une table dans la maison et ramasser le journal au pied la boîte aux lettres pour le déposer à son tour sur la table fera des allers-retours de la boîte à la table plutôt que de ramasser le journal immédiatement lorsqu'il en est proche.

Motivation des préférences

Pour illustrer l'influence exercée par la motivation des préférences, nous allons présenter deux expériences dans lesquelles nous ne changerons que les préférences attribuées aux actions *déverrouiller* et *casser*. Celles-ci sont présentées à la figure 5.7. A nouveau, un agent est situé dans un environnement (voir figure 5.7) dans lequel se trouve une porte, une pomme, une clef et une hache. Le but de l'agent est d'atteindre la pomme (pour la manger). La porte est fermée et verrouillée et il existe deux moyens de la franchir, soit en la déverrouillant avec la clef (puis en l'ouvrant) soit en la cassant avec la hache.

Dans la simulation de gauche, les valeurs de préférences attribuées aux capacités de l'agent sont : $\pi(\textit{déverrouiller}) = 2$, $\pi(\textit{casser}) = 0.8$ et $\pi = 1.1$ pour les autres. Dans la simulation présentée dans l'image de droite, seules les préférences de *déverrouiller* et *casser* sont échangées : $\pi(\textit{déverrouiller}) = 0,8$, $\pi(\textit{casser}) = 2$ et $\pi = 1,1$ pour les autres. On peut constater qu'à gauche l'agent privilégie l'action prendre la clef pour ensuite déverrouiller la porte alors qu'à droite il privilégie l'action prendre la hache pour casser la porte.

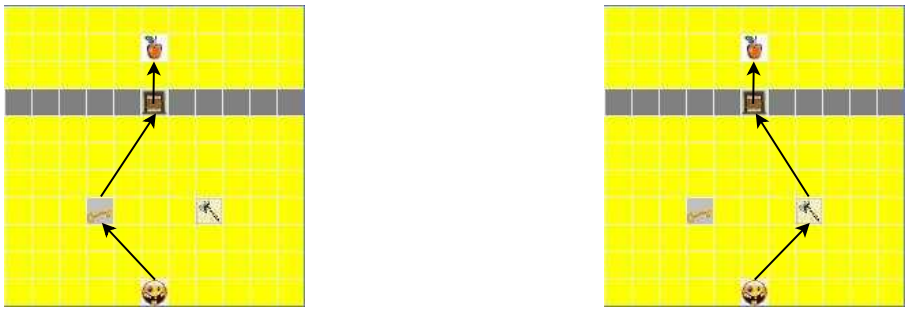


FIG. 5.7 – Selon les préférences accordées à ses capacités, le personnage choisit de *déverrouiller* (à gauche) ou de *casser* (à droite) la porte.

Expliquons rapidement ce qui justifie ces différences de sélection d'action. Le but de l'agent est *prendre(pomme)*, en fonction des capacités de l'agent, le chaînage arrière du planificateur produit pour ce but deux alternatives concurrentes, calculées à partir de l'arbre est/ou produit :

1. $\textit{prendre(pomme)} \longrightarrow \textit{ouvrir(porte)} \longrightarrow \textit{déverrouiller(porte)} \longrightarrow \textit{prendre(clef)}$,
2. $\textit{prendre(pomme)} \longrightarrow \textit{casser(porte)} \longrightarrow \textit{prendre(hache)}$.

La construction des plans prend en compte les déplacements en gérant les sous-plans qui en découlent et donc calcule que le franchissement de la porte est nécessaire à la résolution des buts. L'agent a deux manières de franchir cet obstacle : en le déverrouillant ou en le cassant (ce qui le supprime). Pour chacune la valeur de γ_{pref} s'exprime ainsi :

1. $\gamma_{pref}(prendre(clef)) = Moy_{harm}(\pi(prendre)\,;\pi(ouvrir)\,;\pi(\textit{déverrouiller})\,;\pi(prendre))$,
2. $\gamma_{pref}(prendre(hache)) = Moy_{harm}(\pi(prendre)\,;\pi(casser)\,;\pi(prendre))$.

Pour chacune des simulations on a donc :

à gauche $\gamma_{pref}(prendre(clef)) = Moy_{harm}(1,1\,;1,1\,;2\,;1,1) = 1,24$ et $\gamma_{pref}(prendre(hache)) = Moy_{harm}(1,1\,;0,8\,;1,1) = 0,98$, l'action retenue est donc *prendre(clef)*,

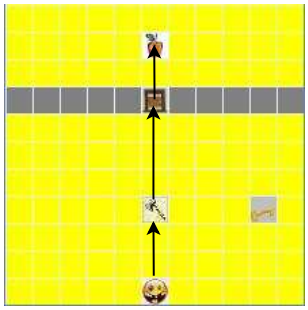


FIG. 5.8 – Influence de la motivation d'accomplissement en espace : l'alternative avec le trajet le plus court est privilégiée.

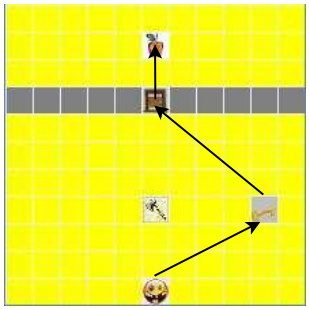


FIG. 5.9 – Influence de la motivation d'accomplissement en temps : l'alternative de moindre coût est privilégiée.

à droite $\gamma_{pref}(prendre(clef)) = Moy_{harm}(1,1\,;1,1\,;0,8\,;1,1) = 1,01$ et $\gamma_{pref}(prendre(hache)) = Moy_{harm}(1,1\,;2\,;1,1) = 1,29$, l'action retenue est donc *prendre(hache)*.

Dans les deux simulations, l'environnement, les capacités et les connaissances (l'état de sa mémoire) de l'agent étant rigoureusement les mêmes, les plans construits, et donc les alternatives proposées par le planificateur, sont identiques. La différence de déroulement des simulations est uniquement due à des choix différents réalisés par le mécanisme de sélection d'action. Dans ce cas, on peut percevoir l'influence de la motivation des préférences. Un observateur extérieur pourrait considérer l'agent de la simulation de droite comme "plus brutal" ou "moins discret" que celui de la simulation de gauche. Ainsi, en jouant sur les valeurs attribuées aux préférences des différentes capacités (c'est-à-dire son profil comportemental pour cette motivation) d'un agent, on influe sur la perception que l'observateur a de sa personnalité.

L'accomplissement en espace

Cette fois, seule la motivation appelée *accomplissement en espace* est utilisée dans un contexte de simulation similaire à celui utilisé dans la simulation sur les préférences de l'agent. Cette motivation prend en compte les déplacements nécessaires à la réalisation d'une alternative. La simulation de gauche de la figure 5.8 illustre ce point : toutes les autres motivations sont écartées (notamment l'opportunisme), on constate que l'agent privilégie l'alternative nécessitant le moins de déplacement et donc ici l'action *prendre(hache)*.

L'accomplissement en temps

La motivation de l'accomplissement en temps a pour objectif de favoriser les alternatives dont les actions prennent le moins de temps (cumulé). Elle prend donc en compte les actions nécessaires à la réalisation d'une alternative. Nous expérimentons cette motivation dans le même contexte que l'accomplissement en espace, mais seule cette motivation influence l'ASM (voir la figure 5.9 à droite). Nous avons, arbitrairement, attribué un coût de 5 à l'action *casser* et de 1 à toutes les autres actions. Le seuil θ_{accT} est fixé à 5. Si l'on reprend les alternatives présentées dans la simulation sur les préférences de l'agent, nous obtenons :

1. $cost(resolution(prendre(clef)) = 1 + 1 + 1 + 1 = 4$ et donc $accT(prendre(clef)) = 1 + \log_5(\frac{5}{4}) = 1,14$
2. $cost(resolution(prendre(hache)) = 1 + 5 + 1 = 7$ et donc $accT(prendre(hache)) = 1 + \log_5(\frac{5}{7}) = 0,79$

L'action *prendre(clef)* est favorisée alors que *prendre(hache)* est pénalisée ce qui justifie le choix de prendre la clef retenu par l'ASM (à nouveau les autres motivations ont été désactivées, en particulier l'opportunisme et l'accomplissement en espace).

Nous avons illustré l'influence exercée par la plupart des motivations que nous avons proposées. Dans ces expériences, le paramétrage est présenté de façon à illustrer l'influence des motivations sur la sélection d'action. Cependant il convient d'insister sur le fait que notre proposition n'a pas pour but de paramétrer les motivations afin de réaliser un scénario prédéfini, mais de permettre l'expression d'une individualité pour chaque agent dans tous les contextes.

5.2.3 La combinaison des motivations

Dans cette simulation, je vais présenter comment plusieurs motivations influencent le comportement de l'agent. Pour cela, je considère un agent-PNJ c caractérisé par un champ de vision et un attribut représentant son *énergie*. Son énergie est une propriété dont la valeur décroît à chaque pas de temps. Les capacités de c sont de pouvoir *se déplacer*, *prendre* les objets, *manger*, *casser*, *déverrouiller*, *ouvrir* et *explorer*[14] l'environnement (voir tableau 5.1).

Pour définir la personnalité de l'agent, j'ai attribué arbitrairement pour chaque capacité une préférence afin de modéliser un agent qui manifeste un caractère brutal. Pour cela, j'ai donc affecté une valeur de $\pi(casser) = 1,5$ à l'action *casser* et pris $\pi(déverrouiller) = 0,5$ (répulsion), les autres actions reçoivent une préférence neutre (ie. $\pi = 1$)[15]. Ces choix devraient mener c à préférer casser les portes plutôt que de les déverrouiller.

Une fois l'ASM construit, c est situé dans l'environnement, présenté à la figure 5.10. Les buts sont alors donnés à c. Dans notre cas, son premier but g_1 est de *prendre l'objet o* dans la pièce en haut à gauche fermée par une porte vitrée dont la clef est dans la pièce en haut à droite. Son deuxième but, g_2 est de maintenir son niveau d'*énergie au dessus d'une certaine valeur v*.

g_1 influence le comportement de l'agent par une fonction constante. L'influence de g_2 est fonction du niveau d'énergie de l'agent. Deux pommes et une hache sont présentes dans l'environnement, l'action de manger une pomme redonne de l'énergie à l'agent et la hache peut être utilisée pour casser les portes.

c n'a pas de connaissance *a priori* sur son environnement et il doit l'explorer. Les lieux inconnus apparaissent en noir dans la figure 5.10 où l'on peut également voir la portée de la vision de l'agent. A chaque pas, c exécute l'action sélectionnée par l'ASM dans le but de résoudre ses buts. Le trajet de l'agent est montré par des pointillés noirs.

[14] Comme nous l'avons présenté précédemment dans la section 2.1.4, *explorer* est l'interaction qui permet la recherche d'une cible inconnue.

[15] La valeur $\pi(explorer) = 1$ n'est utilisée ici que pour mettre en évidence une revalorisation multi-buts. Si l'agent connaît l'emplacement de la clef et pas celui de la hache, il peut sembler plus rationnel qu'il utilise la clef plutôt qu'il parte à la recherche de la hache qui pourrait ne pas exister. Pour cela je conseille d'utiliser une

open :
`condition` = `target.opened = false and target.locked = false`
`guard` = `distance(actor, target) < 1`
`action` = `target.opened = true`

eat :
`condition` = `actor.own(target)`
`guard` = –
`action` = `actor.energy.add(target.energy) and remove(target)`

take :
`condition` = –
`guard` = `distance(actor, target) < 1`
`action` = `target.inventory.add(target)`

break :
`condition` = `actor.own(axe)`
`guard` = `distance(actor, target) < 1`
`action` = `remove(target)`

unlock :
`condition` = `target.locked = true and actor.own(target.key)`
`guard` = `distance(actor, target) < 1`
`action` = `target.locked = true`

explore et *moveTo* sont des interactions de déplacement
`remove` est une primitive qui supprime de l'environnement

TAB. 5.1 – Le code des interactions utilisées dans la simulation.

Considérons le parcours de c. Au début, c est localisé au point **1**, il perçoit seulement la pomme à droite. Comme notre agent démarre par un niveau d'énergie supérieur à v, le seul but actif est g_1(ie. g_2 est satisfait, sa priorité est de $-\infty$). Comme c ne connaît pas son environnement, il doit l'explorer afin de trouver o. La figure 5.11 montre les différentes valeurs obtenues par les actions exécutables suite à l'évaluation de notre ASM. Au début, la seule action exécutable est *explorer*, de ce fait elle est sélectionnée. Pendant son exploration, c perd de l'énergie. En atteignant le point **2**, son énergie tombe en dessous de la valeur v, le but g_2 n'est plus satisfait, l'influence des buts note donc l'action résolvant g_2 selon la valeur actuelle de v. Cela implique que l'action de *se déplacer vers la pomme* (pour la manger) devient une action exécutable (*explorer* perd donc son inertie). Comme le montre la figure 5.11, la valeur de cette action est la meilleure, alors c choisit d'exécuter cette action. Comme son énergie continue à décroître pendant le déplacement, la priorité du but s'accroît et l'évaluation de l'action également. Au point **2'**, c mange la pomme, il reçoit l'énergie et le but g_2 redevient inactif. c reprend donc l'exploration afin de trouver o (*explorer* regagne donc l'inertie).

valeur inférieure aux autres actions pour $\pi(explorer)$, car l'exploration doit rester le plus souvent la dernière solution à envisager.

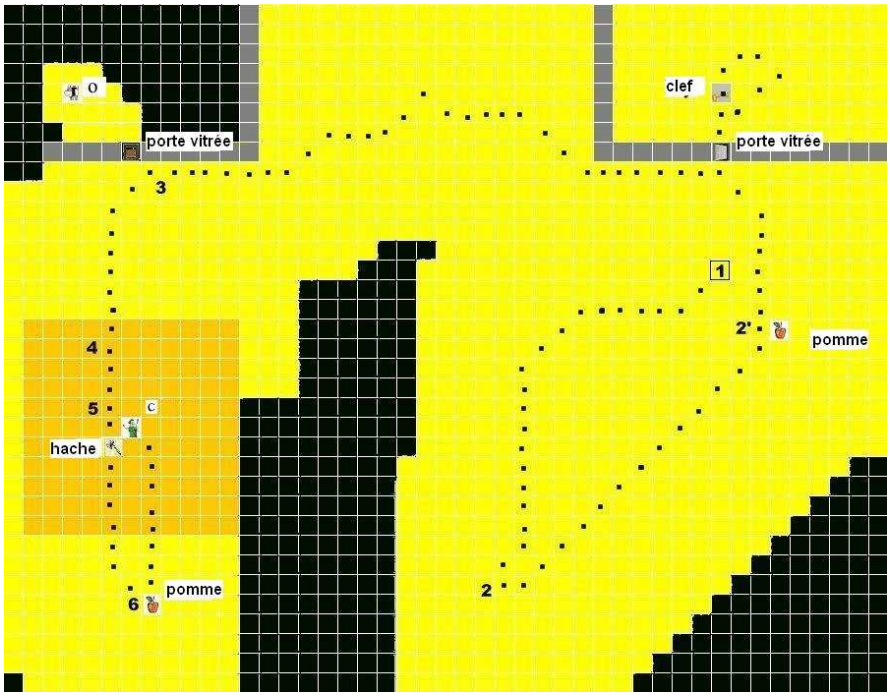


FIG. 5.10 – l'environnement et le cheminement de l'agent dans celui-ci.

Atteignant le point **3**, c perçoit o, en essayant d'*ouvrir* la porte, c acquiert l'information que celle-ci est verrouillée. Le plan propose alors deux possibilités pour traverser la porte : la *déverrouiller* ou la *casser*. Deux actions exécutables apparaissent dans le graphe correspondant aux deux alternatives : la première, *se déplacer* vers la clef (qui a été vue précédemment), la seconde *explorer* pour trouver un objet pour casser la porte. Dans la mesure où casser a été favorisée, l'agent préfère casser plutôt que de déverrouiller la porte (d'autant plus que *déverrouiller* a été pénalisée $\pi = 0,5$). C'est pourquoi l'alternative de l'action *explorer* est favorisée par rapport à l'alternative de l'action *se déplacer* qui correspondant à la plus basse courbe démarrant en **3**. L'autre courbe est particulièrement haute, car par coïncidence, au même moment, le but g_2 est redevenu actif, et l'action *explorer* est également présente pour l'alternative concernant la recherche de nourriture. De ce fait, la *revalorisation multi-buts* a valorisée l'action *explorer*, présente dans deux alternatives différentes.

En explorant, c se dirige vers "le bas" et perçoit une hache au point **4**. Alors, l'action exécutable pour *casser* n'est plus *explorer*, mais *prendre* la hache, cette dernière correspond à la nouvelle courbe au milieu. *Explorer* perd donc la faveur de l'évaluateur de *revalorisation multi-buts*, ce qui explique la chute de sa courbe. Néanmoins, elle reste l'action la plus prioritaire.

Au niveau du point **5**, par l'influence de l'*opportunisme* sur la cible hache, l'action exécutable *prendre la hache* est favorisée, devenant ainsi la plus prioritaire. Le pic au point **5** est dû à l'*opportunisme*, la baisse de l'action *explorer* au même moment est due à la perte de l'*inertie*.

Une fois la hache prise, l'influence de l'*opportunisme* disparaît et l'action *explorer* devient à nouveau la plus prioritaire. Plus tard, c trouve une pomme et la mange au point **6**. *Casser*

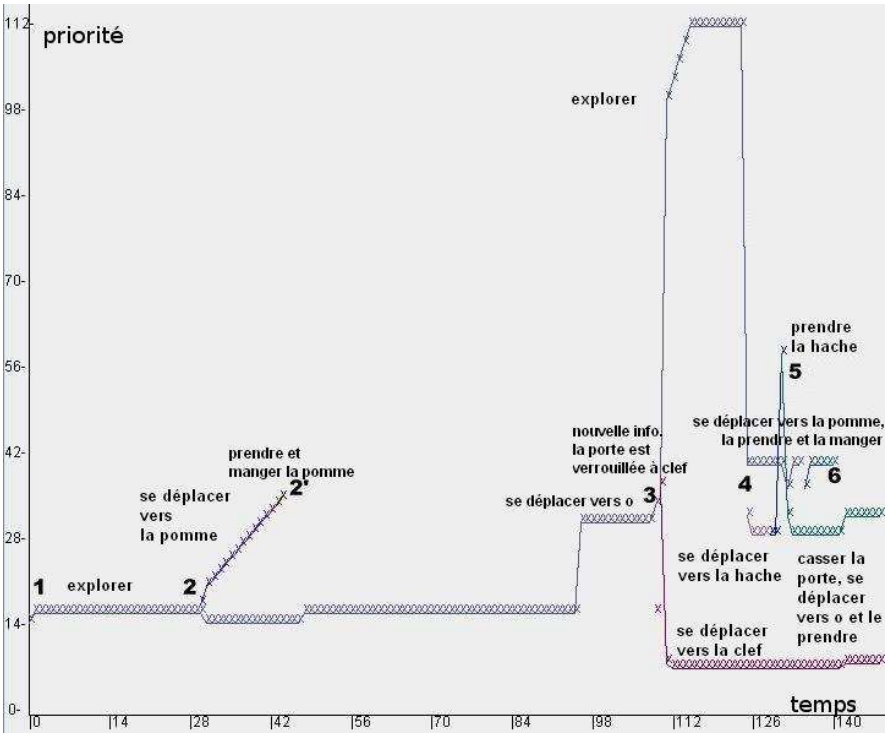


FIG. 5.11 – Courbes d'évaluation des actions exécutables par l'ASM.

la porte devient l'action sélectionnée par l'ASM. c se déplace vers la porte, la casse et prend o.

