N° Ordre : 2966
de la thèse

THÈSE

présentée

DEVANT L'UNIVERSITÉ DE RENNES 1

pour obtenir

le grade de : **DOCTEUR DE L'UNIVERSITÉ RENNES 1**

Mention : INFORMATIQUE

PAR

LAMARCHE Fabrice

Équipe d'accueil : SIAMES, IRISA, Rennes

École doctorale : MATISSE

Composante universitaire : INSA de Rennes

TITRE DE LA THÈSE

Humanoïdes virtuels, réaction et cognition : une architecture pour leur autonomie

Soutenue le 19 décembre 2003 devant la commission d'Examen

COMPOSITION DU JURY

M. :	Daniel	HERMAN	Président du jury
MM. :	Marc	CAVAZZA	Rapporteurs
	Ronan	BOULIC	
MM. :	Jean Paul	LAUMOND	Examinateurs
	Stéphane	DONIKIAN	
	Bruno	ARNALDI	

Remerciements

Durant ces trois dernière années passées à l'IRISA et plus particulièrement dans le projet SIAMES, j'ai eu l'occasion de travailler, rencontrer et discuter avec de nombreuses personnes. Ces expériences diverses n'ont fait que renforcer mon goût pour la recherche et l'intérêt que je porte à l'animation comportementale et à la réalité virtuelle en général, même si, je dois bien l'admettre, ma curiosité me pousse fréquemment à sortir de mon domaine de recherche pour aller voir ce qui se passe ailleurs.

En premier lieu, je souhaiterais remercier Bruno Arnaldi et Stéphane Donikian pour m'avoir encadré, conseillé et soutenu durant ces trois années de recherches intenses sur l'animation comportementale. Grâce à vous, j'ai eu l'occasion de travailler sur un sujet qui me tient particulièrement à cœur depuis de nombreuses années. Vous m'avez permis de réaliser un rêve d'enfant ayant débuté avec un jeu nommé l'Arche du Capitaine Blood (Philippe Ulrich) qui a fait parti des premiers jeux mettant en scène des "personnages intelligents" (je ne m'en vante pas, mais c'est vrai).

Je voudrais aussi remercier Ronan Boulic et Marc Cavazza qui ont acceptés de rapporter cette thèse et donc de m'accorder de leur temps ; merci à Jean Paul Laumond qui a accepté de faire partie de mon jury ; merci à Daniel Herman d'avoir présidé mon jury, ce fût un réel plaisir.

Je souhaite remercier l'ensemble des membres ou ex-membres du projet SIAMES que j'ai eu grand plaisir à côtoyer. Ils m'ont particulièrement soutenu durant la rédaction de ce document qui fût parfois douloureuse. Leur accueil au sein du projet ainsi que la bonne humeur régnant dans les couloirs, mais aussi en dehors, m'ont offert un cadre de travail idéal durant ces dernières années. Bruno, Stéphane, Georges, Thierry, Kadi, Alain, Anathole, j'espère pouvoir continuer à travailler avec vous. Il est une personne que l'on ne cite pas assez : Stéphane Ménardais. Merci à toi pour ton travail formidable, ta disponibilité, ta réactivité ; je crois que je me souviendrai longtemps d'Imagina et de ces nuits blanches passées... Arrive la mention spéciale aux habitants et ex-habitants du bureau du bonheur : Richard, Gwenola Turbo, Nicobouille, l'abbé Rom et le petit nouveau Seb. Je souhaite que ce bureau garde toujours cette petite ambiance qui fait son charme et sa réputation. Monsieur Jeff, ton coté belle des champs me manque autant que tes expressions bordelaises ; ce fût un plaisir de "bosser" avec toi. Une mention pour Guillé dont j'adore le débordement d'idées (le terme n'est pas assez fort), pour Glouzaille pour ces sons qui ont fait son surnom, pour Marco dont l'accent italien réchauffe le cœur, pour Marwan, encore un fou du code, pour Mathieu, Isa et leurs enfants, quelle belle famille, pour Samy keep zen, pour Pierre Antoine, jamais à court de blagues, pour les "filles de la cafet'", dont les grands cafés m'ont sauvés nombre de fois du sommeil qui me manquait, pour Sophie, dont le punch motive les troupes, pour Evelyne et Catherine, des tops de secrétaires, pour Pascale Laurent, une bibliothécaire remarquable tant par son travail que pour ses qualités humaines, pour Loïc, tu as sauvé mon affiche de thèse, pour Baldou, monsieur idée, pour Grizzly, si un jour j'ai besoin d'un graphiste, c'est toi que je débaucherai, pour Julian, mon binôme éternel avec qui j'espère pouvoir retravailler un jour, nos prises de bec me manquent (ton code aussi), pour