Cette expérimentation illustre le fonctionnement de l'ASM et la combinaison de différentes motivations.

5.2.4 L'influence des propriétés

Comme nous l'avons vu, je propose une utilisation des motivations qui n'est pas simplement une représentation d'une propriété interne de l'agent. Néanmoins, afin de montrer qu'il est possible d'obtenir des agents dont le comportement est dirigé par leurs propriétés internes, j'ai construit une simulation dans lequel les buts de l'agent seront de satisfaire leurs propriétés internes. Cette simulation se base sur une simulation utilisée par Etienne de Sevin dans sa thèse[dS06] dont les motivations sont des influences exprimant le besoin de satisfaire une propriété de l'agent. Notre agent aura les mêmes capacités, les mêmes propriétés internes et l'environnement sera très proche de la version de de Sevin.

L'environnement

Comme le montre la figure 5.12, les deux environnements sont très proches. On retrouve les différentes pièces et les éléments sur lesquels l'agent peut effectuer des actions. Dans le bureau (en haut à gauche) on retrouve le plan de travail (PT), la bibliothèque (Bib), une

FIG. 5.12 – *À gauche :* l'environnement utilisé par de Sevin [dS06]. *À droite* : l'environnement utilisé dans CoCoA.

place pour faire de l'exercice (EP), l'ordinateur (PC) et un téléphone (Tp). Dans la chambre (en haut au centre) se trouve le lit (Lit), la plante (Pl), l'étagère (Et) et une place pour faire de l'exercice (EP). En haut à droite se trouve la salle de bain avec une baignoire (B) et les toilettes (To) dans la pièce en dessous. À noter que la baignoire permet, comme le présente de Sevin, d'effectuer les actions se laver et nettoyer, il n'était donc pas utile de représenter une douche et un lavabo. En bas à gauche se trouve la cuisine dans laquelle se trouve un évier (Ev), un four (Fo) et une place pour faire de l'exercice (EP). En bas à droite se trouve la partie salon-salle-à-manger dans laquelle se trouve une table (T), un téléphone (Tp), une télévision (TV), un sofa (So) et une place pour faire de l'exercice (EP).

Les motivations

De Sevin propose treize buts correspondant à douze propriétés internes de l'agent (faim, soif, repos, aller aux toilettes, dormir, se laver, faire à manger, nettoyer, communiquer, faire de l'exercice, arroser et cuisiner) et d'une action par défaut (regarder la télé). Pour réaliser cette simulation, j'ai mis en place douze buts (un but par propriété sauf pour `cuisiner`, plus l'action par défaut). Chaque but exprime le fait de garder une propriété interne de l'agent au dessus d'un certain seuil. Nous verrons comment ces propriétés évoluent par la suite. En plus de la motivation de la priorité des buts, nous avons mis en place les motivations d'opportunisme (avec une valeur de seuil de 5) et de revalorisation multi-buts. Contrairement à la simulation de de Sevin, le but `cuisiner` n'a pas lieu d'être pour un agent CoCoA. En effet, l'interaction `cuisiner` est nécessaire pour pouvoir `manger`, l'agent doit d'abord cuisiner pour produire un repas qu'il pourra ensuite manger. Or l'enchainement entre les interactions `cuisiner` et `manger` est calculé par la partie raisonnement du comportement de l'agent CoCoA. Pour simplifier la simulation, j'ai défini l'effet de l'interaction `cuisiner` comme l'incrémentation d'une propriété (nommée *food*) correspondant à une quantité de repas qui doit être supérieure à 0 pour pouvoir exécuter l'interaction `manger`. Cette propriété n'est pas évolutive, seules les interactions `cuisiner` et `manger` en modifient la valeur (voir la figure 5.13).

communiquer :
condition = true
guard = distance(actor, target) < 1
action = add(actor.communicating, 1)

regarderTV :
condition = true
guard = distance(actor, target) < 2
action = add(actor.watching, 1) and add(actor.dowatching, 1)

manger :
condition = actor.food > 0
guard = distance(actor, target) < 1
action = add(actor.hunger, 1) and substract(actor.food, 1)

cuisiner :
condition = true
guard = distance(actor, target) < 1
action = add(actor.food, 1)

FIG. 5.13 – Exemples des interactions `communiquer`, `regarderTV`, `cuisiner` et `manger`. La propriété *dowatching* de l'agent permet de maintenir actif le but de regarder la télé, qui est un but par défaut. Ainsi, l'agent ne doit pas maintenir une propriété pour regarder la télé. En effet, l'agent n'a pas "besoin" de regarder la télé, il le fait par défaut. La propriété $food$ de l'agent permet de représenter une quantité de repas cuisinés. Ainsi la partie raisonnement de CoCoA peut déduire qu'il faut `cuisiner` pour pouvoir `manger`.

Les interactions

De Sevin propose un ensemble de douze buts pouvant être résolus par vingt-cinq actions. De par l'aspect générique de l'approche centrée interaction, une seule interaction peut être utilisée sur plusieurs cibles différentes. Par exemple, dans notre cas il n'y a qu'une interaction `lire` que la cible soit l'ordinateur ou la bibliothèque. Pour De Sevin, il existe une action `read` qui s'effectue sur la bibliothèque et une action `read1` qui s'effectue sur l'ordinateur. Nous n'utilisons donc que douze interactions (voir le tableau 5.2) dont l'effet est d'incrémenter de 1 la valeur de la propriété interne de l'agent correspondant (voir la figure 5.13).

La simulation en deux parties

Le but de cette simulation est de pouvoir construire un agent qui effectue des interactions pour maintenir ses propriétés internes au dessus d'un certain seuil et qui par défaut va regarder la télévision. Bien que la notion de pas de temps existe pour suivre plus facilement le déroulement de la simulation, la notion de jour ou d'heure n'existe pas dans une simulation CoCoA. J'ai donc effectué deux simulations dans le même environnement, avec les mêmes interactions et les mêmes agents, en changeant l'évolution des propriétés de l'agent afin de montrer deux manières de simuler l'évolution des propriétés dans le temps. Dans la première simulation, les interactions peuvent avoir des durées différentes. Dans la seconde simulation, les interactions ont une durée identique, l'agent devra donc effectuer plusieurs fois une même interaction pour avoir le même effet que la première simulation.

propriétés internes	interactions bénéfiques	cibles possibles	identifiant
faim	manger	table	T
		evier	Ev
soif	boire	table	T
		evier	Ev
regarder la télé	regarder	télé	TV
lecture	lire	bibliothèque	Bib
		ordinateur	PC
nettoyage	nettoyer	plan de travail	PT
		étagère	Et
		baignoire	B
		table	T
exercice	faire de l'exercice	place1	EP
		place2	EP
		place3	EP
		place4	EP
communication	communiquer	ordinateur	PC
		téléphone1	Tp
		téléphone2	Tp
repos	se reposer	lit	Lit
		canapé	So
sommeil	dormir	lit	Lit
arrosage	arroser	plante	Pl
hygiène	se laver	baignoire	B
satisfaction	aller aux toilettes	toilettes	To
	cuisiner	évier	Ev
		four	Fo

TAB. 5.2 – Les douze propriétés internes, les treize interactions utilisées, les agents cibles possibles et les identifiants dans la figure 5.12. Comme dans la thèse de de Sevin, `se reposer` et `dormir` sont deux interactions qui ne sont pas liées à la même propriété interne de l'agent.

Pour ces deux simulations, un but s'active (est à satisfaire) lorsque la valeur de la propriété correspondante est inférieure de 0.0. Les propriétés sont évolutives et la fonction d'évolution associé est décroissante afin de représenter la perte de la propriété qu'il faudra combler par l'exécution d'une interaction. La pente de cette fonction décroissante a été calculée à partir de la fréquence à laquelle l'agent doit effectuer l'interaction (dont l'effet est d'incrémenter la valeur de la propriété associée). De plus, les valeurs des priorités des buts sont liées aux propriétés correspondantes. Dans le cas général, la valeur d'un but est égale à l'opposé de la valeur de la propriété correspondante. Par exemple, le but `actor.sleep < 0,0` a une priorité de $5,0$ lorsque la propriété *sleep* de l'agent vaut $-5,0$. Dans le cas où la valeur de la propriété est supérieure ou égale à $0,0$, le but est satisfait, la partie raisonnement ne calcule pas les alternatives correspondantes.

Le seul but qui ne répond pas à cette description est le but qui incite l'agent à regarder la télé. Ce but a une priorité constante qui vaut 0 (une inhibition). Pour rappel, une inhibition est une répulsion extrême, l'agent exécute une interaction dont l'évaluation est une inhibition

dans le cas où cette interaction est la seule que l'agent puisse effectuer (notamment lorsqu'il n'y a pas d'autres buts à satisfaire). C'est pourquoi le but par défaut a été construit avec une priorité inhibitrice et une condition de satisfaction qui n'est pas atteignable. Pour cela, j'ai utilisé un but du type prémisse :

```
type      : premissGoal
condition : actor.watching > actor.dowatching
priority  : constantValue : 0
```

Ainsi tant que la propriété *dowatching* est supérieure à la propriété *watching*, ce but est à satisfaire. L'interaction `regarderTV` a été construite en conséquence (voir la figure 5.13).

La première simulation

Pour la première simulation, l'objectif est d'obtenir des exécutions d'interaction de durées différentes. En effet, à chaque pas de temps, l'agent effectue une interaction ou un déplacement, il m'était donc impossible d'attribuer le même nombre de pas pour un déplacement, pour l'interaction `manger` et pour l'interaction `dormir`. Pour cela, il fallait au préalable déterminer la valeur d'un pas de simulation en temps. J'ai fixé arbitrairement 1 heure à 100 pas de simulation. Ainsi une journée est représentée par 2400 pas de simulation et une semaine par 16800 pas. À partir de cette échelle de conversion j'ai pu déterminer le temps que devrait prendre chaque interaction en nombre de pas de simulation. Ensuite à partir de la durée d'une interaction et de la fréquence à laquelle cette interaction est habituellement effectuée, j'ai pu déterminer la pente de la fonction d'évolution associée à la propriété que je cherche à construire (voir la figure 5.3). Enfin, comme chaque interaction prend un pas de temps pour être exécutée, j'ajoute dans la partie `action` des interactions, l'appel à la primitive `wait` en fournissant en paramètre la durée en nombre de pas. La primitive `wait` bloque l'agent pendant un nombre de pas de simulation équivalant à la valeur de son paramètre.

Une semaine est constituée de 168 heures. En prenant en compte les fréquences et les durées des interactions, nous pouvons en déduire que dans une semaine il est possible d'exécuter 100 interactions (si on ne compte pas les interactions de déplacements, ni l'interaction `cuisiner`). Parmi ses interactions, en une semaine simulée, il est possible d'exécuter 6 fois l'interaction `regarderTV`.

Les résultats (voir la figure 5.14) montrent que malgré l'influence des déplacements et de l'interaction `cuisiner` (qui prennent un pas de simulation à chaque exécution) sur l'évolution des propriétés de l'agent, il est possible d'obtenir des résultats permettant de simuler une semaine. Même si un facteur 100 a été mis en place pour simuler 1 heure, la consommation de pas de simulation, lors des déplacements, représente du temps en moins pour l'exécution de l'interaction `regarderTV` que j'estime être en moyenne de $2,498$ par semaine. En effet, les déplacements représentent $1,487\%$ des interactions effectuées[16]. Sur 168 interactions par semaine, ce pourcentage représente donc $2,498$ interactions par semaine en moyenne. En prenant en compte ce manque et en l'incluant au nombre d'exécutions de l'interaction `regarderTV` par semaine, ce dernier passe à $5,85$ interactions par semaine (soit un nombre assez proche des 6 interactions par semaine).

[16] 424952 pas de simulation et 418633 pas d'exécution qui ne sont pas des déplacements

Interaction	Durée en heures	Durée en nombre de pas	Fréquence	Pente de la fonction d'évolution
nettoyer	4	400	1 fois par semaine	$\frac{-1}{16800}$
communiquer	2	200	1 fois par jour	$\frac{-1}{2400}$
manger	1	100	3 fois par jour	$\frac{-3}{2400}$
boire	1	100	3 fois par jour	$\frac{-3}{2400}$
faire de l'exercice	3	300	1 fois par semaine	$\frac{-1}{16800}$
lire	2	200	1 fois par semaine	$\frac{-1}{16800}$
se reposer	1	100	1 fois par jour	$\frac{-1}{2400}$
aller aux toilettes	1	100	1 fois par jour	$\frac{-1}{2400}$
dormir	8	800	1 fois par jour	$\frac{-1}{2400}$
se laver	1	100	2 fois par jour	$\frac{-2}{2400}$
arroser	1	100	1 fois par semaine	$\frac{-1}{16800}$
regarderTV	1	100	–	0.0

TAB. 5.3 – La durée des interactions et le calcul de la pente pour l'évolution des propriétés correspondantes (en fonction de la fréquence). Les interactions de déplacement et l'interaction `cuisiner` ne sont pas liées directement à une propriété qui évolue.

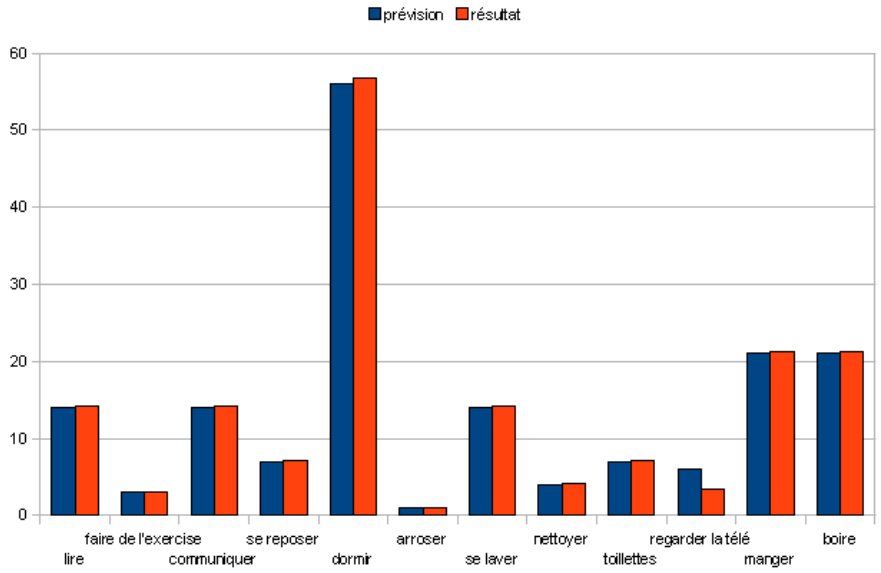


FIG. 5.14 – Exécutions moyennes des interactions pour une semaine en heure simulée (résultat et prévision) après une simulation de 25 semaines (en temps simulé sans prendre en compte les déplacements). L'exécution moyenne de l'interaction `regarderTV` est inférieure à la prévision car cette interaction est exécutée pour satisfaire un but par défaut et à cause de l'influence des déplacements sur l'évolution des propriétés internes de l'agent.

Interaction	Durée en nombre de pas	Fréquence	Pente de la fonction d'évolution
nettoyer	100	4 fois par semaine	$\frac{-4}{16800}$
communiquer	100	2 fois par jour	$\frac{-2}{2400}$
manger	100	3 fois par jour	$\frac{-3}{2400}$
boire	100	3 fois par jour	$\frac{-3}{2400}$
faire de l'exercice	100	3 fois par semaine	$\frac{-3}{16800}$
lire	100	2 fois par semaine	$\frac{-2}{16800}$
se reposer	100	1 fois par jour	$\frac{-1}{2400}$
aller aux toilettes	100	1 fois par jour	$\frac{-1}{2400}$
dormir	100	8 fois par jour	$\frac{-8}{2400}$
se laver	100	2 fois par jour	$\frac{-2}{2400}$
arroser	100	1 fois par semaine	$\frac{-1}{16800}$
regarderTV	100	–	0.0

TAB. 5.4 – La durée des interactions et le calcul de la pente pour l'évolution des propriétés correspondantes (en fonction de la fréquence).

La deuxième simulation

Dans la seconde simulation, les interactions ont une durée identique (100 pas de simulation, sauf pour les interactions de déplacement et l'interaction `cuisiner` qui ont une durée de 1). Ainsi une interaction doit être exécutée plusieurs fois pour obtenir la même durée que dans la simulation précédente. J'ai donc pour cela redéfini les fonctions d'évolutions des propriétés (voir le tableau 5.4). Dans cette simulation, chaque interaction doit simuler une durée d'une heure (soit 100 pas de simulation). Dans une semaine, 168 interactions sont exécutées (si on ne compte pas les interactions de déplacements, ni l'interaction `cuisiner`) dont 6 fois l'interaction `regarderTV`.

Les résultats (voir la figure 5.15) montrent que malgré l'influence des déplacements et de l'interaction `cuisiner` (qui prennent un pas de simulation à chaque exécution) sur l'évolution des propriétés de l'agent, il est possible d'obtenir des résultats permettant de simuler une semaine. Même si un facteur 100 a été mis en place pour simuler 1 heure, la consommation de pas de simulation lors des déplacements représente du temps en moins pour l'exécution de l'interaction `regarderTV` que j'estime être en moyenne de $1,897$ par semaine. En effet, les déplacements représentent $1,129\%$ des interactions effectuées [17]. Sur 168 interactions par semaine, ce pourcentage représente donc $1,897$ interactions par semaine en moyenne. En prenant en compte ce manque et en l'incluant au nombre d'exécutions de l'interaction `regarderTV` par semaine, ce dernier passe à $5,65$ interactions par semaine (soit un nombre assez proche des 6 interactions par semaine).

[17] 542706 pas de simulation, 5359 exécutions d'interactions (hors déplacement et `cuisiner`) et 678 interactions `cuisiner`, soit un total de 536578 pas de simulation qui ne sont pas des déplacements

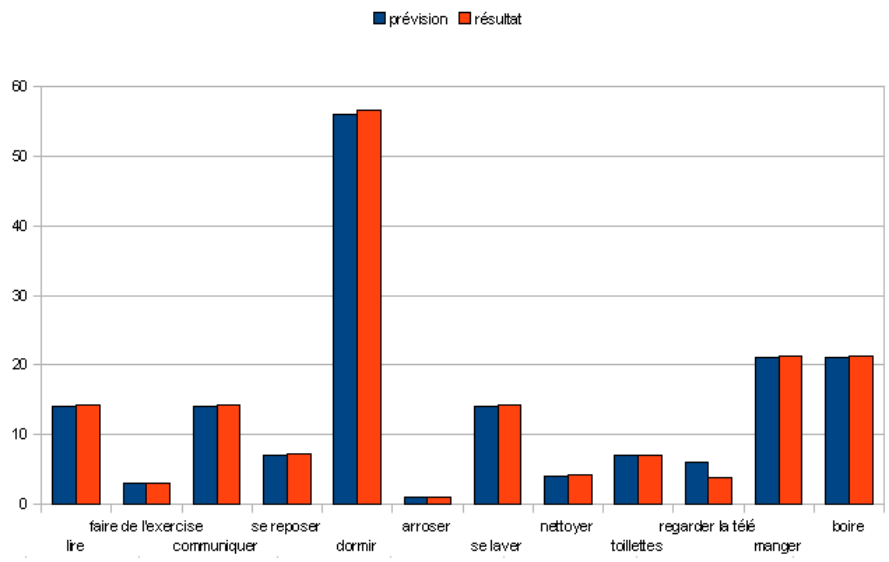


FIG. 5.15 – Exécution moyenne des interactions par semaine (résultat et prévision) après avoir simulé 31 semaines. Le nombre moyen d'exécution de l'interaction `regarderTV` est inférieure à la prévision car cette interaction est exécutée pour satisfaire un but par défaut et à cause de l'influence des déplacements sur l'évolution des propriétés internes de l'agent.

5.2.5 Les profils et les prototypes

Comme je l'ai présenté dans la partie 3.2, tous les agents ont le même planificateur et le même mécanisme de sélection d'action. Si deux agents sont dans le même environnement, avec les mêmes connaissances, les mêmes capacités, les mêmes fonctions de combinaison et le même ensemble de motivations, alors ces agents vont agir différemment s'ils possèdent des profils d'individualité différents. Le but de cette expérimentation est d'illustrer cette proposition. Pour cela j'effectuerai plusieurs simulations. Dans toutes ces simulations, j'utiliserai le même environnement, dont les agents percevront la totalité (i.e. l'agent a un champ de vision assez étendu pour voir l'ensemble de l'environnement dès le début de la simulation), ils possèderont le même moteur comportemental, seuls leurs profils d'individualité seront différents d'un agent à l'autre.

Comme défini dans la partie 3.3.2, il y a deux niveaux de conception possibles suivant les connaissances qu'un utilisateur a de l'ASM. Le premier niveau construit l'ASM, les évaluateurs et des prototypes de profil d'individualité pour que le second niveau n'ait qu'à piocher parmi ces prototypes pour concevoir un comportement. Pour cette expérimentation je définirai plusieurs prototypes et à l'aide de ces derniers je construirai différents profils d'individualité (et donc différents comportements). Ensuite, je comparerai l'exécution des simulations de chaque profil.

Les buts

Les agents auront quatre buts à satisfaire :

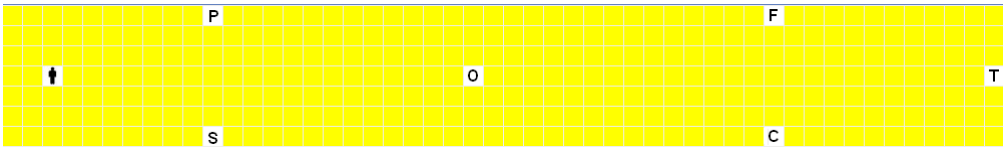


FIG. 5.16 – Environnement de la simulation. L'agent doit résoudre 4 buts impliquant les agents `Ordinateur` (O), `Pomme` (P) `Soda` (S), `Frêne` (F), `Citronnier` (C) et `Trésor` (T).

Le but $g1$: l'agent doit maintenir un niveau d'amusement. La priorité de ce but est liée à une propriété évolutive nommée *entertainment* (i.e. liée à la décroissance de la valeur de cette propriété). Ce but est à satisfaire lorsque la valeur de la propriété *entertainment* est au dessous de 0. Pour cela, un agent `computer` sera présent dans l'environnement, cet agent peut subir l'interaction *jouer* dont l'effet sera d'augmenter la propriété *entertainment* de l'agent acteur de l'interaction. La valeur de la propriété *entertainment* décroit à chaque pas de simulation de 1. La valeur de la priorité du but est égale à l'inverse de la valeur de cette propriété (ou 0 si la valeur de la propriété est au dessus de 0).

Le but $g2$: l'agent doit effectuer une interaction pour remonter son niveau d'énergie. Dans cette simulation la priorité du but sera une constante. Ce but oblige l'agent à faire un choix entre *manger* `une pomme` et *boire* `un soda`.

Le but $g3$: l'agent devra posséder une quantité de bois supérieure à 0. Pour cela, l'environnement possède deux agents : `frêne` et `citronnier`. Ces deux agents peuvent subir l'interaction *couper*, mais ils possèdent une propriété *timeOfCutting* dont la valeur est différente, qui correspond au temps de coupe de l'arbre.

Le but $g4$: l'agent doit posséder le trésor (i.e. l'agent `trésor`).

L'environnement et les interactions

L'environnement est composé comme un parcours (voir la figure 5.16). L'agent débute tout à gauche et il doit se déplacer vers la droite pour résoudre ses 4 buts. Lorsque l'agent a la possibilité de résoudre un but de deux manières différentes, les agents cibles sont placés aux extrémités (haute et basse) du parcours afin de clairement visualiser le choix de l'agent.

Afin que le comportement de l'agent ne semble pas uniquement influencé par la proximité des cibles, j'ai placé l'agent en début de simulation à une distance égale entre le soda et la pomme (voir la figure 5.16), l'ordinateur se trouve au milieu (entre le haut et le bas de l'environnement) et l'interaction *jouer* sera effectuée sur la même case que cet ordinateur (voir la figure 5.17).

Les prototypes

Pour cette expérimentation, j'ai créé huit prototypes de profil d'individualité différents (voir le tableau 5.5). Ces prototypes sont basés sur quatre motivations : la priorité des buts, les préférences de l'agent, l'accomplissement en temps et l'opportunisme.

Les huit prototypes vont par pair, "Mangeur" et "Buveur" sont des préférences sur des interactions augmentant l'énergie de l'agent, "Cupide" et "Hippie" concernent la priorité du

```
prendre :
 condition  =  true
 guard      =  distance(actor, target) < 2
 action     =  add(actor.inventory, target) and remove(target)

manger :
 condition  =  actor.own(target)
 guard      =  true
 action     =  add(actor.energy, target.energy)
               and substract(actor.inventory, target) and destroy(target)

boire :
 condition  =  actor.own(target)
 guard      =  true
 action     =  add(actor.energy, target.energy)
               and substract(actor.inventory, target) and destroy(target)

jouer :
 condition  =  true
 guard      =  distance(actor, target) < 1
 action     =  add(actor.entertainment, target.entertainment)

couper :
 condition  =  true
 guard      =  distance(actor, target) < 2
 action     =  add(actor.wood, target.wood) and remove(target)
               and destroy(target) and wait(actor, target.timeOfCutting)
```

FIG. 5.17 – Principales interactions utilisées (hors déplacement). Les interactions *manger* et *boire* sont similaires, leur distinction est due aux cibles différentes (soda et pomme). Ces interactions nécessitent de posséder l'agent, c'est-à-dire d'avoir effectuée l'interaction *prendre* au préalable. Remarquons que les interactions *prendre* et *couper* se font à une distance maximale de 1 (sur une case voisine), alors que l'interaction *jouer* s'exécute à une distance de 0 de la cible (sur la même case).

but $g4$, "Organisé" et "Étourdi" sont liés à l'accomplissement en temps (i.e. cette motivation est active pour "Organisé" et inactive pour "Étourdi"), "Opportuniste" et "Candide" correspondent à l'action de l'opportunisme sur le comportement (i.e. cette motivation est active pour "Opportuniste" et inactive pour "Candide"). Un agent mangeur favorise les alternatives contenant l'interaction manger alors qu'un agent buveur favorise les alternatives contenant l'interaction boire. Les autres couples de profils s'opposent. Un agent cupide privilégie les résolutions du but $g4$ contrairement à un agent hippie. Un agent organisé prendra en compte les coûts d'exécution des interactions en favorisant les alternatives dont la somme des coûts d'exécution des interactions est la plus faible possible alors que l'étourdi ne le fera pas. Un agent opportuniste privilégiera les alternatives dont la cible de l'interaction exécutable est proche de lui (à partir de 5 cases) tandis qu'un agent candide ne tient pas compte de son voisinage. Le mécanisme de sélection d'action utilisé dans ces simulations sera composé des sept motivations présentées précédemment. Seules les quatre motivations liées aux prototypes auront des paramètres différents d'une simulation à une autre. Les autres valeurs intervenant dans le profil d'individualité de cette expérimentation sont définies dans le tableau 5.6.

Nom du prototype	Description	Profil d'individualité correspondant
Mangeur	Le mangeur préfère manger	$\pi(manger) = 5$
Buveur	Le buveur préfère boire	$\pi(boire) = 5$
Cupide	Le cupide aime l'argent et les trésors, pour lui c'est une priorité	$priority(g4) = 70$
Hippie	Le hippie n'est pas plus attiré par l'argent que ça, il y a peut être des choses plus importantes pour lui	$priority(g4) = 1$
Organisé	L'organisé fait ce qu'il doit faire vite et bien	$\theta_{accT} = 5$
Étourdi	L'étourdi ne fait pas attention, il peut se retrouver à faire un travail pénible car il n'a pas réfléchi avant	$\theta_{accT} = 0$
Opportuniste	L'opportuniste adapte ses actions pour prendre avantage de l'environnement proche	$\theta_{opp} = 5$
Candide	Le candide est un peu niais, il ne tient pas compte de l'environnement proche	$\theta_{opp} = 0$

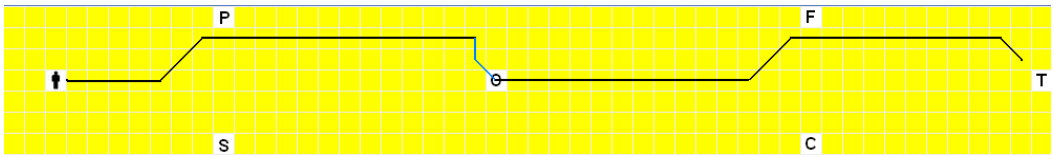
TAB. 5.5 – Les huit prototypes de profil d'individualité qui vont être utilisés.

$priority(g1)$	Fonction dépendante de la propriété *entertainment* dont la valeur est définie par une fonction linéaire décroissante de pente -1 avec 17 comme valeur de départ (à l'étape de simulation $t = 17$, le but est à satisfaire)
$priority(g2)$	12
$priority(g3)$	10
$\pi(a_i)$	1
$cost(a_i)$	1 (sauf pour l'interaction *couper* qui dépend de la propriété *timeOfCutting*)
θ_{rmg}	1
$\theta_{inertia}$	$0, 10$
θ_{accS}	0
`Frêne`.*timeOfCutting*	3
`Citronnier`.*timeOfCutting*	5

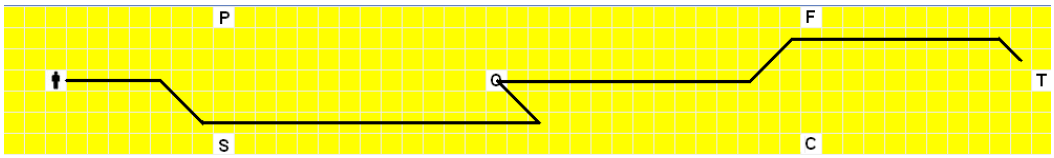
TAB. 5.6 – Les paramètres des profils d'individualités qui ne sont pas définis par les prototypes de profil d'individualité.

Profil	Prototypes utilisés
P1	Mangeur, Hippie, Organisé, Opportuniste
P2	Buveur, Hippie, Organisé, Candide
P3	Buveur, Hippie, Étourdi, Candide
P4	Buveur, Cupide, Organisé, Opportuniste

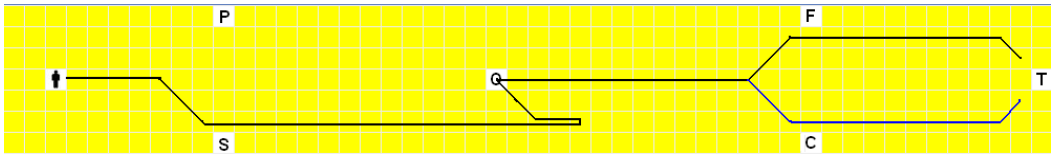
TAB. 5.7 – Les profils utilisés pour les expérimentations.



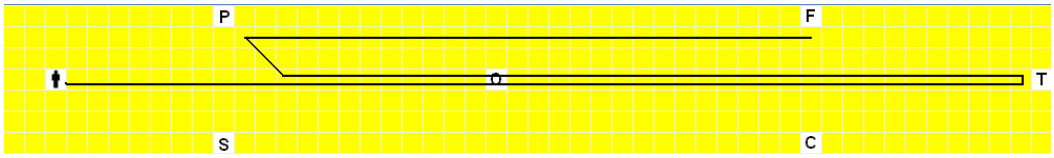
(a) Parcours de l'agent P1, la partie en bleu représente le moment où l'agent passe d'une résolution pour le but $g3$ à une résolution pour le but $g1$ (voir la figure 5.19(a)). L'agent mangeur a préféré *manger* la `Pomme`, opportuniste, il a profité d'être proche de l'`Ordinateur` pour résoudre $g1$. Prenant en compte les coûts de coupe l'agent organisé a choisi de *couper* le `Frêne`. Ce parcours a été exécuté en 56 pas de simulations.



(b) Parcours de l'agent P2. L'agent buveur a préféré *boire* le `Soda`, candide, il a dû faire demi-tour pour résoudre le but $g1$, mais l'accomplissement en temps a permis à cet agent organisé de prendre en compte la résolution du but $g1$ avant l'agent étourdi présenté dans la figure 5.18(c) (i.e. *jouer* a un coût de 1 alors que l'interaction *couper* a un coût de 3 ou 5 selon la cible). Prenant en compte les coûts de coupe l'agent organisé a choisi de *couper* le `Frêne`. Ce parcours a été exécuté en 59 pas de simulations.

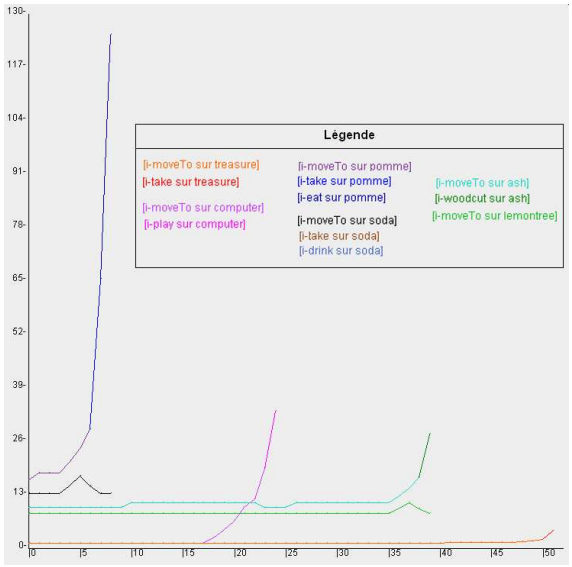


(c) Parcours de l'agent P3, la partie en bleu représente un autre choix possible de l'ASM dû à l'égalité des évaluations et des distances acteur-cible (voir la figure 5.19(c)). Étourdi, l'agent ne prend pas compte les coûts d'exécution des interactions. Les agents `Frêne` et `Citronnier` étant à la même distance de l'agent, le choix a été effectué aléatoirement entre ces deux résolutions donnant deux parcours différents pour la même simulation. Le parcours entièrement noir a été exécuté en 63 pas de simulations, le parcours avec la partie bleue en 65 pas.

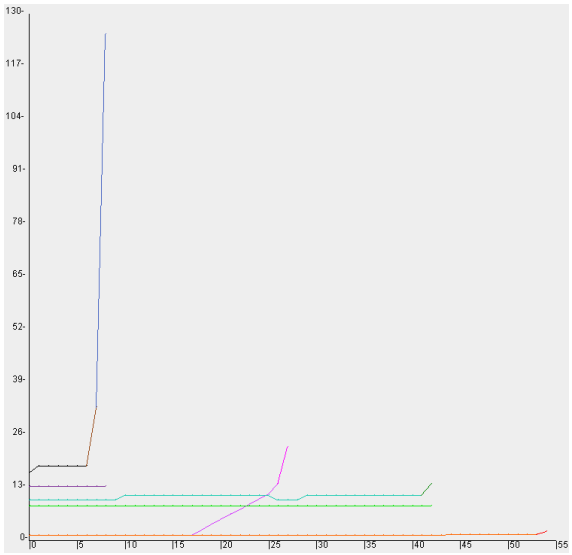


(d) Parcours de l'agent P4. La priorité des buts du prototype "Cupide", donne un comportement borné où la première chose que l'agent doit faire est d'obtenir le `Trésor`. L'aller-retour d'un bout à l'autre de l'environnement dû au prototype "Cupide" donne un parcours qui a été exécuté en 115 pas de simulations.

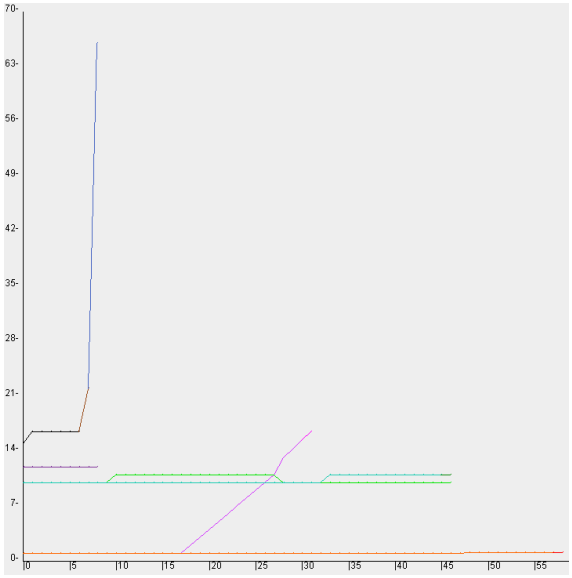
FIG. 5.18 – Les quatre parcours des quatre profils. Ces parcours ont été regroupés afin de pouvoir être comparés.