tous les gens que j'aurais oublié dont j'implore le pardon...

Il est deux personnes que je ne peux pas me permettre de ne pas remercier de manière explicite : Romain Thomas et Flavianne Kampf, vous m'avez aidé, soutenu et sorti d'un gouffre rédactionnel, mon moral vous en remercie, ma tête aussi (comme quoi, il n'est jamais trop tard). Un petit mot pour mes parents, vous avez eu peur de mon départ pour la faculté, c'est fini, je suis diplômé, vous pouvez être rassurés. J'étais heureux de votre sourire le jour de ma soutenance. Enfin, dans mon habitude de garder le meilleur pour la fin, mille bisous à IsabL, l'une des seules personnes qui soit capable de mettre en exergue le coté créatif d'un travail qui me prend beaucoup (trop?) de temps.

Table des matières

Introduction générale

Les ordinateurs ont longtemps été utilisés pour produire des images de synthèse, représentant des mondes virtuels statiques issus de l'imagination des concepteurs. Dans la continuité, l'animation a fait son apparition, les concepteurs, au travers d'outils dédiés, pouvaient alors animer des personnages et des objets de manière similaire à celle utilisée dans les dessins animés. Allant de paire avec l'évolution de la puissance de calcul, les animations sont devenues de plus en plus complexes à gérer, avec des exigences de qualité accrues. Pour créer des environnements toujours plus réalistes, la notion de simulation dynamique est utilisée. Son rôle est d'automatiser un certain nombre d'animations sur la base de calculs modélisant des règles physiques, similaires à celles régissant notre propre environnement. La question du peuplement de ces environnements par des entités virtuelles à l'image des organismes vivants s'est aussi posée. Il s'agit du problème traité par l'animation comportementale.

L'animation comportementale cherche à offrir des modèles et des outils permettant la création d'entités virtuelles autonomes. Ces entités perçoivent leur environnement, agissent sur ce dernier et surtout prennent d'elles-mêmes des décisions en rapport avec la situation perçue dans le but d'exhiber un comportement cohérent proche de l'organisme vivant simulé. La question des outils de contrôle, de leur niveau d'abstraction et de la validité des modèles proposés se pose alors.

Dans le cadre de cette thèse, nous proposons une architecture et des outils en vue de modéliser le comportement des humanoïdes virtuels. Cette architecture tente de faire un compromis entre des modèles issus de la psychologie comportementale, relatifs à l'être humain, et les problèmes intrinsèques liés à l'informatique : le coût de calcul et les moyens de description. Nous aborderons cette problématique au travers d'algorithmes et de langages dédiés, permettant de modéliser la composante décisionnelle des humanoïdes et leur lien avec l'environnement. Nous étudierons la description des comportements effectifs, permettant de décrire les interactions entre l'entité et son environnement, ainsi que les moyens de représenter la connaissance et de l'utiliser pour choisir des actions permettant de réaliser un but. L'entité étant située dans un environnement, nous aborderons le problème de sa représentation, dans le but d'automatiser l'extraction d'une structure adéquate pour la gestion du comportement primordial que constitue la navigation. Dans ce cadre, nous présenterons une méthode de gestion de la navigation réactive, s'inspirant d'études socio-psychologiques sur le comportement piétonnier, et permettant la gestion de foules composées de plusieurs centaines d'entités autonomes.