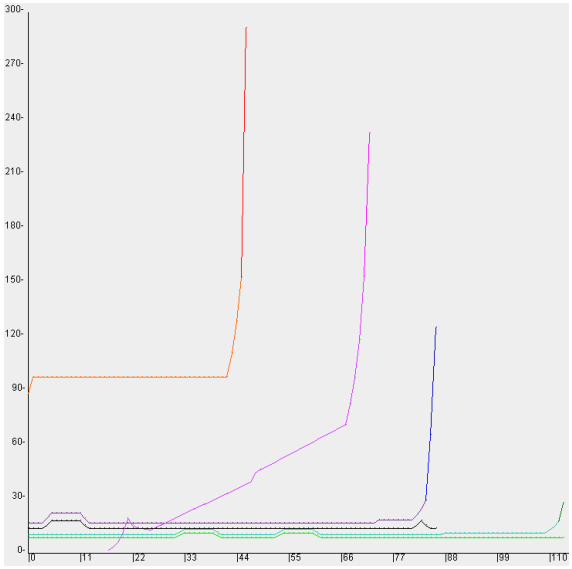
(a) Évaluation des interactions exécutables pour l'agent P1.



(b) Évaluation des interactions exécutables pour l'agent P2.

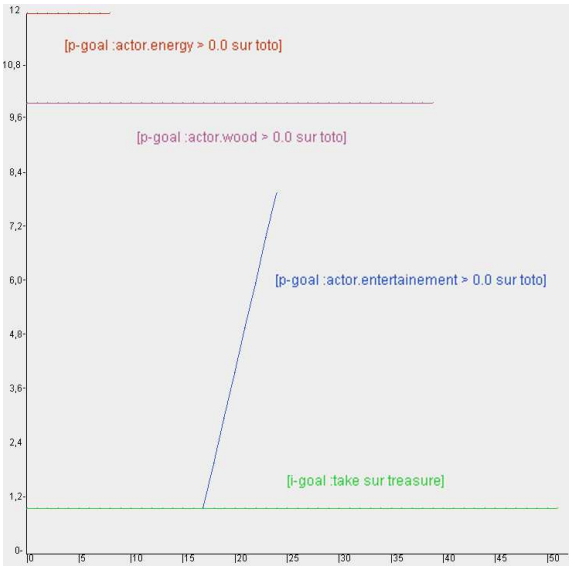


(c) Évaluation des interactions exécutables pour l'agent P3. Ce graphe montre l'évaluation aléatoire de l'alternative en cas d'égalité des meilleures évaluations et équidistances entre l'acteur et la cible. On peut voir par l'effet de l'inertie qu'à la première égalité l'ASM avait sélectionné le `Citronnier` et à la seconde le `Frêne`.

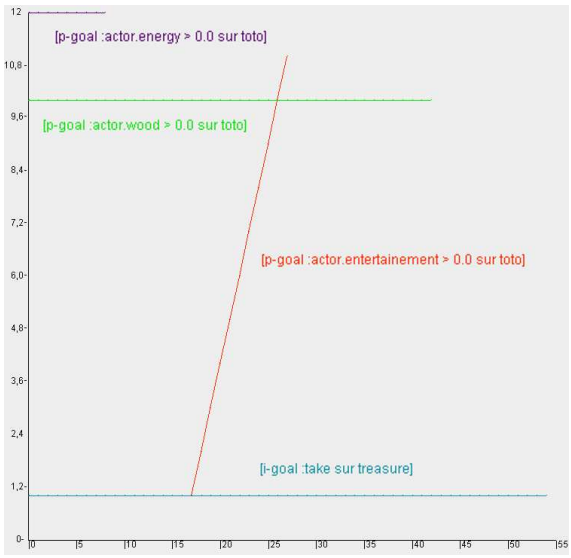


(d) Évaluation des interactions exécutables pour l'agent P4.

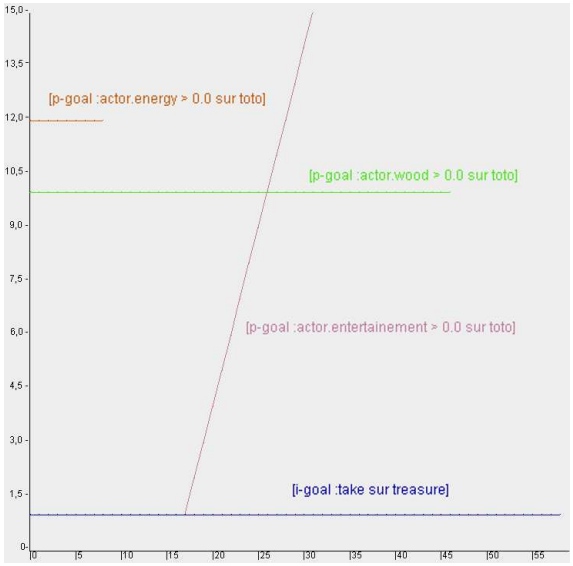
FIG. 5.19 – Évaluation des interactions exécutables pour les quatre profils. Les attentes (wait) ne sont pas prises en compte dans ces graphes. Ces graphes ont été regroupés afin de pouvoir être comparés.



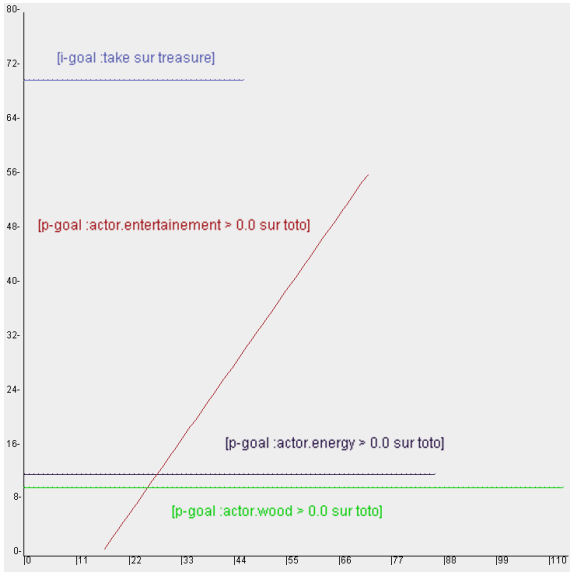
(a) Évolution des priorités des buts pour l'agent P1.



(b) Évolution des priorités des buts pour l'agent P2.



(c) Évolution des priorités des buts pour l'agent P3.



(d) Évolution des priorités des buts pour l'agent P4.

FIG. 5.20 – Évolution des priorités des buts pour les quatre profils. Les attentes (wait) ne sont pas prises en compte dans ces graphes. Ces graphes ont été regroupés afin de pouvoir être comparés.

À l'aide de ces huit prototypes, je peux construire jusqu'à 16 profils d'individualité différents[18]. Dans cet expérimentation, je présenterai quatre profils différents (voir le tableau 5.7). La construction de ces prototypes correspond au deuxième niveau de conception, c'est-à-dire créer des profils différents en piochant des prototypes dans d'un ensemble existant. Nous pouvons déjà prévoir que le profil P3 va provoquer une égalité dans l'évaluation de l'interaction *couper* sur les cibles `Frêne` et `Citronnier` qui se trouvent placées à égale distance de l'agent. Si plusieurs évaluations ont la même meilleure note, l'ASM résout cette égalité en privilégiant l'interaction exécutable la plus proche de l'agent acteur. Dans notre cas, les deux cibles sont à la même distance de l'acteur, cette seconde égalité est résolue par l'ASM par un choix aléatoire.

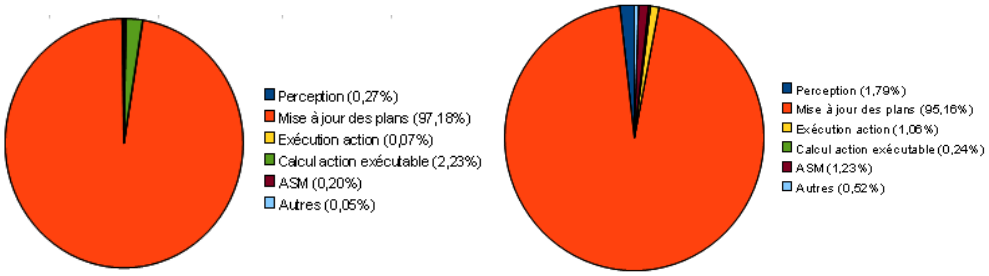
Ces profils ont été testés plusieurs fois, les résultats par profil sont les mêmes pour toutes les simulations testées (voir la figure 5.18). Les résultats obtenus correspondent aux profils d'individualités utilisés. Les agents mangeurs ont choisi de manger la pomme, les agents opportunistes ont pris en compte la proximité des cibles dans le choix des actions à effectuer, les agents organisé ont privilégié l'arbre dont le coût de coupe est moins important (voir la figure 5.19). Nous pouvons également voir que le mécanisme de sélection d'action n'offre pas des comportements prédéfinis et que le choix des profils d'individualité a un impact important sur le comportement obtenu. En effet, dans le cas de l'agent cupide, l'importance de la priorité de son but (voir la figure 5.20), n'a pas permis aux autres motivations d'avoir un impact assez important pour influencer le choix de la résolution.

Dans cette expérimentation, j'ai mis en évidence les deux niveaux de conception par l'utilisation de prototypes de profil d'individualité. J'ai pu ainsi créer des profils d'individualité en piochant parmi des prototypes existants. Comme nous l'avons vu, l'utilisation des prototypes de profil d'individualité permet de concevoir facilement différents profils d'individualité et donc différents comportements. De plus, cette expérimentation a également montré que des agents dans le même environnement avec les mêmes capacités, les mêmes connaissances et le même moteur comportemental pouvaient avoir des comportements différents s'ils avaient des profils d'individualité différents.

5.2.6 Les performances

L'implémentation du mécanisme de sélection d'action dans CoCoA a été réalisée dans le but de pouvoir conduire des expérimentations. Bien que cette implémentation puisse être optimisée afin d'améliorer les performances en temps de calcul de l'ASM, j'ai effectué des tests afin de calculer le surcoût de son utilisation dans la plateforme de simulation CoCoA. Pour cela, j'ai utilisé Profiler4J un outil permettant de connaître le nombre de fois qu'une méthode est appelée ainsi que le temps d'exécution total et moyen de cette méthode. Profiler4J introduit un surcoût de par le filtrage qu'il effectue afin de calculer les données de profilage. Le temps de chaque appel dépend également de l'ordinateur utilisé pour les tests, de ses disponibilités de calcul et taille mémoire allouée à la JVM, je montrerai les performances de l'ASM à travers le surcoût qu'il engendre en moyenne.

[18] Bien que les prototypes Mangeur et Buveur peuvent être utilisés simultanément (sans poser de problème de cohérent sur les paramètres des motivations), les utiliser en même temps ne permettrait pas de mettre en évidence le choix de la résolution du but $g2$ basé sur les préférences de l'agent. C'est pourquoi je définis pouvoir construire jusqu'à 16 profils différents au lieu de 32



(a) Première simulation sur l'influence des propriétés.

(b) Deuxième simulation sur l'influence des propriétés.

FIG. 5.21 – Simulations sur l'influence des propriétés. Beaucoup de buts et donc beaucoup de plans à maintenir. Les alternatives sont courtes en nombres d'actions, le coût de l'ASM est assez faible.

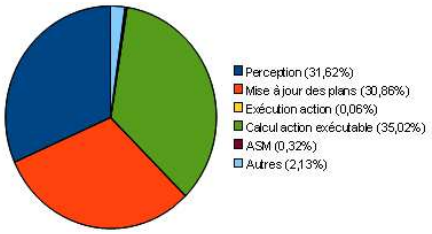


FIG. 5.22 – Simulation sur les prototypes et les profils d'individualité. L'agent a un champ de perception très important, le surcoût par rapport aux autres simulations est visible. Les alternatives ne contiennent pas beaucoup d'interactions, le coût de l'ASM est assez faible.

Les simulations "toys" (voir la partie 5.2.2) n'ont pas été évaluées, en effet de par leur simplicité ces simulations ne fourniraient pas de valeurs significatives des performances de l'ASM. Les simulations de l'expérimentation sur l'influence des propriétés (voir la partie 5.2.4) proposent de maintenir douze propriétés via douze buts. En moyenne chaque but peut être résolu par deux cibles différentes. Bien que nombreuses, les alternatives sont courtes en nombre d'actions. La satisfaction d'un but ne nécessite qu'une interaction et la gestion des déplacements. De par la taille des alternatives, leurs collectes et leurs évaluations ont un coût assez faible (voir la figure 5.21).

Les simulations sur les prototypes et les profils d'individualité (voir la partie 5.2.5) sont dans un environnement d'une taille plus importante que l'expérimentation sur l'influence des propriétés. L'environnement ne possédant pas d'obstacle, le calcul des chemins est plus long (car il y a plus de possibilités) mais le calcul à effectuer est beaucoup moins important (moins de buts, moins de déplacement à planifier). De plus, dans cette expérimentation, l'agent possède un champ de perception lui permettant de connaitre l'ensemble de son environnement dès le début. Nous pouvons constater le surcoût lié à la perception dans la figure 5.22.

L'expérimentation présentant la combinaison des motivations (voir la partie 5.2.3) se déroule dans un environnement d'une taille plus importante que les deux précédentes. L'agent ne possède que deux buts, mais il existe plusieurs façons de les résoudre. De plus, la taille importante de l'environnement requiert de nombreux calculs de déplacement (dus aux nombreuses

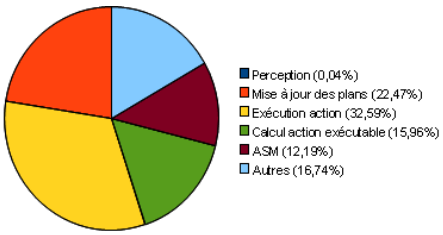


FIG. 5.23 – Simulation sur la combinaison des motivations. L'environnement est de plus grande taille que les autres expérimentations, le coût de l'ASM s'en trouve augmenté. En effet, les calculs des distances représentent $43,58\%$ du temps total. Le temps nécessaire à l'accomplissement en espace représente $12,1869\%$ du temps total de la simulation.

possibilités de déplacement). Ces calculs de déplacement prennent une part importante du temps de calcul total ($43,58\%$) de la simulation. L'accomplissement en espace, basé sur une estimation des déplacements à effectuer, entraine un surcoût dans l'évaluation des alternatives par le mécanisme de sélection d'action(voir la figure 5.23).

Dans les simulations que j'ai effectuées, nous pouvons voir que le coût de l'ASM dépend de la taille des plans (voir la figure 5.21 et la figure 5.22). En effet, plus un plan possède d'alternatives (i.e. plus il existe de façons différentes de résoudre un but) et plus les alternatives contiennent d'actions (i.e. plus les résolutions sont longues) plus la recherche dans un *Arbre – et/ou* et l'évaluation prennent du temps. De même, la taille de l'environnement est également un facteur pouvant augmenter le coût de l'utilisation de l'ASM. En effet, si l'environnement est de taille importante, les calculs de distance prendront beaucoup de temps. Parmi ces calculs de distance, il y a les calculs de chemin nécessaire à l'accomplissement en espace (voir la figure 5.23).

Conclusion

Dans ce chapitre je me suis focalisé sur la conception de comportements. J'ai d'abord présenté l'atelier de conception qui a été mis en place afin de permettre à l'utilisateur de concevoir des comportements indépendamment des simulations. Cet atelier, destiné à un utilisateur averti, permet de concevoir (en totalité ou partiellement) le comportement d'un agent tout en exploitant les fonctionnalités offertes par le mécanisme de sélection d'action. De plus j'ai mis en place un système de monitoring permettant de suivre l'évolution des évaluations des alternatives de chaque agent afin de mettre en évidence l'influence de l'ASM sur le comportement. Dans le but de montrer l'importance des différents éléments intervenant dans le mécanisme de sélection d'action, j'ai effectué plusieurs expérimentations. La première est composée de plusieurs petites simulations qui montrent l'influence de chaque motivation sur le comportement de l'agent. La seconde a illustré l'action complémentaire d'un ensemble de motivations et de la combinaison de ces motivations sur le comportement de l'agent. La troisième a permis de comprendre l'importance des propriétés internes de l'agent dont la valeur évolue durant la simulation. La dernière a montré comment la définition de prototypes et leurs utilisations ont permis d'obtenir, dans le même environnement, différents comportements pour des agents qui possèdent les mêmes capacités et connaissances.

Conclusion Générale

« La fin justifie les moyens. Mais qu'est-ce qui justifiera la fin ? »
Albert Camus

Conclusion

Le comportement d'un agent se définit à partir de l'observation des actions qu'il exécute dans un environnement. Chaque action exécutée résulte d'un choix de l'agent parmi ses capacités. Le comportement se construit donc à partir des actions que l'agent peut effectuer pour résoudre ses buts et d'un choix influencé par ses traits de caractère. L'une des applications possibles des travaux de simulation de comportements est les jeux vidéo. La modélisation de comportements dans les jeux vidéo est un critère important. Plus les comportements des personnages dans un jeu sont variés, plus l'immersion du joueur est intense et captivante. Une solution pour rendre les comportements plus variés est d'accroître le nombre des traits de caractère de l'agent afin d'augmenter la diversité des influences sur son comportement. Or, ces traits de caractère peuvent être de natures différentes. Il donc est important de définir une structure permettant à la fois de prendre compte tous ces éléments, d'autoriser l'ajout et la suppression d'un trait de caractère tout en gardant la cohérence du comportement obtenu.

Ma contribution consiste à définir des comportements qui sont composés de deux parties : le **raisonnement** et l'**individualité**. Le comportement est construit à partir des capacités de l'agent lui permettant de résoudre ses buts, le raisonnement, et d'un choix influencé par ses traits de caractère, l'individualité. Le raisonnement utilise un planificateur pour déduire l'ensemble des résolutions possibles des buts de l'agent à partir de ses connaissances et de ses capacités. L'individualité utilise un mécanisme de sélection d'action (ASM pour *Action Selection Mechanism*) afin de déterminer la meilleure action parmi l'ensemble des actions exécutables. C'est sur cette dernière partie que mes recherches se focalisent.