Chapitre 1

Contexte et contributions

1.1 L'animation comportementale : contexte et définition

Le processus de création d'environnements virtuels s'avère être un processus complexe et long. Il s'agit d'une part d'identifier ce que l'on veut représenter mais aussi d'être à même de le représenter. La phase de conceptualisation devient alors très dépendante des possibilités de réalisation offertes par les outils disponibles. Du point de vue du créateur, il ne sert à rien d'imaginer des choses trop complexes si les modèles sous-jacents ne permettent pas de les représenter. Un autre facteur très limitant est le temps de réalisation. Plus ce temps augmente, plus le coût de production augmente. Depuis déjà un certain nombre d'années, l'informatique œuvre dans ce domaine, de manière à fournir des outils permettant aux concepteurs de créer des environnements virtuels en un temps réduit. Des outils de modélisation permettent de construire, visuellement, des modèles géométriques de ces environnements. Ils offrent des interfaces conviviales et des boîtes à outils évoluées, dans le but de simplifier le processus de description. Le domaine de l'animation n'est pas en reste, il cherche à offrir des moyens de contrôle de plus en plus haut niveau permettant d'automatiser la génération d'animations réalistes. Les modèles d'animation peuvent se décliner en trois niveaux, en fonction de l'abstraction du contrôle qu'ils offrent [Arn94] :

1. Les modèles descriptifs ou guidés traduisent des effets, sans en connaître les causes. Ils constituent la première génération de modèles d'animation. Les animations peuvent être décrites image par image (de façon similaire aux dessins animés), en utilisant des méthodes mathématiques, qui génèrent des animations en fonction de contraintes fournies par l'utilisateur, ou encore par des techniques de capture de mouvement, qui reproduisent des animations calquées sur les mouvements des êtres vivants.
2. Les modèles générateurs permettent la description des causes pour produire les effets. Ce sont des systèmes dynamiques. Leur but est de produire des animations réalistes au travers de la modélisation de systèmes mécaniques et de la résolution mathématique des forces qui leurs sont appliquées. Ces modèles permettent de créer des mondes virtuels régis par des règles physiques similaires à celles de notre environnement.
3. Les modèles comportementaux visent à simuler le comportement d'organismes vivants à l'intérieur de mondes virtuels. Il s'agit de laisser une ou plusieurs entités autonomes évoluer seules dans un environnement virtuel, plutôt que d'avoir à effectuer une description exhaustive de leurs animations dans le but de les rendre « vivantes ».

En ce sens, l'animation comportementale constitue le niveau le plus abstrait de contrôle en animation. Elle cherche à offrir des outils et des concepts permettant de modéliser le comportement d'êtres virtuels. Avant de continuer, arrêtons nous sur la définition du mot comportement :

comportement : *PSYCHOL. Ensemble des manifestations objectives de l'activité d'un individu ou d'un animal. La psychologie du comportement exclut toute référence à la conscience et ne s'occupe que des relations qui existent entre les stimuli et les réponses du sujet. Cependant, allant à l'encontre des théories mécanistes du behaviorisme, la psychologie contemporaine tend à élargir le concept de comportement, pour y inclure les motivations de toute nature (affectives, sexuelles, sociales, etc.) propres aux conduites humaines.*

Encyclopédie Hachette.

Le comportement est donc ce qui caractérise l'activité d'un individu ou d'un animal à l'intérieur d'un environnement, et se traduit par un certain nombre d'interactions. Cela souligne les deux concepts importants sous-jacents à l'animation comportementale : l'être vivant simulé est donc doté de ses propres moyens de perception et d'action, qui sont reliés par une composante décisionnelle. Son état interne et l'état perçu de l'environnement déterminent directement ou indirectement le comportement qu'il adopte. Ces concepts permettent de définir l'autonomie d'une entité au même titre que pour un être vivant : il s'agit de sa capacité à décider seule de ses actions, en fonction d'une situation perçue et éventuellement de l'état de ses connaissances sur le monde.