Le mécanisme de sélection d'action que je propose se base sur les notions d'alternative et de motivation. Une alternative est une séquence d'actions composée d'une action exécutable, d'actions intermédiaires et de l'action permettant de résoudre un but. Les alternatives sont des prévisions à moyen terme des résolutions possibles des buts, elles sont calculées par la partie raisonnement du comportement. Les motivations sont l'expression de traits du caractère de l'agent qui influencent le choix de l'action à exécuter. Leur rôle est d'évaluer à chaque ins-

tant les actions exécutables par l'agent en prenant en compte les alternatives correspondantes. Les motivations sont définies indépendamment du contexte d'application du comportement, elles sont donc réutilisables pour d'autres agents et d'autres simulations. Elles sont également indépendantes les unes des autres, rendant le mécanisme de sélection d'action robuste aux ajouts et aux suppressions de motivation. Enfin, les motivations sont paramétrables afin de créer des profils d'individualité. Ces profils sont l'expression de la manière dont la motivation va influencer l'individualité de l'agent. Les paramètres ont donc une interprétation qui facilite leur utilisation. Ma proposition repose sur une définition d'individualité sans a priori sur l'environnement dans lequel le comportement sera appliqué. Cette définition se veut être réutilisable (même partiellement) pour d'autres environnements et d'autres agents.

Ces travaux s'inscrivent dans le cadre du projet CoCoA de l'équipe SMAC de Lille. Le mécanisme de sélection d'action a été implémenté et testé dans la plateforme de ce projet. Cette tâche a nécessité la création d'algorithmes tels que la recherche d'alternatives dans un $Arbre - et/ou$ et l'ajout de nouvelles notions comme les propriétés qui évoluent pendant la simulation. Ce mécanisme vient enrichir le planificateur basé sur l'approche centrée interaction afin d'obtenir un moteur comportemental générique et une définition du comportement réutilisable tant sur la partie raisonnement que sur la partie individualité. L'ASM que j'ai mis en place dans CoCoA utilise sept motivations qui répondent aux critères de Tyrrell définissant un bon mécanisme de sélection d'action. Plusieurs simulations ont été réalisées afin de montrer l'importance de chaque motivation, de la combinaison des motivations, de la notion de propriétés évolutives et des profils d'individualité. Enfin des outils de conception et de visualisation ont également été mis en place.

Perspectives

Nous avons vu dans une simulation que les prototypes de profils d'individualité permettent de simplifier la conception des comportements. Les prototypes utilisés n'avaient aucune motivation en commun. Lorsque l'on veut utiliser l'ensemble des prototypes paramétrant les mêmes motivations, la question de la combinaison des paramètres se pose. Selon moi, deux solutions sont à explorer, à tester et à comparer. La première serait de définir une fonction combinant les paramètres de différents profils pour une même motivation. La difficulté est de maintenir la cohérence entre les différents prototypes et de faire des choix. En effet, comment peut-on combiner une inhibition et une forte attraction ? Pour une même motivation, l'inhibition est-elle toujours prioritaire ? La seconde serait d'utiliser plusieurs fois une même motivation (i.e. plusieurs instances d'une même motivation) mais avec des paramètres différents. Cette solution bien qu'elle soit actuellement réalisable dans CoCoA, risque de rendre plus difficile la compréhension des profils.

Dans [Cañ97], l'auteur prend en compte l'état émotionnel de l'agent dans la sélection d'action. Dans cette approche, certaines émotions influencent la production d'hormones, ces dernières sont prises en comptes lors du calcul des tendances des motivations et de l'importance des stimuli (internes et externes). L'état émotionnel qui modifie la production d'hormones influence donc le comportement de l'agent. Cette influence est tout à fait naturelle et doit être prise en compte par le mécanisme de sélection d'action. Pour cela, j'envisage deux pistes possibles. La première est de pouvoir modifier l'impact des motivations, la seconde est d'ajouter des motivations émotionnelles. Dans le première cas, les émotions pourraient modifier les pa-

ramètres du profil d'individualité de l'agent. Par exemple, les préférences de l'agent pourraient être modifiées sous l'influence d'une émotion, la *peur* pourrait favoriser la fuite, l'*ennui* favoriserait le changement d'alternatives en diminuant l'impact de l'inertie et la *curiosité* pousserait l'agent à favoriser les résolutions présentant des explorations. Dans le deuxième cas, comme les motivations sont indépendantes, il est possible d'ajouter de nouvelles motivations sans toucher au mécanisme de sélection d'action, ces motivations émotionnelles évalueraient les alternatives suivant l'état émotionnel de l'agent.

J'envisage également d'utiliser le mécanisme de sélection d'action pour la coopération d'agents cognitifs. Je suis convaincu que des motivations peuvent permettre d'exprimer des comportements du type *altruisme* ou de prendre en compte la *réputation* d'autres agents. L'altruisme ou empathie[CC03] a pour objet d'exprimer de quelle manière l'agent est prédisposé à aider les autres agents. La réputation d'un agent peut être issue d'échanges sociaux précédents où d'une réputation sociale accordée aux autres agents. Ainsi un agent exécutera plus volontiers des buts (ordres) provenant d'un agent possédant une bonne réputation. Ces motivations permettraient d'obtenir des comportements coopératifs, autres que ceux actuellement mis en place dans CoCoA. Pour le moment, les comportements coopératifs d'agents cognitifs dans CoCoA nécessitent l'utilisation d'une structure hiérarchique préalablement définie par la désignation d'un chef connu par tous les agents subordonnés[DMR05]. Or, la réputation des agents peut-être vue comme une relation hiérarchique implicite. Ainsi, l'altruisme et la réputation permettraient une coopération sans pour autant avoir défini explicitement une structure hiérarchique entre les agents. Mais pour pouvoir effectuer des comportements coopératifs dans CoCoA, il est nécessaire qu'un agent ne puisse pas exécuter une interaction qui rendra la réalisation d'un but d'un autre agent impossible. C'est pourquoi la notion d'actions mutuellement exclusives (MUTEX) doit être prise en compte par la partie raisonnement afin de limiter les situations de blocage. Pour cela l'utilisation d'un planificateur comme GraphPlan[BF95, GA00] peut être une solution.

Une autre piste serait d'utiliser un système d'allocation de ressources décentralisée et mon mécanisme de sélection d'action pour définir une tâche collective. L'idée est de voir une action faisant partie d'un plan comme une ressource à distribuer et d'attribuer comme préférence l'évaluation de cette action par l'ASM. Ainsi, le système d'allocation de ressources décentralisée pourrait donner une distribution des tâches maximisant le bien commun, c'est-à-dire distribuer les alternatives à résoudre en prenant en compte les individualités des agents. La distribution des tâches pourrait également se baser sur les émotions et la notion de charge cognitive présentée dans [PCS03].

Bibliographie

[And03] Fenintsoa Andriamasinoro : *Proposition d'un modèle d'agents hybrides basé sur la motivation naturelle.* Thèse de doctorat, Université de la Réunion, 2003.

[Arr51] K.J. Arrow : *Social choice and individual values.* J. Wiley, New York, 1951.

[BF81] Avron Barr et Edward Feigenbaum : *The Handbook Of Artificial Intelligence*, volume I, chapitre II. Pitman, 1981.

[BF95] Avrim L. Blum et Merrick L. Furst : Fast planning through planning graph analysis. *In Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95)*, pages 1636–1642, 1995.

[Blu94] Bruce Blumberg : Action selection in hamsterdam : Lessons from ethology. *In Proceedings of 3rd International Conference on the simulation of Adaptative behaviour*, pages 108–117. MIT Press, 1994.

[Bon08] Laetitia Bonte : IA, http ://devblog.dofus.com/fr/billets/53-ia.html, 2008.

[BQLC03] Cédric Buche, Ronan Querrec, Pierre De Loor et Pierre Chevaillier : Mascaret : Pedagogical multi-agents system for virtual environment for training. *In International Conference on Cyberworlds (CW 2003)*, pages 423–431. IEEE Computer Society, 2003.

[Bro85] R. A. Brooks : A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, 1985.

[Bue05] Axel Buendia : *Une architecture pour la conception de modèles décisionnels : application aux créatures virtuelles dans les jeux vidéo.* Thèse de doctorat, Institut International du Multimédia, Paris, Novembre 2005.

[Bur36] Albert Burloud : *Principes d'une psychologie des tendances.* Librairie Félix Alcan, Paris (France), 1936.

[CAG07] Lola Cañamero et Orlando Avila-García : A bottom-up investigation of emotional modulation in competitive scenarios. *In Affective Computing and Intelligent Interaction (ACII), Second International Conference*, volume 4738 de *Lecture Notes in Computer Science*, pages 398–409, Lisbon, Portugal, 2007. Springer.

[Cañ97] Dolores Cañamero : Modeling motivations and emotions as a basis for intelligent behavior. *In* W. Lewis Johnson, éditeur : *Proceedings of the First International Symposium on Autonomous Agents, Agents'97*, pages 148–155. The ACM Press, 1997.

[CC03] François Charpillet et Iadine Chadès : Modèle de conception de sma coopératifs par planification réactive. *In* Andreas, Brahim Chaib-draa, Philippe

MATHIEU et HERZIG, éditeurs : *Modèles formels de l'interaction, Actes des Secondes Journées Francophones*, pages 51–60. Cépaduès, 2003.

[Che06] Pierre CHEVAILLIER : *Les systèmes multi-agents pour les environnements virtuels de formation.* Habilitation à diriger des recherches, Université de Bretagne Occidentale, Brest, 2006.

[Clo05] Christine CLOAREC : La motivation au travail : Tour d'horizon des grandes théories, 2005.

[Dev07] Damien DEVIGNE : *Approche centrée interaction pour la simulation d'agents cognitifs situés.* Thèse de doctorat, Université des Sciences et Technologies de Lille, Juillet 2007.

[DMR04] Damien DEVIGNE, Philippe MATHIEU et Jean-Christophe ROUTIER : Planning for spatially situated agents. *In Proceedings of IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'04)*, pages 385–388. IEEE Press, 2004.

[DMR05] Damien DEVIGNE, Philippe MATHIEU et Jean-Christophe ROUTIER : Teams of cognitive agents with leader : how to let them some autonomy. *In* Graham KENDALL et Simon LUCAS, éditeurs : *Proceedings of IEEE Symposium on Computational Intelligence Games (CIG'05)*, pages 256–262, 2005.

[Don01] Jean-Paul DONCKÈLE : *Profils D'enseignants, Profils D'enseignés. Se Connaître, Les Connaître.* Chronique Sociale, 2001.

[dS06] Etienne de SEVIN : *An Action Selection Architecture for Autonomous Virtual Humans in Persistent Worlds.* Thèse de doctorat, Ecole Polytechnique Fédérale de Lausanne, 2006.

[DZ01] Mark A. DEPRISTO et Robert ZUBEK : Being-in-the-world. *In AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment*, 2001.

[Eva02] Richard EVANS : *AI Game Programming Wisdom : Varieties of Learning*, volume I, chapitre 11. Pitman, 2002.

[Fer95] Jacques FERBER : *Les systèmes multi-agents, vers une intelligence collective.* InterEditions, Paris (France), 1995.

[FN71] R. FIKES et N. J. NILSSON : Strips : A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[GA00] Emmanuel GUÉRÉ et Rachid ALAMI : An accessibility graph learning approach for task planning in large domains. *In Proceedings of the 14th Workshop "New Results in Planning, Scheduling and Design" (PuK2000)*, 2000.

[GCM97] Stephen GRAND, Dave CLIFF et Anil MALHOTRA : Creatures : Artificial life autonomous software agents for home entertainment. *In the 1st Intl. Conf. on Autonomous Agents*, 1997.

[HYH09] Frederick W. P. HECKEL, G. Michael YOUNGBLOO, et D. Hunter HALE : Behaviorshop : An intuitive interface for interactive character design. *In Proceedings of the Fifth Artificial Intelligence for Interactive Digital Entertainment Conference*, 2009.

[Isl05] Damian ISLE : Handling complexity in the halo 2 ai. *In Game Developers Conference*, San Francisco, France, 2005.

[KMP08] Yoann KUBERA, Philippe MATHIEU et Sébastien PICAULT : Interaction-oriented agent simulations : From theory to implementation. *In Proceedings of the 18th European Conference on Artificial Intelligence (ECAI'08)*, pages 383–387. IOS Press, 2008.

[Kud10] The kodu project. microsoft research. http ://fuse.microsoft.com/kodu/, 2010.

[Lai01] John E. LAIRD : It knows what you're going to do : adding anticipation to a quakebot. *In* Jorg P. MULLER, Elisabeth ANDRE, Sandip SEN et Claude FRASSON, éditeurs : *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 385–392, Montreal, Canada, 2001. ACM Press.

[LNR87] John E. LAIRD, Allen NEWELL et Paul S. ROSENBLOOM : Soar : An architecture for general intelligence. *Artificial Intelligence*, pages 1–64, 1987.

[LvL00] John E. LAIRD et Michael van LENT : Human-level AI's Killer Application : Interactive Computer Games. *In Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence.*, pages 1171 – 1178. AAAI, 2000.

[Mae90] Pattie MAES : A bottom-up mechanism for behavior selection in an artificial creature. *In Proceedings of the first international conference on simulation of adaptive behavior on From animals to animats*, pages 238–246, Cambridge, MA, USA, 1990. MIT Press.

[Mas54] A.H. MASLOW : *Motivation and personality.* Harper, 1954.

[Mas98] A.H. MASLOW : *Toward a Psychology of Being, 3rd Edition.* Lowry Richard, J.Wiley and Sons, 1998.

[MP93] Jörg P. MÜLLER et Markus PISCHEL : The agent architecture inteRRaP : Concept and application. Rapport technique, German Research Center for Artificial Intelligence, 1993.

[MRU01] Philippe MATHIEU, Jean-Christophe ROUTIER et Pascal URRO : Un modèle de simulation agent basé sur les interactions. *In Actes des Premières Journées Francophones sur les Modèles Formels de l'Interaction (MFI'01)*, volume 3, pages 407–418, 2001.

[Nar98] A. NAREYEK : Specification and development of reactive systems. *In 1998 AIPS Workshop*, pages 7–14, Menlo Park California, 1998. AAAI Press.

[NDA08] Jean-Baptiste NDAGIJIMANA : Motivation et réussite des apprentissages scolaires. Mémoire de D.E.A., école normale supérieure - Université de Bouake (Côte d'ivoire), 2008.

[Ork06] Jeff ORKIN : Three states and a plan : The a.i. of f.e.a.r. *In Proceedings of the Game Developers Conference*, 2006.

[PCS03] Pierre De Loor PIERRE CHEVAILLIER et Cyril SEPTSEAULT : Les émotions : une métaphore pour la résolution de problèmes dynamiques distribués. *Technique et Science Informatiques*, 22(4):331–344, 2003.

[Per09] Camille PERSON : Gestion de la mémoire d'agents cognitifs. Mémoire de D.E.A., Université de Lille 1, 2009.

[PS04] Marc PONSEN et Pieter SPRONCK : Improving adaptive game ai with evolutionary learning. *In* Stéphane Natkin QUASIM MEHDI NORMAN GOUGH et David

Al-Dabass, éditeurs : *Computer Games : Artificial Intelligence, Design and Education (CGAIDE 2004)*, pages 389–396, University of Wolverhampton, November 2004.

[Rat95] Spencer A. Rathus : *Psychologie générale.* Études Vivantes (Montréal), 1995.

[RG06] G. Robert et A. Guillot : MHiCS, une architecture de sélection de l'action adaptative pour joueurs artificiels. *In* T. Cazenave, éditeur : *Intelligence Artificielle et Jeu*, pages 47–80. Hermès, Paris, France, 2006.

[RN03] S. J. Russell et Norvig : *Artificial Intelligence : A Modern Approach (Second Edition).* Prentice Hall, 2003.

[Ros97] J. K. Rosenblatt : DAMN : A distributed architecture for mobile navigation. *In Journal of Experimental and Theoretical Artificial Intelligence*, pages 339–360. AAAI Press, 1997.

[Rou05] Jean-Christophe Routier : *Conception par agent orientée compétences.* Habilitation à diriger des recherches, Université des Sciences et Technologies de Lille, Novembre 2005.

[Sch05] Bernd Schmidt : Human factors in complex systems : The modelling of human behaviour. *In Simulation in wider Europe, 19th European Conference on Modelling and Simulation (ECMS 2005)*, pages 5–14, 2005.

[Sep07] Cyril Septseault : *Représentation d'environnements virtuels informés de leur dynamique par un personnage autonome en vue d'une crédibilité comportementale.* Thèse de doctorat, Université de Bretagne occidentale, 2007.

[Set98] Anil K Seth : Evolving action selection and selective attention without actions, attention, or selection. *In the 5th International Conference on Simulation of Adaptive Behavior*, pages 139–147. MIT Press, 1998.

[SPSKP06] Pieter Spronck, Marc Ponsen, Ida Sprinkhuizen-Kuyper et Eric Postma : Adaptive game ai with dynamic scripting. *Machine Learning*, 63(3):217–248, 2006.

[Toz02] Paul Tozour : The perils of ai scripting. *In AI Game Programming Wisdom*, pages 541–547, 2002.

[Tyr93a] Toby Tyrell : The use of hierarchies for action selection. *In Proceedings of the second international conference on From animals to animats 2 : simulation of adaptive behavior*, pages 138–147, Cambridge, MA, USA, 1993. MIT Press.

[Tyr93b] Toby Tyrrell : *Computational Mechanisms for Action Selection.* Thèse de doctorat, University of Edinburgh, 1993.

[Wel99] Daniel S. Weld : Recent advances in AI planning. *AI Magazine*, 20(2):93–123, 1999.

[Wit99] Mark Witkowski : Integrating unsupervised learning, motivation and action selection in an a-life agent. *In 5th European Conference on Artificial Life (ECAL-99)*, pages 355–364, 1999.

Annexe A

Le choix de la fonction de combinaison

Dans cette partie, je présenterai les différentes fonctions mathématiques que j'ai testées comme candidates potentielles pour être des fonctions de combinaisons $Comb$ dans CoCoA. Pour chaque fonction, je montrerai en quoi elle est une bonne ou une mauvaise candidate. Pour cela, Je me baserai sur l'indépendance de chaque motivation, sur l'impact de l'ajout d'une motivation attractive ou répulsive, sur la possibilité d'exprimer l'attraction, la neutralité, la répulsion et l'inhibition et sur les intervalles permettant d'exprimer ces quatre évaluations. Chaque fonction candidate ϕ sera présentée tel que :

- alt est une alternative et ϕ est une fonction de combinaison avec n évaluateurs, tels que $\phi(alt) = Comb(\{\gamma_1(alt), ..., \gamma_n(alt)\})$.
- ϕ' la même fonction candidate avec un évaluateur supplémentaire nommé γ_m, telle que $\phi'(alt) = Comb(\{\gamma_1(alt), ..., \gamma_n(alt)\}, \gamma_m(alt)\})$.

A.1 La fonction puissance

La fonction puissance que j'ai testée est définie telle que $\phi'(alt) = (\phi(alt))^{\gamma_m(alt)}$ avec $\phi_0(alt) > 1$. Pour cette fonction, l'évaluation 0 ($\gamma_i(alt) = 0$), bloque l'évaluation finale à 1 ($\phi(alt) = 1$). La valeur 0 bloque les évaluations des autres motivations et permettant d'exprimer une inhibition. La valeur 1 ne change pas l'évaluation finale et permet d'exprimer une évaluation neutre. De plus, toutes les valeurs $\gamma_i(alt) > 1$ augmentent la valeur finale et toutes les valeurs v telles que $v \in]0; 1[$ diminuent la valeur finale. Le domaine serait donc $\mathbb{R}^+$. Avec l'augmentation du nombre d'évaluations, cette fonction peut donner des évaluations avec de grands nombres. Se pose alors la question de l'intérêt d'obtenir de grands nombres et de les manipuler. De plus, les calculs devenant vite complexes, leur vérification n'est pas toujours simple. Bien que cette fonction permette d'exprimer les quatre évaluations possibles, dans la pratique, l'évaluation finale demande des calculs pouvant être complexes et une manipulation de grands nombres dont l'intérêt est discutable.

A.2 La fonction racine

La fonction racine que j'ai testée est définie telle que $\phi'(alt) = \sqrt[\gamma_m(alt)]{(\phi(alt))}$. La fonction racine est une fonction puissance donc les évaluations sont sous la forme $\frac{1}{\gamma_m(alt)}$. Les valeurs des évaluations peuvent donc encore être de grands nombres. Pour la fonction racine, il est possible d'exprimer :

- Soit une attraction est une valeur $\gamma_m(alt) \in]0;1[$ et donc la meilleure action est l'action ayant obtenu la plus grande évaluation finale.
- Soit une attraction est une valeur $\gamma_m(alt) > 1$ et donc la meilleure action est l'action ayant obtenu la plus petite évaluation finale.

Contrairement à la fonction puissance, il n'existe pas de valeur permettant d'exprimer l'inhibition. Il faut donc gérer la division par zéro dans un cas particulier, la fonction ne pouvant pas exprimer l'inhibition. Globalement, les calculs restent complexes et l'interprétation des résultats n'est pas évidente.

A.3 La somme

La fonction somme que j'ai testée est définie telle que $\phi'(alt) = \phi(alt) + \gamma_m(alt)$. Pour cette fonction, les intervalles sont plutôt évidents lorsque l'on travaille dans $\mathbb{R}$. Une valeur positive est une attraction, une valeur négative est une répulsion et le 0, qui est l'élément neutre de l'opération, exprime naturellement la neutralité de l'évaluation. Les calculs sont simples et les valeurs facilement compréhensibles. Toutefois, il n'est pas possible d'exprimer explicitement une inhibition dans une somme. Au mieux, il serait possible de voir l'inhibition comme la plus forte des répulsions possibles. En définissant une valeur spécifique pour l'inhibition. Néanmoins ne connaissant pas le nombre de motivations et donc le nombre d'éléments à sommer, cette forte répulsion ne peut assurer l'effet bloquant désiré par l'inhibition. De plus, si plusieurs actions sont inhibées, il est possible de voir des inhibitions plus ou moins fortes, ce qui n'a pas réellement de sens dans le modèle proposé.

A.4 La somme probabiliste

La fonction somme probabiliste que j'ai testée est définie telle que $\phi'(alt) = (\phi(alt) + \gamma_m(alt)) - (\phi(alt) * \gamma_m(alt))$. Cette fonction permet d'utiliser des valeurs d'évaluations $\gamma_m(alt) \in]-1,1[$. Les valeurs négatives représentent des répulsions, les valeurs positives des attractions et le 0 comme pour la somme est une valeur neutre. Néanmoins, il n'existe pas de valeur d'inhibition explicite pouvant bloquer l'évaluation d'une action.

A.5 Le produit

La fonction produit que j'ai testée est définie telle que $\phi'(alt) = \phi(alt) * \gamma_m(alt)$. Cette fonction est simple puisqu'elle se base sur l'opérateur de multiplication. Pour la multiplication, par définition, l'élément neutre est le 1 et l'élément inhibiteur est le 0 (l'élément absorbant de la multiplication). La valeur 0 inhibe les évaluations des autres motivations et permet

d'exprimer une inhibition. La valeur 1 ne change pas l'évaluation finale et permet d'exprimer une évaluation neutre. De plus, toutes les valeurs $\gamma_i(alt) > 1$ augmentent la valeur finale et toutes les valeurs v telles que $v \in]0;1[$ diminuent la valeur finale. Les valeurs négatives n'exprimant que le signe moins suivant la parité de leur nombre, n'apportent pas une bonne solution pour exprimer la répulsion. Le domaine serait donc $\mathbb{R}^+$.

La fonction produit répond aux propriétés définie dans la partie 3.3, c'est donc celle que j'ai choisie.

Annexe B

Combiner les préférences de l'agent

Contrairement aux motivations, les préférences ne sont pas indépendantes. Un agent exprime quatre types de préférences différentes pour une interaction.

- Soit l'agent aime effectuer une interaction.
- Soit l'agent n'aime pas effectuer une interaction.
- Soit l'agent a une inhibition pour cette interaction, cette inhibition devra être répercutée sur l'évaluation de toute l'alternative.
- Soit l'agent a une indifférence pour cette interaction, l'interaction n'influence donc pas l'évaluation de son alternative.

Afin de garder une certaine cohérence et ne pas perturber l'utilisateur entre le sens des valeurs des évaluations des motivations et le sens des valeurs des préférences des agents, j'ai fixé une sémantique pour les valeurs de préférence qui est proche de celle utilisée pour les valeurs des évaluateurs. Ainsi, pour un agent c, la fonction π_c donne pour chaque interaction a, la valeur de la préférence de l'agent $\pi_c(a)$ et ses valeurs sont incluses dans $\mathbb{R}^+$:

- $\pi_c(a) = 0$ exprime une inhibition pour l'interaction a : l'agent ne veut absolument pas effectuer l'interaction a,
- $\pi_c(a) = 1$ dénote que l'agent c exprime une indifférence pour l'interaction a,
- $\pi_c(a) \in]0, 1[$ désigne une préférence négative à l'exécution de l'interaction a,
- $\pi_c(a) > 1$ marque l'attirance de l'agent c pour l'exécution de a.

De par la dépendance des préférences, j'ai privilégié le fait d'obtenir une valeur "moyenne" pour la préférence d'une alternative. Toutefois, une action inhibitrice doit toujours inhibée l'alternative et une indifférence ne doit pas influencer la valeur finale.

Maintenant que le fonctionnement de la fonction de combinaison des préférences de l'agent pour une alternative est défini, je vais présenter quelques fonctions étudiées afin d'expliquer le choix de la fonction mis en place dans CoCoA. Ces fonctions doivent permettre d'avoir une vision globale de l'évaluation d'une alternative. Comme la préférence de l'agent évalue l'ensemble de l'alternative il est intéressant que la fonction choisie puisse aider à éviter les oscillations entre les alternatives en gardant une certaine stabilité dans l'évaluation de l'alternative pendant son exécution. Enfin, la fonction retenue doit être simple à comprendre et à mettre en place.

B.1 Maximum

Le maximum comme évaluation $\gamma_m(alt)$ correspond à la note maximale des préférences des interactions de alt. L'évaluation est simplifiée à l'extrême, donnant une vision optimiste de l'évaluation des préférences d'une alternative : $\gamma(alt) = \max_{i=1}^{|alt|}(\pi_c(a_i))$.
Toutefois la fonction maximum ne peut que décroître au cours de la simulation. Si la préférence pour une alternative précédemment sélectionnée décroît alors la possibilité de choisir une autre alternative augmente. Une combinaison des préférences basée sur le maximum peut ainsi favoriser les oscillations entre les différentes alternatives. De plus, si l'inhibition correspond toujours à une valeur répulsive extrême (comme la valeur 0), alors l'inhibition n'est pas pris en compte par la fonction. Enfin, l'évaluation d'une alternative par le maximum ne se base que sur une seule préférence de l'alternative, ce qui ne reflète pas vraiment une évaluation globale de l'alternative.

B.2 Minimum

La fonction minimum (opposée du maximum) $\gamma_m(alt)$ correspond à la note minimale des actions de l'alternative. C'est une vision pessimiste de l'évaluation d'une alternative qui ne se base que sur une seule interaction :$\gamma(alt) = \min_{i=1}^{|alt|}(\pi_c(a_i))$.
Contrairement au maximum, au cours de l'exécution de l'alternative, le minimum ne peut qu'augmenter (sauf si l'agent découvre de nouvelles informations et qu'il doit replanifier et modifier l'alternative). Cet accroissement des préférences permet d'aider à éviter les oscillations entre les différentes alternatives. De plus, l'inhibition est prise en compte (si une interaction a une préférence de 0, la valeur minimale de son alternative est donc 0 et l'évaluation de la préférence sera également de 0). Bien que plus avantageux que le maximum sur de nombreux points, le minimum ne se base que sur une seule interaction (celle avec la préférence la plus faible) ce qui ne reflète pas vraiment l'idée de préférence globale d'une alternative que l'on recherche.

B.3 N^{ieme} plus petite préférence

L'un des défauts de la fonction minimum est cette vision pessimiste qui ne reflète pas vraiment l'évaluation globale de l'alternative. L'idée est donc de prendre en compte la n^{ieme} note minimale[19] d'une alternative où n correspond à une notion d'erreur, un peu comme la notion de bruit en statistique. Soit $mini$ un sous-ensemble des $a_i \subset alt$, tel que $|mini| = n-1$ alors,

$$\gamma(alt) = \pi_c(min_n)| \quad \begin{array}{l} \forall a_i \in mini, \pi_c(min_n) \geq \pi_c(a_i) \\ \forall a_j \in alt \setminus (mini \cup min_n), \pi_c(min_n) \geq \pi_c(a_j) \end{array}$$

Ceci permettrait d'obtenir une vision pessimiste dégradée de l'évaluation d'une alternative. Globalement, cette idée garde la stabilité offerte par le minimum tout en enlevant le bruit. Pour l'inhibition deux choix sont possibles : soit avoir une inhibition effective sur n inhibitions, soit gérer une inhibition comme un cas exceptionnel et le prendre en compte indépendamment

[19] Si $n = 1$ alors cette fonction revient à un minimum classique

de la fonction. De plus, il faut définir la valeur n et se poser la question du choix du meilleur n. Ensuite, il faut un traitement particulier pour les alternatives contenant moins de n interactions. Enfin, il faut gérer le cas particulier de l'inhibition avec cette valeur n. Bien que cette fonction permette de réduire l'aspect pessimiste du minimum, elle engendre globalement plus de questions et de cas particuliers à gérer que la fonction minimum classique.

B.4 Moyenne arithmétique

L'évaluation $\gamma(alt)$ correspond à la moyenne arithmétique des préférences des interactions de l'alternative. Privilégiant une évaluation globale des préférences de l'alternative : $\gamma_m(alt) = \sum_{i=1}^{|alt|}(\frac{\pi_c(a_i)}{|alt|})$.
Par définition cette fonction, comme toutes les fonctions moyennes, permet de réduire l'impact (de lisser) des valeurs extrêmes de répulsions et d'attractions. Ce lissage augmentant avec le nombre d'interactions, permet d'assurer une certaine stabilité pendant l'exécution de l'alternative dans la notation des préférences. Toutefois, le lissage et la stabilité ne sont pas garantis avec peu d'actions, il est évidemment possible d'avoir une évaluation de préférence qui décroît avec l'exécution de l'alternative. L'inhibition demande une gestion particulière pour être pris en compte par cette fonction.

B.5 Moyenne des N minima

Cette fonction vient de l'idée de garder les avantages du minimum avec ceux proposés par la moyenne arithmétique. Ainsi l'évaluation $\gamma(alt)$ correspond à la moyenne des n plus petites préférences des interactions de l'alternative. L'évaluation se base sur n interactions, apportant une vision générale pessimiste[20]. Soit min_n l'interaction ayant reçu la n^{ieme} plus petite préférence. Alors $\gamma_m(alt) = \sum_{i=1}^{n}(\frac{\pi_c(min_i)}{n})$.

Cette fonction possède une stabilité[21] proche de celle offerte par le minimum (la stabilité dans l'évolution de la valeur permet d'éviter les oscillations entre les meilleures alternatives pour la motivation des préférences de l'agent). L'évaluation se fait sur un sous-ensemble d'interactions permettant d'avoir une vision plus générale sans être pour autant globale. Toutefois, l'inhibition est un cas particulier à gérer, la stabilité est plus faible que le minimum et l'évaluation moins globale qu'une moyenne. Enfin, la valeur n reste à fixer soit de manière statique, soit proportionnellement à la taille des alternatives à évaluer. Globalement, cette fonction possède les avantages de la fonction minimum et d'une fonction moyenne mais dans une version dégradée. Par contre, elle possède les inconvénients liés à la gestion de l'inhibition et au choix de la valeur n.

B.6 Moyenne Géométrique

La moyenne géométrique est utilisée en économie pour calculer un taux d'accroissement moyen, elle est également utilisée en économétrie pour calculer le taux de rendement géomé-

[20] Si $n = 1$, la fonction correspond au minimum
[21] La stabilité dans l'évolution de l'évaluation pendant l'exécution de l'alternative

trique moyen ou le rendement moyen pondéré par le temps. La moyenne géométrique correspond à la racine n-ième du produit des notes des n actions : $\gamma(alt) = \sqrt[|alt|]{\prod_{i=1}^{|alt|} \pi_c(a_i)}$. La moyenne géométrique tente de trouver un coefficient multiplicateur commun entre toutes les valeurs. Ainsi contrairement à la moyenne arithmétique qui surestime l'effet des grands nombres par rapport aux petits, la moyenne géométrique va chercher un coefficient k tel que les valeurs importantes surestiment d'un multiple de k et les petits nombres sous-estiment avec un multiple de $\frac{1}{k}$. De plus, le fait de passer par un produit implique la prise en compte d'une inhibition dont la valeur est 0 pour toute l'alternative. Comme toutes les fonctions moyennes, la moyenne géométrique prend en compte l'ensemble des préférences des interactions de l'alternative et offre un lissage intéressant pour garder une certaine stabilité pendant l'exécution de l'alternative. Réduisant les inconvénients de la moyenne arithmétique cette fonction est très intéressante.

B.7 Moyenne Harmonique

La moyenne harmonique est l'inverse de la moyenne arithmétique de l'inverse des a_i. Elle est notamment utilisée pour calculer une vitesse moyenne d'un déplacement qui possède plusieurs vitesses constantes entre des points équidistants. De même, elle est utilisée en électronique pour calculer la résistance moyenne de résistances montées en parallèle dans un circuit électrique. La résistance moyenne permet de connaître la valeur unique à attribuer à chaque résistance montée en parallèle afin de préserver la résistance totale du circuit. La moyenne harmonique correspond au quotient entre n et la somme des inverses des préférences des n actions : $\gamma(alt) = \frac{|alt|}{\sum_{i=1}^{|alt|}(\frac{1}{\pi_c(a_i)})}$. Afin d'éviter la division par zéro ($\frac{1}{0} = \infty$), une inhibition demande un traitement particulier. Toutefois l'utilisation de la moyenne harmonique dans un circuit électrique pour connaitre la valeur unique à attribuer à chaque résistance est proche de l'idée que l'on souhaite obtenir avec les préférences. Les préférences étant définies indépendamment, elles correspondraient aux résistances montées en parallèle et la valeur unique à attribuer à chaque résistance du circuit correspondrait à la valeur unique à affecter à chaque préférence de l'alternative pour préserver l'évaluation.

B.8 Bilan

La moyenne harmonique et la moyenne géométrique sont deux fonctions de combinaison des préférences équivalentes dans leur avantages, c'est pourquoi j'ai retenu ces deux fonctions. Par défaut, c'est la moyenne harmonique qui est utilisée pour combiner les préférences de l'agent. Mais, l'utilisateur peut changer cette fonction et utiliser la moyenne géométrique. Le choix de la fonction par défaut a été fait par rapport à l'utilisation de la moyenne harmonique dans un circuit électrique qui peut être comparée à la combinaison des préférences d'une alternative.

Annexe C

L'atelier de conception

Dans cette annexe, je présente l'atelier de conception en détail. Sans pour autant être une documentation utilisateur ou un tutoriel pas à pas de l'atelier, ce chapitre permet de comprendre plus en profondeur le fonctionnement de l'interface de l'atelier et ses différents atouts.

C.1 L'interface

L'atelier repose sur trois éléments du comportement : les propriétés, les interactions et les motivations. Le principe de l'atelier est donc de permettre à l'utilisateur de concevoir chaque élément indépendamment tout en gardant une vue globale sur le comportement. C'est pourquoi nous avons choisi de mettre en place une vue globale du comportement qui est présente à chaque étape de la conception (une vue fixe). Chaque élément du comportement correspond à une étape spécifique de la conception. C'est pourquoi ces trois étapes ne sont pas représentées simultanément. Néanmoins, les changements apparaissant dans une étape peuvent avoir des influences sur les autres (par exemple l'ajout de propriétés permet de les utiliser par la suite comme paramètres).

C.1.1 Vue générale

Lorsque l'atelier est lancé, la première chose que l'utilisateur doit faire c'est nommer l'agent dont il va concevoir le comportement (voir la figure C.1). Ce nom a principalement pour but de différencier les comportements pour l'utilisateur. C'est pourquoi on peut également considérer qu'il s'agit du nom du comportement. Une fois le nom fourni, l'atelier se présente dans une fenêtre composée de trois parties :

- le menu permettant de charger et de sauvegarder le comportement
- l'arbre récapitulatif du comportement
- la fenêtre de gestion qui offre trois vues : les propriétés, les interactions et les motivations

Comme nous l'avons déjà vu, le menu et l'arbre récapitulatif du comportement sont toujours visibles quelque soit la partie du comportement qui est gérée. À tout moment de la conception, l'utilisateur peut ainsi consulter l'état global du comportement et sauvegarder ce

FIG. C.1 – L'utilisateur définit dans un premier temps, le nom de l'agent qu'il va créer.

qui a été fait. Les fenêtres de gestion (propriétés, interactions et motivations) sont affichables une à une (dans le même espace) par simple clic sur l'onglet correspondant (voir la figure C.2).