1.2 Vers un humanoïde virtuel autonome

Dans le cadre de l'animation comportementale, il est un organisme vivant qui retient l'attention : l'être humain. Il s'agit d'un être vivant qui exhibe un comportement très complexe. Il est à la fois doté de capacités de réaction à l'environnement et de capacités de raisonnement. L'immense potentiel d'interaction entre l'être humain et son environnement, ainsi que sa capacité, souvent exploitée, de réaliser plusieurs tâches simultanément, traduit cette complexité. Elle soulève de nombreux problèmes dans le cadre de l'animation comportementale et met en évidence le besoin d'outils et de modèles évolués de description du comportement.

L'animation d'un humanoïde de synthèse se décompose généralement en plusieurs niveaux : le niveau géométrique, le niveau cinématique, le niveau contrôle du mouvement, le niveau réactif et enfin le niveau cognitif (voir fig. 1.1).

Le niveau géométrique fournit la représentation de l'humanoïde de synthèse dans l'environnement virtuel, qui se caractérise par un grand nombre de degrés de liberté. Les valeurs associées à ces degrés de libertés sont calculées dans la couche cinématique, en charge de la gestion de l'animation. La couche de contrôle du mouvement, offre des interfaces de haut niveau permettant de spécifier les gestes qui doivent être effectués par l'humanoïde. Le réalisme de l'animation dépend de ces trois premiers niveaux et plus précisément de la précision de représentation du modèle géométrique, mais aussi de la qualité visuelle des mouvements qui peuvent être effectués [BMT96, Men03].

Le niveau réactif définit des enchaînements d'actions atomiques permettant de traduire le comportement de l'humanoïde sous la forme de ses interactions avec l'environnement et effectue donc le lien avec l'animation. Enfin, le niveau cognitif symbolise les capacités de raisonnement dont l'humanoïde peut être doté. Il est dépendant d'une représentation abstraite de l'environnement, qui lui

permet de raisonner sur ses actions et leurs conséquences. Ces deux niveaux permettent d'obtenir un certain réalisme au travers de la cohérence globale du comportement exhibé par l'humanoïde de synthèse. Cette cohérence dépend principalement de la pertinence de l'enchaînement des actions par rapport à l'état perceptible de l'environnement et aux motivations de l'humanoïde. Enfin, la brique d'évolution traduit le phénomène biologique du même nom, que l'animation comportementale n'adresse pas réellement et qui est plutôt du ressort de la vie artificielle.