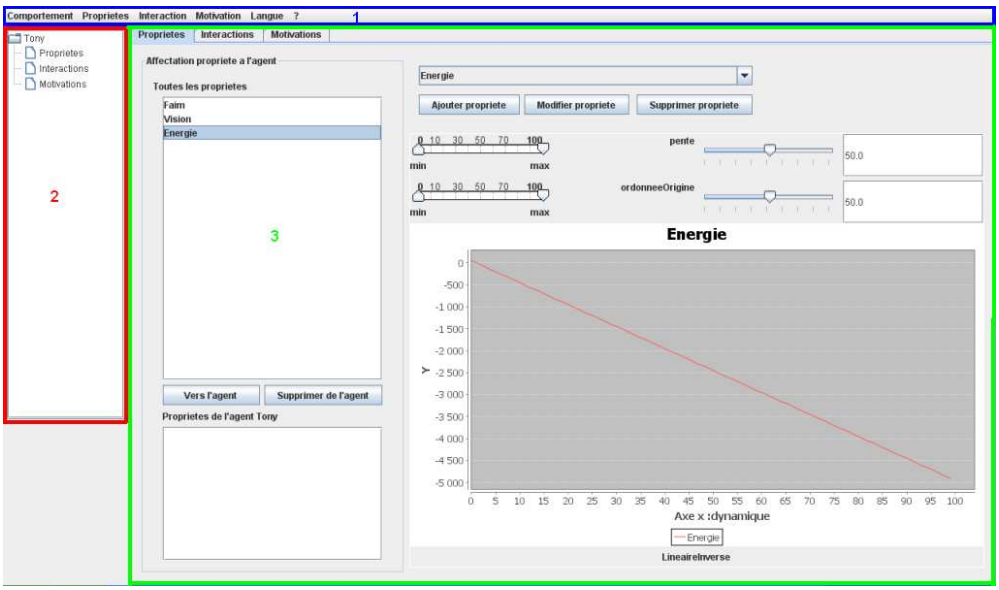


FIG. C.2 – L'interface se décompose en trois parties. Le menu (partie 1 en bleu), l'arbre récapitulatif (partie 2 en rouge) et la fenêtre de gestion (partie 3 en vert) qui gère soit les propriétés, soit les interactions, soit les motivations. Les parties 1 et 2 sont fixes, la partie 3 dépend de l'onglet sélectionné.

C.1.2 Le menu

Le menu permet de gérer principalement le chargement de comportements existants et la sauvegarde du comportement au cours. Il permet également d'importer et de sauvegarder indépendamment chacune des trois parties : les propriétés, les motivations et les interactions. Enfin, il permet de changer la langue de l'atelier sans redémarrage (voir la figure C.3).

FIG. C.3 – La barre de menu de l'atelier qui permet de sauvegarder et charger tout ou partie du comportement.

C.1.3 L'arbre récapitulatif du comportement

La définition du comportement faite avec l'atelier de conception est récapitulée dans un arbre (voir la partie 2 de la figure C.2). À la racine se trouve le nom de l'agent, les feuilles présentent les propriétés, les interactions et les motivations définies pour l'agent.

C.1.4 Les propriétés

La gestion des propriétés permet à la fois de créer des propriétés et de les affecter à l'agent (voir la figure C.4). Une propriété est dynamique ou dépendante (voir la partie 4.1). Si elle est dynamique, la propriété évolue suivant une fonction mathématique. Si la propriété est dépendante, elle dépend d'une propriété existante et la dépendance est gérée par une fonction mathématique.

L'ajout et la modification d'une propriété s'effectue par la fenêtre présentée dans la figure C.5. C'est dans cette fenêtre que l'utilisateur définit si la propriété est dynamique ou dépendante d'une autre propriété, ainsi que la fonction correspondante. La liste des fonctions est définie dans un fichier de configuration (voir la partie C.3). Chaque fonction possède un nom, une description (qui permet un affichage d'aide via une info bulle), les paramètres qui lui sont nécessaires et la classe java calculant l'allure de la fonction.

C.1.5 Les interactions

La gestion des interactions permet d'affecter les interactions que peut effectuer l'agent, de leur donner une valeur de préférence et de coûts et également de créer des groupes de coûts et de préférences pour simplifier le paramétrage (voir la figure C.4). Les interactions que peut effectuer l'agent et les groupes peuvent être importées d'un projet CoCoA existant où à partir d'une sauvegarde spécifique aux interactions d'un comportement (voir la partie C.2).

Chaque interaction reçoit une valeur de préférence et de coût. L'affectation des valeurs de préférence et de coût se fait de la même manière. Je présenterai donc les fenêtres pour l'affectation des préférences, car les valeurs expriment un goût (attraction, neutralité, répulsion et inhibition) qui est représenté par des couleurs qui possèdent une sémantique (les couleurs sont configurables dans par un fichier voir la partie C.2).

La partie droite de la fenêtre de gestion des interactions est suffisante pour définir les capacités de l'agent et ses préférences (voir la figure C.7). La partie gauche de cette fenêtre permet d'utiliser la notion de groupe pour simplifier le paramétrage (voir la figure C.8). La

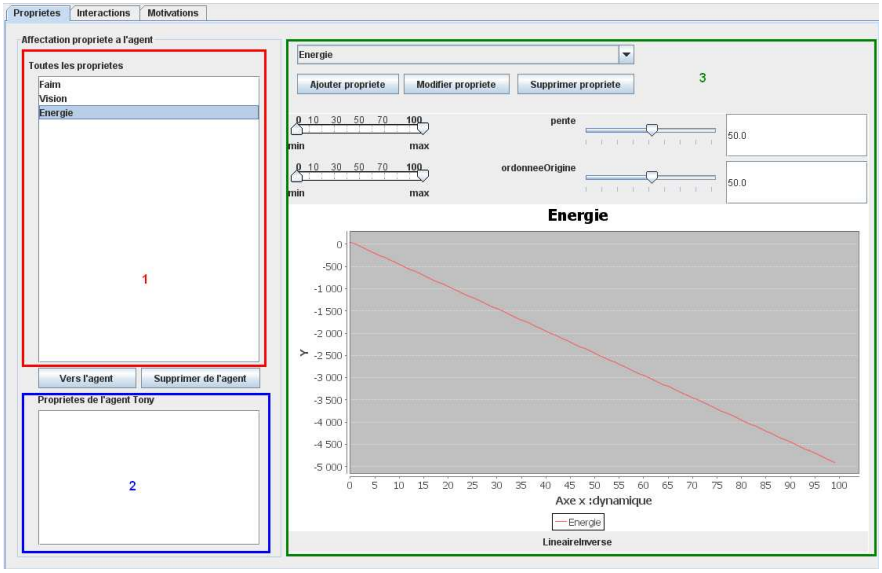


FIG. C.4 – La gestion des propriétés se décompose en trois parties. Dans la partie 1 (en rouge) se trouve l'ensemble des propriétés qui sont disponibles (préalablement créées ou chargées). Dans la partie 2 (en bleu) se trouve les propriétés de l'agent, entre la partie 1 et la partie 2 deux boutons permettent d'ajouter ou de supprimer une propriété à l'agent. La partie 3 (en vert) donne une représentation graphique de l'évolution de la propriété suivant les paramètres de la fonction d'évolution. Cette partie permet également de créer, de modifier ou de supprimer une propriété.

gestion d'un groupe se fait par le bouton + situé à coté de chaque groupe. Il permet de changer la vue pour se focaliser sur le contenu d'un groupe pour y ajouter des interactions où pour spécifier les valeurs de paramètres des interactions de ce groupe (voir la figure C.9).

C.1.6 Les motivations

La gestion des motivations permet de définir les motivations utilisées par le mécanisme de sélection d'action et leur paramétrage (profil d'individualité) (voir la figure C.10). L'idée principale de cette fenêtre est de proposer une partie standard commune à toutes les motivations et une partie spécifique pour chaque motivation. En effet, pour mieux comprendre le fonctionnement d'une motivation il faut pouvoir apporter une illustration la plus précise possible. La figure C.10, nous montre l'affichage spécifique de l'opportunisme (le carré bleu correspondant à la valeur de l'opportunisme) par rapport à la valeur de la propriété vision de l'agent (la distance maximale de perception de l'agent). Ainsi par rapport à son champ de vision, il est possible de se faire une idée du seuil d'opportunisme (la distance à partir de laquelle l'opportunisme favorise une action exécutable).

Pour la motivation des préférences de l'agent (voir la figure C.11, l'affichage spécifique permet de simuler la combinaison des préférences avec les valeurs de préférences affectées via l'onglet `Interactions`.

Dans le cas où le paramètre d'une motivation est lié à une propriété, l'affichage spécifique reprend les principes de la fenêtre de gestion des propriétés (voir la figure C.12).

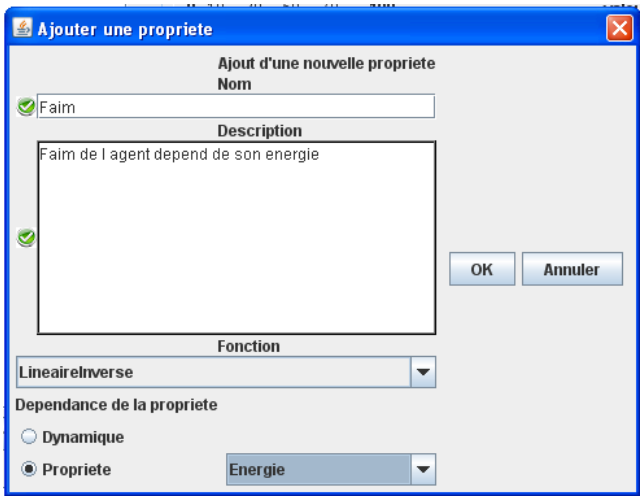


FIG. C.5 – L'ajout et la modification d'une propriété s'effectue par cette fenêtre. Une propriété possède un nom, une description (qui s'affichera en info-bulle), une fonction et le type de la propriété (dynamique ou dépendante d'une autre propriété). La liste des fonctions est définie par rapport à un fichier de configuration, la liste des propriétés sont les propriétés présentes dans la partie 1 de la figure C.4.

C.2 Les fichiers définissant le comportement

Lors de la sauvegarde du comportement, quatre fichiers sont générés automatiquement, un fichier par partie du comportement (propriétés, interactions et motivations) et un fichier qui englobe toutes les parties. Ceci permet de réutiliser tout un comportement ou simplement une partie. Il est également possible qu'un utilisateur veuille ne créer qu'une partie du comportement et la sauvegarder. Cette décomposition en quatre fichiers permet une conception modulaire, car il est possible de charger des données provenant des différents fichiers générés.

C.2.1 Le comportement

Le fichier de comportement (dont le nom est suffixé par `-behavior.xml`) définit l'ensemble des données nécessaires à la construction d'un comportement pour un agent CoCoA (voir la figure C.13). Un comportement est composé d'un nom (le nom de l'agent), d'une description, de propriétés, de motivations et d'interactions. Une propriété est composée d'un nom, d'une description et d'une fonction (en cas d'absence la fonction est considérée comme une fonction constante). La valeur d'une propriété au début de la simulation est spécifique à la simulation, c'est pourquoi elle n'a pas été définie dans l'atelier de conception. Une fonction est composée d'un nom, de paramètres et d'une fonction d'évolution qui est soit dynamique, soit dépendante d'une autre propriété. Les paramètres sont des valeurs qui peuvent être bornées. Une motivation est composée d'un nom, d'une valeur de paramètre (pour le profil d'individualité) et d'une fonction (dans le cas d'une dépendance à une propriété). Une interaction est composée d'un nom, d'une description, d'une valeur de préférence et d'une valeur de coût. Les groupes n'apparaissent pas dans ce fichier généré car ils ne sont qu'une simplification de paramétrage.

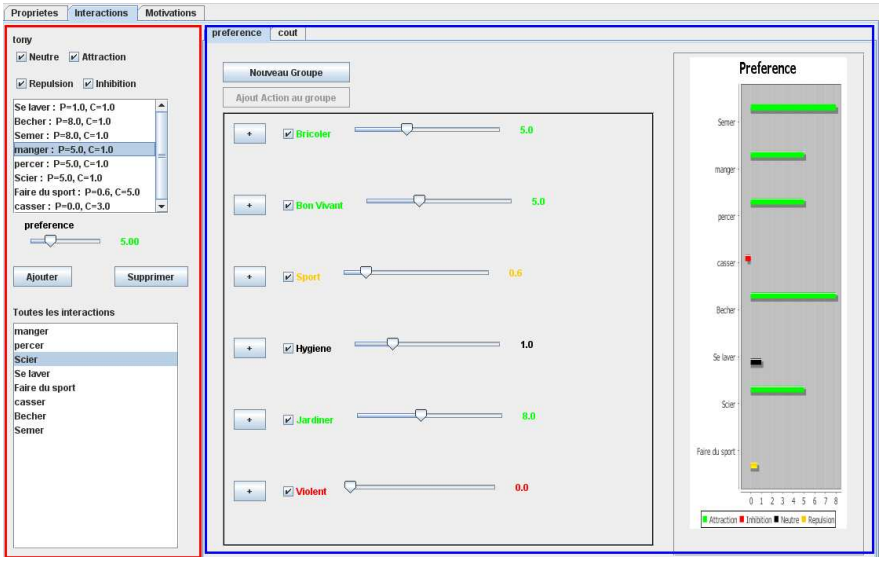


FIG. C.6 – La gestion des interactions se décompose en deux parties. La partie gauche (en rouge) gère l'affectation des interactions et la valeur de préférence et de coût pour chaque interaction. La partie de droite (en bleu) permet de créer les groupes de préférence et de coût, d'affecter les interactions par groupe (en passant à la vue par groupe avec le bouton +), d'affecter les valeurs pour chaque groupe et de visualiser (et comparer) les valeurs affectées pour chaque interaction. Dans cette partie, il est également possible de se focaliser soit sur les préférences, soit sur les coûts, via les onglets.

C.2.2 Les propriétés

Le fichier des propriétés (dont le nom est suffixé par `-property.xml`) regroupe l'ensemble des propriétés définies (voir la figure C.14). La définition de la propriété est la même que dans le fichier de comportement.

C.2.3 Les motivations

Le fichier des motivations (dont le nom est suffixé par `-motivations.xml`) regroupe l'ensemble des motivations définies (voir la figure C.15). La définition d'une motivation est la même que dans le fichier de comportement.

C.2.4 Les interactions

Le fichier des interactions (dont le nom est suffixé par `-interactions.xml`) regroupe l'ensemble des interactions, leur groupes et leur préférence et leur coût (voir la figure C.16). Contrairement au fichier de comportement, ce fichier prend en compte les groupes afin de permettre la récupération du travail effectué. De plus, les interactions mémorisées correspondent à toutes les interactions (pas seulement les interactions que l'agent peut-effectuer) qui étaient dans la liste de la fenêtre de gestion. Cette liste d'interactions peut être également importée à partir de n'importe quel fichier provenant de CoCoA et contenant des interactions. Ce fichier

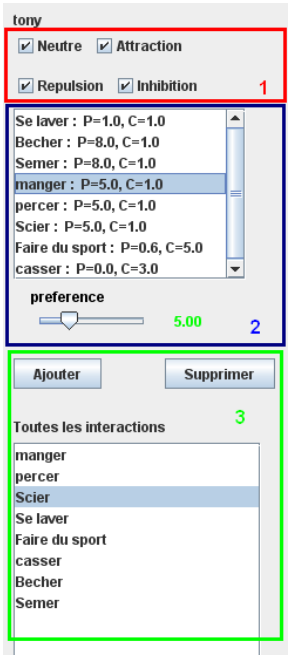


FIG. C.7 – Cette partie de la gestion des interactions permet d'affecter les interactions à l'agent (partie 3 en vert) et de gérer une par une les valeurs de préférence et de coût des interactions (partie 2 en bleu). Nous avons également mis en place un système de filtre par rapport aux valeurs des interactions (partie 1 en rouge). Ce système permet d'afficher les interactions dont la valeur est neutre, attractive, répulsive ou inhibitrice afin de simplifier la recherche et donc l'affectation.

peut être soit un fichier provenant d'une simulation particulière, soit un fichier de la librairie des interactions de CoCoA. Les interactions que l'agent ne peut pas effectuer ont, par défaut, un coût et une préférence de 1.

C.3 Les fichiers de configuration

Comme nous l'avons vu, l'atelier de conception est configurable via des fichiers. Il existe cinq fichiers de configuration ayant chacun un rôle bien précis.

color.config : ce fichier définit les différentes couleurs utilisées.

function.config : ce fichier définit les différentes fonctions.

slider.config : ce fichier définit le comportement des sliders (les barres qui fixent une valeur).

motivation.config : la fenêtre de gestion des interactions.

fr.properties : ce fichier gère le texte pour chaque mot présent dans l'atelier qui ne vient pas de l'utilisateur. Le fichier `en.properties` est le même fichier mais pour la traduction anglaise.

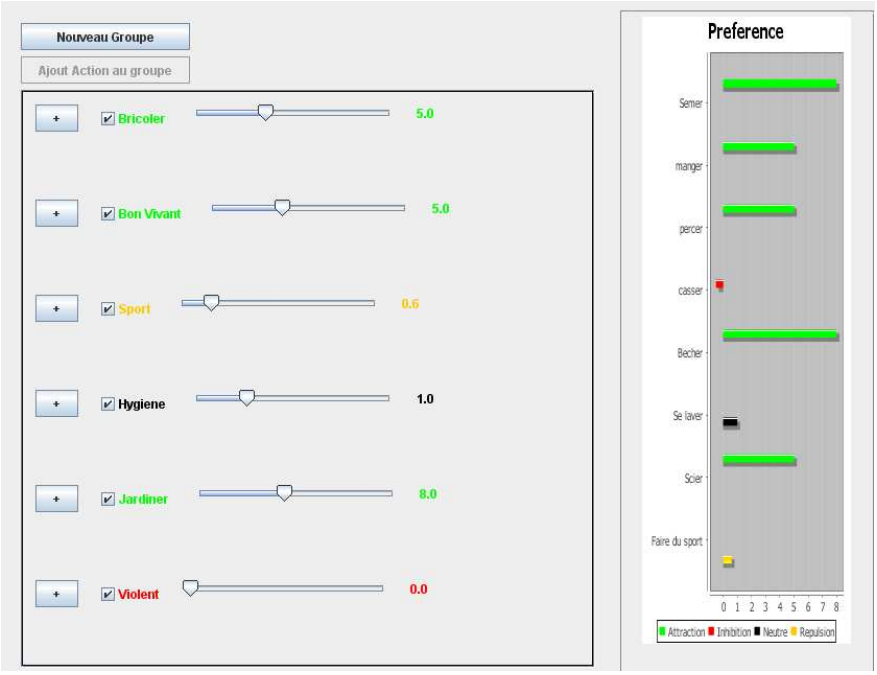


FIG. C.8 – Cette partie permet d'affecter les valeurs sur les interactions via la notion de groupe. Elle affiche également les interactions dans le graphique à gauche (les interactions cochées) afin d'avoir une vue d'ensemble des différentes valeurs. Enfin, elle permet également de créer un groupe. Chaque groupe peut être géré dans une vue spécifique (voir la figure C.9) qui est accessible via le bouton +.