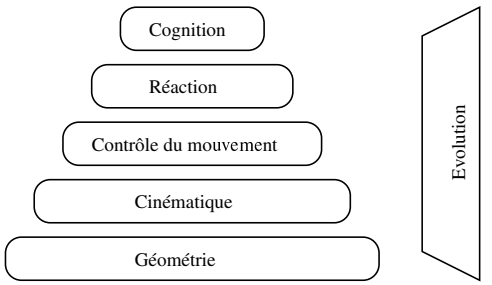


FIG. 1.1 – *Pyramide de l'animation d'un humanoïde de synthèse [TIJ+98].*

Dans le cadre de cette thèse, nous nous sommes intéressés à la spécification d'outils et d'algorithmes permettant de décrire les briques réactive et cognitive des humanoïdes de synthèse. Dans le cadre de la description du comportement, nous nous sommes focalisés sur la description des tâches qui peuvent être effectuées par un humanoïde et principalement à un mode de spécification permettant par la suite de manipuler ces tâches à un très haut niveau d'abstraction, sans connaissance précise sur leur déroulement. Cette notion est utile pour décrire simplement des comportements apparemment très complexes à reproduire conjointement tels que lire, boire, et fumer en même temps par exemple. Elle s'avère, d'autre part, nécessaire lorsque l'on passe au niveau cognitif de l'humanoïde. A ce niveau, l'humanoïde manipule ses connaissances en vue d'enchaîner, de manière cohérente, des comportements pour satisfaire un but fixé. Les tâches sont alors manipulées de manière abstraite, sans se soucier de leur modalité de réalisation. Pour faciliter la description, nous avons adopté une approche langage pour décrire les tâches et le monde cognitif de l'humanoïde. La description du monde cognitif s'effectue au travers d'une modélisation abstraite du monde et des modalités d'interaction qu'il offre, en faisant le lien avec les tâches qui peuvent être réalisées par l'humanoïde.

Mais un humanoïde de synthèse n'est rien sans un environnement dans lequel il peut évoluer. Cela soulève le point crucial du comportement de navigation, qui pose deux problèmes : la représentation de l'environnement et les interactions dynamiques avec les autres humanoïdes. L'environnement contraint la navigation dans la mesure où il contient un certain nombre d'obstacles considérés comme statiques que l'humanoïde doit éviter. Cela pose, alors, le problème de la planification d'un chemin pour se déplacer d'un endroit à un autre de l'environnement. Les interactions avec les autres humanoïdes lors de la navigation posent le problème de la détection de collision ainsi que de la réaction à adopter. Comme nous le verrons au travers d'analyses du comportement humain, le processus de navigation est soumis à un certain nombre de règles sociales à respecter pour obtenir un certain réalisme et franchir le pas entre l'animation et la simulation.

1.3 Domaines d'application

Les domaines d'application de l'animation comportementale sont variés et s'étendent des domaines ludiques tels que les jeux vidéo aux domaines de la médecine et des soins des phobies.

Jeux vidéo. Avec l'évolution de la puissance de calcul des ordinateurs, allant de paire avec la puissance des cartes graphiques, une bonne qualité de rendu graphique peut désormais être obtenue en utilisant relativement peu de temps processeur. Cette évolution permet aux concepteurs de jeux vidéo de se concentrer sur le comportement des personnages avec lesquels les joueurs peuvent interagir. Deux exemples notables sont les jeux *creatures*, de la société Cyberlife, et *Black and White*, de la société Lionhead, qui proposent des mondes peuplés de créatures autonomes. Ces créatures peuvent prendre des décisions seules, sans l'intervention du joueur, qui peut cependant, dans certaines circonstances influer sur leur comportement et tenter de leur apprendre des choses. Le développement de l'animation comportementale a fait naître une nouvelle tendance : la fiction interactive [CCM02]. L'idée est ici de fournir les grandes lignes d'un scénario, tout en laissant le joueur libre d'interagir à son gré avec des entités semi-autonomes. Ces entités suivent la ligne directrice du scénario mais peuvent prendre des décisions pour interagir « intelligemment » avec le joueur. Cela donne une grande sensation de liberté, tout en pouvant générer des situations imprévues faces aux actions de l'utilisateur. Dans le domaine du jeu, outre la qualité des animations et des graphismes, la crédibilité et la cohérence du comportement sont primordiaux pour le réalisme et l'implication du joueur.

Films et effets spéciaux L'animation comportementale a fait son apparition dans le monde de la cinématographie et des effets spéciaux depuis déjà un certain temps. La tendance actuelle est d'accroître son utilisation, au travers du développement d'outils de modélisation du comportement pour les logiciels de production d'animation en 3d. Cette tendance s'explique par un besoin de productivité accru, au travers de scènes de plus en plus complexes mais allant de paire avec des exigences de qualité croissantes. L'un des meilleurs exemples actuels est le deuxième volet de l'adaptation cinématographique du *seigneur des anneaux*. Pour la grande bataille du *gouffre de Helm*, un outil spécifique : MASSIVE[1], a été utilisé pour reproduire les comportements de foule et modéliser la composante décisionnelle des entités. Dans le même domaine, divers outils sont désormais disponibles, parmi lesquels AI-Implant de la société biographics[2], qui peut être utilisé sous la forme de module pour 3D studio Max et Maya, ou bien encore les modules RTK-Crowd et RTK-Behavior de softimage[3]. En terme de conception, ces outils offrent de grands avantages :

- le travail concepteur est simplifié par la gestion automatisée des comportements de groupe ;
- les modèles décisionnels permettent d'automatiquement générer des animations adaptées au contexte ;
- les scènes paraissent plus réelles car elles sont plus variées.

Grâce à l'automatisation, ces techniques permettent de décrire, de manière implicite, des scènes d'une complexité inatteignable sans l'aide d'outils dédiés.

1. http://www.massivesoftware.com
2. http://www.biographicstech.com
3. http://www.softimage.com

Validation ergonomique de sites. La création de lieux publiques pose des problèmes d'ordre ergonomique, autrement dit, relatifs à la qualité de leur utilisation. Des problèmes peuvent se poser quant à la bonne navigation à l'intérieur des lieux, la lisibilité des divers panneaux de direction, ou lors de situations de panique [HFV00]. L'animation comportementale, au travers de modèles se basant sur l'analyse du comportement humain, peut être utilisée pour effectuer des validations sur des maquettes virtuelles. Les éventuels problèmes peuvent alors être détectés et corrigés avant la construction des divers aménagements. Dans ce cadre, son utilisation offre des atouts qui sont de l'ordre de la sécurité, de l'ergonomie mais aussi d'ordre économique.

Mise en situation. L'interaction avec des agents autonomes, au travers de la réalité virtuelle, permet de mettre un être humain en situation dans le cadre d'un scénario. Ces capacités peuvent être utilisées dans un cadre pédagogique où l'interaction avec des humanoïdes intelligents aura pour conséquence d'augmenter la rapidité d'apprentissage par l'intermédiaire de moyens d'interaction plus proches de la réalité [LTC+00]. Un autre domaine d'application concerne l'armée où la définition du comportement d'humanoïdes permet de tester des scénarios catastrophes ou de mettre les soldats en situation [tRGM03]. Dans le domaine hospitalier, l'animation comportementale, couplée à la réalité virtuelle, peut être utilisée dans le cadre de soins aux personnes phobiques [KVD03]. L'idée consiste à plonger la personne, par l'utilisation de la réalité virtuelle, dans un milieu qui déclenche sa phobie. Dans le cadre du soin de l'agoraphobie, par exemple, les simulations de foules s'avèrent utiles, pour peupler les environnements virtuels et fournir au patient un milieu réaliste.

Ces domaines d'application variés, justifient les recherches effectuées en animation comportementale, et particulièrement le besoin d'établir un lien avec les études effectuées sur le comportement humain. Les modèles proposés, pour permettre d'obtenir un réalisme suffisant, doivent donc prendre en compte les caractéristiques propres au comportement humain.

1.4 Organisation du document

Ce document se scinde en deux parties. La première partie est un état de l'art sur l'animation comportementale. Elle se découpe en trois chapitres :

- le premier chapitre présente les grandes classes de modèles décisionnels : les modèles réactifs et orientés but ;
- le second chapitre traite de la représentation de l'environnement pour la planification de chemin et des modèles de navigation pour les entités autonomes ;
- le troisième chapitre présente trois architectures regroupant diverses approches pour la modélisation du comportement d'humanoïdes virtuels.