C.3.1 Le fichier color.config

Ce fichier associe à chaque "thème", une couleur. Il est principalement utilisé pour définir les couleurs permettant d'exprimer l'attraction, l'inhibition, la répulsion et la neutralité. Chaque couleur est représentée par sa définition RGB (voir la figure C.17). Un thème est un nom (un identifiant) associé à une couleur.

C.3.2 Le fichier function.config

Ce fichier définit pour chaque fonction, son nom, ses paramètres et une classe java permettant l'application de la fonction. Chaque paramètre est défini par un nom (un identifiant) et une description (voir la figure C.18). Les bornes des paramètres sont définies par l'utilisateur.

C.3.3 Le fichier slider.config

Ce fichier permet de définir des comportements pour chaque slider (voir la figure C.19). Le comportement d'un slider est défini par une valeur minimale, une valeur maximale des paliers et la valeur d'un pas pour chaque palier. Ainsi, il est possible de définir des paliers de valeurs entre lesquelles un pas du slider aura une valeur spécifique. Par exemple, on peut définir un

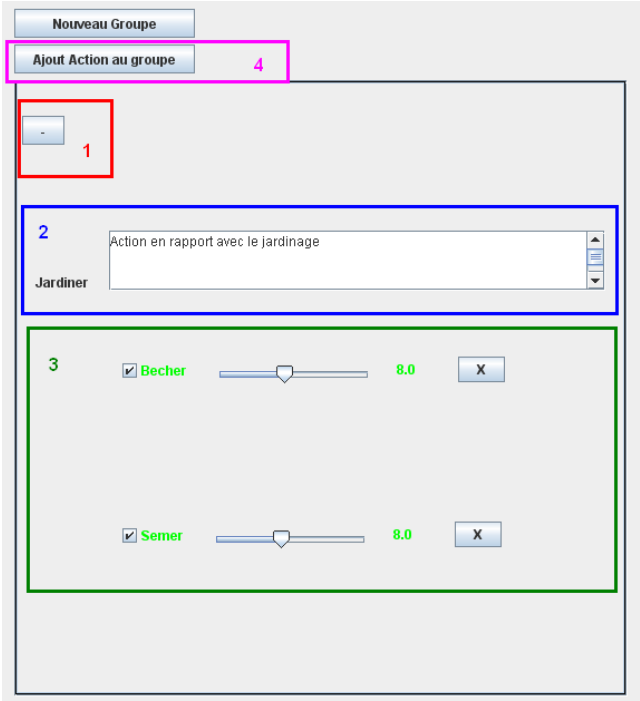


FIG. C.9 – Cette vue d'un groupe permet d'ajouter des interactions au groupe (partie 4 en rose) parmi l'ensemble des interactions que peut effectuer l'agent. Un groupe a un nom et une description, cette dernière peut être modifiée (partie 2 en bleu). Il est possible de spécifier une valeur pour chaque interaction du groupe (dans la partie 3). Enfin le bouton - (partie 1 en rouge) permet de revenir à la vue par groupe présentée dans la figure C.8).

slider dans lequel entre 0 et 1 le pas vaut 0.01, entre 1 et 10 le pas vaut 0.1 et entre 10 et 100 le pas vaut 1.

C.3.4 Le fichier motivation.config

Ce fichier permet de spécifier pour chaque motivation, le type de paramètre accepté, ainsi que son affichage spécifique (voir la figure C.20). Pour cela, la partie fixe est un choix entre les types : `Integer, Float, Empty`. Les valeurs entières (`Integer`) sont utilisées par exemple pour l'opportunisme et les accomplissements. Les valeurs flottantes (`Float`) sont utilisées pour la revalorisation multi-buts et l'inertie. Les valeurs vides (`Empty`) sont utilisées pour les préférences de l'agent (dont l'onglet `Interactions` permet de définir la valeur de préférences pour chaque interaction) et pour l'influence des buts, qui ne nécessite pas de paramètre. Ce fichier permet également de définir la description d'une motivation, afin que cette information apparaisse en info-bulle pendant la conception. La partie spécifique de la motivation détermine une classe Java qui implémente une interface permettant de donner un rendu spécifique pour une motivation.

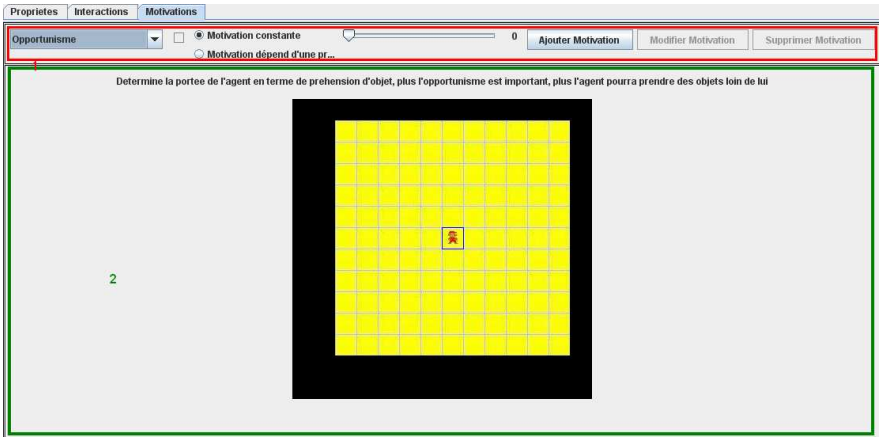


FIG. C.10 – La gestion des motivations se décompose en deux. La première partie (en rouge) est un affichage fixe dont les valeurs dépendent d'un fichier de configuration. Cet affichage est composé d'une liste permettant le choix de la motivation à traiter, de la définition du paramètre de la motivation (constante ou valeur dépendante d'une propriété) et de boutons permettant l'ajout, la modification (du paramétrage) et la suppression de la motivation pour l'agent. La deuxième partie (en vert) est un affichage spécifique à la motivation. Cet affichage spécifique, comme la liste des motivations, est configurable via un fichier (voir la partie C.2).

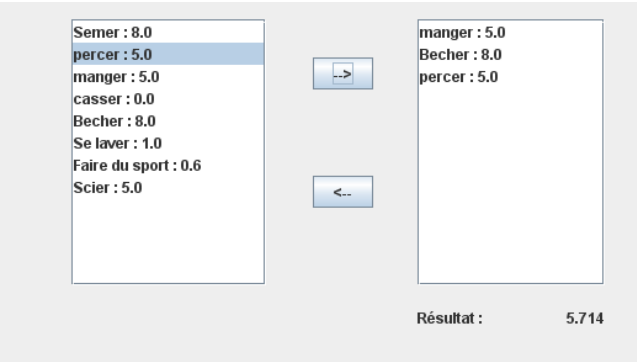


FIG. C.11 – L'affichage spécifique pour la motivation des préférences de l'agent, permet de simuler le calcul de la motivation en choisissant des interactions parmi les interactions que l'agent peut effectuer. Les valeurs de préférences contiennent les valeurs définies par l'utilisateur via l'onglet `Interactions`. Cette affichage spécifique permet donc également de voir l'impact des valeurs de préférences choisies.

C.3.5 Les fichiers de la langue fr.properties et en.properties

Les fichiers de langue sont des fichiers `properties` classiques, qui associent à un label une valeur.

C.4 Conclusion

L'atelier de conception de comportement a été mis en place dans le but de définir les comportements des agents indépendamment d'une simulation ou de CoCoA. Les informations

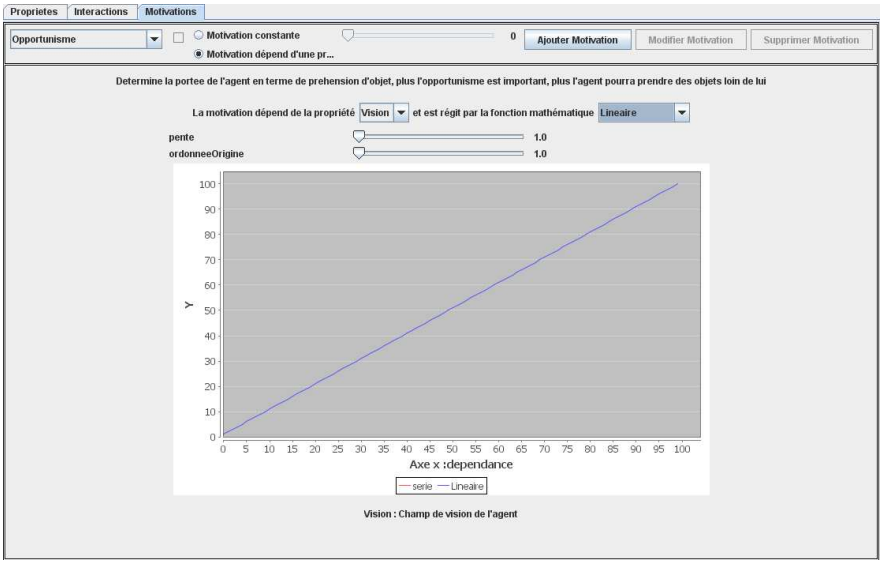


FIG. C.12 – L'affichage spécifique pour un paramétrage dépendant d'une propriété de l'agent reprend l'affichage de la gestion d'une propriété afin d'avoir une vue globale de l'évolution de la propriété.

```
<behavior>          ::=  <name>, <description>, <property>*, <motivation>*, <interaction>*
<name>              ::=  Symbol
<description>       ::=  Symbol
<property>          ::=  <property-name>, <description>, <function>?
<property-name>     ::=  Symbol
<value>             ::=  Value
<function>          ::=  <name>, <parameter>*, <evolution>
<parameter>         ::=  <name>, <value>, <min>?, <max>?
<evolution>         ::=  "Dynamic" | <property-name>
<motivation>        ::=  <motivation-name>, <parameter-value>, <function>?
<motivation-name>   ::=  Symbol
<parameter-value>   ::=  Symbol
<function>          ::=  <name>, <parameter>*, <evolution>
<parameter>         ::=  <name>, <value>, <min>?, <max>?
<min>               ::=  Value
<max>               ::=  Value
<evolution>         ::=  "Dynamic" | <property-name>
<interaction>       ::=  <name>, <description>, <agent-taste>, <cost>
<agent-taste>       ::=  Value
<cost>              ::=  Value
```

FIG. C.13 – Représentation d'un comportement, ce dernier est composé d'un nom, d'une description, d'un ensemble de propriétés, de motivations et d'interactions. Une propriété a une valeur constance ou évoluant suivant une fonction dont la valeur peut elle-même dépendre d'une autre propriété. Les motivations sont paramétrables par une valeur fixe ou par une valeur dépendant d'une fonction ou encore par une valeur dépendante d'une propriété (la relation de dépendance étant fixée par une fonction). Enfin, les interactions ont pour ce comportement une valeur de préférence et une valeur de coût.

```
<properties-file>  ::=  <property>*
<property>         ::=  <property-name>, <description>, <value>, <function>?
<property-name>    ::=  Symbol
<value>            ::=  Value
<function>         ::=  <name>, <parameter>*, <evolution>
<parameter>        ::=  <name>, <value>, <min>?, <max>?
<min>              ::=  Value
<max>              ::=  Value
<evolution>        ::=  "Dynamic" | <property-name>
```

FIG. C.14 – Représentation d'un fichier généré lors de la création des propriétés d'un comportement. Ce fichier peut être chargé pour inclure un certain nombre de propriétés existantes dans un nouveau comportement.

```
<motivations-file>  ::=  <motivation>*
<motivation>        ::=  <motivation-name>, <parameter-value>, <function>?
<motivation-name>   ::=  Symbol
<parameter-value>   ::=  Symbol
<function>          ::=  <name>, <parameter>*, <evolution>
<parameter>         ::=  <name>, <value>, <min>?, <max>?
<min>               ::=  Value
<max>               ::=  Value
<evolution>         ::=  "Dynamic" | <property-name>
```

FIG. C.15 – Représentation d'un fichier généré lors de la création des motivations d'un comportement. Ce fichier peut être chargé pour inclure un certain nombre de motivations existantes ou reprendre des profils d'individualité existant.

```
<interactions-file>  ::=  <interaction>*, <action-group>*, <cost-group>*
<interaction>        ::=  <name>, <description>, <action-groups>, <cost-groups>
<name>               ::=  Symbol
<description>        ::=  Symbol
<action-groups>      ::=  <action-group-name>*
<cost-groups>        ::=  <cost-group-name>*
<action-group-name>  ::=  Symbol
<cost-group-name>    ::=  Symbol
<action-group>       ::=  <action-group-name>, <description>
<cost-group>         ::=  <cost-group-name>, <description>
```

FIG. C.16 – Représentation d'un fichier généré lors de la création des groupes d'interactions. Ce fichier peut être chargé pour inclure un certain nombre de groupes existants dans un nouveau comportement.

```
<color.config>  ::=  <theme>*
<theme>         ::=  <name>, <red-value>, <green-value>, <blue-value>
<name>          ::=  Symbol
<red-value>     ::=  Value
<green-value>   ::=  Value
<blue-value>    ::=  Value
```

FIG. C.17 – Représentation du fichier color.config.

```
<function.config>  ::=  <function>*
<function>         ::=  <name>, <class-name>,<param>*
<name>             ::=  Symbol
<class-name>       ::=  Symbol
<param>            ::=  <param-name> <description>
<param-name>       ::=  Symbol
<description>      ::=  Symbol
```

FIG. C.18 – Représentation du fichier function.config.

```
<slider.config>  ::=  <slider>*
<slider>         ::=  <name>, <min>, <max>, <borne>*
<name>           ::=  Symbol
<min>            ::=  Value
<max>            ::=  Value
<borne>          ::=  <upper-limit>, <step>
<upper-limit>    ::=  Value
<step>           ::=  Value
```

FIG. C.19 – Représentation du fichier slider.config.

fournies par les fichiers générés viendront complétées celles créées par CoCoA (voir la partie 2.4). L'atelier permet de construire des propriétés dynamiques et dépendantes, de définir les capacités de l'agent (ainsi que les préférences et les coûts de ses interactions), de choisir les motivations qui vont influencer le mécanisme de sélection d'action et de concevoir le profil d'individualité de l'agent. Cet atelier est configurable via cinq fichiers de configuration qui permettent de modifier les couleurs qui ont un sens particulier dans l'interface (l'attraction, la neutralité, la répulsion et l'inhibition), de gérer le comportement des sliders, de prendre en compte un affichage fixe et un affiche spécifique pour chaque motivation, de définir les fonctions mathématiques qui seront utilisées et également de changer de langue. Enfin, cet atelier de conception, génère trois différents fichiers correspondant aux trois parties de la définition du comportement, ainsi qu'un fichier regroupant tout le comportement. Les données de ces fichiers sont chargeables par l'atelier (et donc réutilisables) ce qui rend la conception modulaire. Il est ainsi possible de ne construire qu'une partie du comportement ou même simplement définir les groupes d'interactions qui vont simplifier une future conception.

```
<motivation.config>  ::=  <motivation>*
<motivation>         ::=  <name>, <description>, <fixed-part>, <special-part>
<name>               ::=  Symbol
<description>        ::=  Symbol
<fixed-part>         ::=  'Integer'|'Float'|'Empty'
<special-part>       ::=  Symbol
```

FIG. C.20 – Représentation du fichier motivation.config

Lexique

Animats : anima-materials
ASM : Action Selection Mechanism
BDI : Believe Desire Intention
CoCoA : Cognitive Collaborative Agents
CRPG : Computer Role-Playing Game
CTF : Capture The Flag
F.E.A.R. : First Encounter Assault Recon
FPS : First Person Shooter
FSM : Finite State Machines
HCS : Hierarchical Classifier System
HFSM : Hierarchical Finite State Machines
IA : Intelligence Artificielle
MHiCS : Motivational and Hierarchical with Classifier Systems
MIT : Massachusetts Institute of Technology
MMORPG : Massively Multiplayer Online Role Playing Game
MUD : Multi-User-Dungeons
PECS : Physical conditions Emotional state Cognitive capabilities Social status
PNJ : Personnage Non-Joueur
SMA : Systèmes Multi-Agents
TFC : Team Fortress Classic
TMS : Truth Maintenance System
WoW : World of Warcraft

Résumé : Cette thèse est une proposition pour la conception de comportements crédibles dans les simulations informatiques pour agents situés dans un environnement virtuel. Le comportement d'un agent se définit à partir de l'observation des actions qu'il exécute dans un environnement. Chaque action exécutée résulte d'un choix de l'agent parmi l'ensemble des actions qu'il peut effectuer. Le comportement se construit donc à partir des capacités de l'agent lui permettant de résoudre ses buts, le raisonnement, et d'un choix influencé par ses traits de caractère, l'individualité. Ma contribution consiste à définir des comportements qui sont composés de ces deux parties : *le raisonnement* et *l'individualité*.

Cette thèse se concentre sur la proposition d'un mécanisme de sélection d'action pour la partie individualité du comportement. Ce mécanisme se base sur les notions de *motivations* et d'*alternatives*. Les motivations sont l'expression de traits du caractère de l'agent, elles influencent le choix de l'action à exécuter. Les alternatives sont les résolutions possibles calculées par la partie raisonnement du comportement. Le mécanisme de sélection d'action s'appuie sur les alternatives pour déterminer, à l'aide des motivations, la meilleure action à exécuter à chaque instant.

Les motivations influençant le comportement de l'agent sont définies indépendamment du contexte d'application permettant leur réutilisation pour d'autres agents et d'autres simulations. Ma contribution vise également à apporter un enrichissement au projet CoCoA par l'apport d'un mécanisme de sélection d'action concret basé sur les motivations ainsi que la réalisation d'un atelier de conception de comportement.

Mots clés : agent logiciel, comportement, sélection d'action, motivation, individualité, raisonnement, réutilisation, personnalité, prise de décision, jeux vidéo

Abstract : This thesis proposes a new design approach for building software agents that reflect believable behaviors in simulated virtual environments. Observing a specific agent's actions usually leads to defining its overall behavior schema, which is done according to the actions it performs while attempting to reach a goal and, the choices this agent decides to make because of specific personality traits.

The contribution of this thesis is directly related to the design phase wherein an agent's behavior is to be defined. Defining an agent's behavior consists of *reasoning* and *individuality*. This thesis focuses on the introduction of a new action selection mechanism to an agent behavior's individuality. The main concepts behind the design of our action selection mechanism are *motivations* and *alternatives*. We look at motivations as the independent expressions of a specific agent's traits that influence its upcoming action selection. We look at alternatives as possible resolutions computed by the reasoning part of an agent's behavior.

In our approach, the motivations influencing agents' behaviors are domain-free. Therefore, we were able to utilize our work in contributing to the CoCoA project by providing a concrete motivation-based action selection mechanism. Further to the interaction-oriented approach promoted by the CoCoA project, the action selection mechanism we propose makes it possible to provide the means to have reliable behavioral engine that is the same for all agents.

Keywords : software agent, behavior, action selection, motivation, individuality, reasoning, reuse, personality, decision making, video games