La seconde partie du document est dédiée à la présentation des travaux effectués durant cette thèse. Elle se découpe en cinq chapitres :

- le premier chapitre présente un mode de subdivision spatiale (à partir d'un modèle géométrique de l'environnement) permettant d'extraire des propriétés topologiques qui seront utilisées pour créer un algorithme de planification de chemin hiérarchique ;
- le second chapitre traite de la navigation réactive, au travers d'un modèle configurable inspiré d'études sur le comportement piétonnier, et d'une structure permettant d'optimiser la détection de collisions entre les entités ;

- le troisième chapitre présente HPTS++[4] un modèle de description de comportements réactifs équipé d'un mécanisme d'ordonnancement permettant la synchronisation et l'adaptation automatique des comportements ;
- le quatrième chapitre présente BCOOL[5], un langage orienté objet de modélisation de la composante cognitive des humanoïdes ainsi que le mécanisme de sélection d'actions qui lui est associé ;
- le cinquième chapitre présente l'architecture complète ainsi que différentes réalisations effectuées sur la base des modèles présentés.

Puis, nous conclurons par une synthèse des travaux effectués et exposerons les perspectives qui y sont associées.

4. Hierarchical Parallel Transition System ++
5. Behavioral and Cognitive Object Oriented Language

Première partie

Animation comportementale : l'entité et son environnement

Préambule

Les organismes vivants sont caractérisés par leur rapport avec l'environnement dans lequel ils évoluent. Ils sont dotés de capacités de perception, leur donnant une vue partielle de cet environnement et leur permettant d'acquérir une information, locale, alimentant leur processus décisionnel. Leurs capacités de décision permettent alors de corréler leurs actions à l'état perçu du monde. L'animation comportementale s'inspire de ces notions, en offrant des moyens de contrôle abstraits, permettant de décrire le comportement d'une entité autonome (ou agent) évoluant à l'intérieur d'un environnement virtuel. Dans le domaine de l'animation, l'animation comportementale se situe donc au niveau le plus abstrait du contrôle : à partir d'une évaluation de l'état de l'environnement, un modèle décisionnel contrôle les mouvements d'une entité, à l'image de l'organisme vivant qu'elle cherche à simuler. Il s'agit donc d'une branche de l'animation qui tente de faire le pont entre l'animation classique et l'intelligence artificielle, mais toujours sous la contrainte forte de temps-réel. Parmi les êtres vivants que cette branche de l'animation cherche à simuler, se trouve l'être humain. Il s'agit, certainement, de l'être vivant qui est doté du comportement le plus complexe, en particulier si l'on prend en considération ses capacités de raisonnement ainsi que la multitude des interactions qu'il est en mesure de réaliser. Dans ce cadre, l'animation comportementale devient un domaine pluridisciplinaire, s'inspirant de différentes théories informatiques, des sciences cognitives mais aussi de la psychologie du comportement.

Modèle général : boucle perception-décision-action

L'être humain évolue dans un environnement dynamique. Il est doté de trois capacités fondamentales lui permettant de réagir aux modifications de cet environnement et d'agir sur ce dernier : la perception, la décision et l'action. Ces trois briques forment la boucle perception-décision-action [Mal97] qui constitue le modèle le plus général représentant un organisme vivant, autonome, évoluant dans un environnement dynamique. La figure 0.2 présente ce modèle, qui est à la base d'un grand nombre de théories relatives au comportement, dans lequel chaque brique possède un rôle bien défini :

- La **perception** alimente le processus décisionnel avec des informations captées sur l'état de l'environnement. Ces informations sont locales ; un organisme, quel qu'il soit, ne peut percevoir que des informations qui se trouvent à proximité de lui. Bien entendu, cette notion de proximité est toute relative et dépend du type de perception ; elle définit une zone d'acquisition qui, comparativement à la taille du monde, peut être considérée comme ponctuelle.
- L'**action** représente les moyens que l'organisme possède pour modifier l'état du monde qui l'entoure. Une action peut se traduire sous plusieurs formes : parler, manipuler un objet, se déplacer...

– La **décision** joue un rôle central ; elle effectue une corrélation entre la *perception* et l'*action*. La perception alimente le processus décisionnel avec des informations "abstraites" issues des capteurs (ou sens) qui peuvent être mémorisées et/ou directement exploitées. A partir de ces informations, la composante décisionnelle choisit une ou plusieurs actions à réaliser qui sont adaptées au contexte et aux motivations de l'individu.

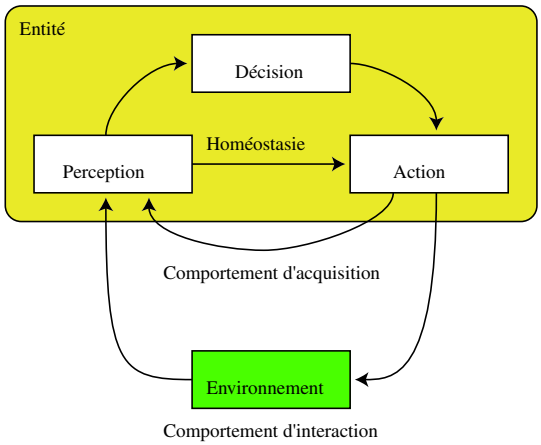


FIG. 0.2 – *Boucle perception-décision-action.*

Il existe un certain nombre de retours/interdépendances entre la perception et l'action. Ceux-ci sont de natures différentes :

– L'**homéostasie** est la boucle de régulation interne de l'organisme. Elle traduit la tendance de l'être humain à maintenir leurs paramètres biologiques face aux variations du milieu ambient. Cette boucle n'agit pas directement au niveau du comportement et ne sera donc pas prise en compte dans la suite de cette étude.

– Le **comportement d'acquisition** est la partie du comportement visant à améliorer la perception. Il peut s'agir, par exemple, de fixer une chose particulière en vue d'obtenir une information nécessaire au processus décisionnel.

– Les **comportements d'interaction** constituent la boucle la plus importante. Supervisées par le processus décisionnel et la perception, les actions modifient l'état de l'environnement, traduisant ainsi l'activité de l'organisme dans son environnement.

Ces informations traduisent le lien ténu qui lie l'être humain et l'environnement dans lequel il évolue. Le comportement semble déterminé par l'environnement mais inversement l'être humain influe sur ce dernier. Cette relation d'influence traduit l'autonomie des êtres vivants qui peuvent, en fonction de ce qu'ils perçoivent et de leur état interne, décider de leur propres actions.

Décision

Par nécessité, selon Newell [New90], un système intelligent est construit sur la base de différents niveaux de systèmes symboliques où chaque niveau du système est constitué d'un ensemble de composants du niveau inférieur, le composant de base étant le neurone. Il devient alors possible de mettre en relation le niveau d'abstraction avec le temps de réponse du système : plus le niveau

Échelle (en sec.)	Unités de temps	Système	Monde (théorie)
10^7	mois		
10^6	semaines		bande sociale
10^5	jours		
10^4	heures	tâche	
10^3	10 minutes	tâche	bande rationnelle
10^2	minutes	tâche	
10^1	10 secondes	tâche unitaire	
10^0	1 seconde	opérations	bande cognitive
0.3	0.3 seconde	*embodiment*	
10^-1	100 ms	action délibérée	
10^-2	10 ms	circuit neuronal	
10^-3	1 ms	Neurone	bande biologique
10^-4	100 micro sec	"organelle"	

FIG. 0.3 – *Échelle de temps des actions humaines*

augmente, plus la taille des composants augmente et donc plus le temps de réponse est grand. En partant de cette constatation, Newell a proposé une décomposition hiérarchique du processus décisionnel, basée sur le temps de réponse de chaque niveau (voir fig. 0.3). Il distingue quatre grandes bandes temporelles, caractérisant différents niveaux de systèmes participant au comportement :

- **Bande biologique**. Le cerveau est organisé sous la forme de circuits neuronaux interconnectés par des synapses. Ces circuits sont fonctionnellement cohérents mais représentent des fonctions arbitraires liant les entrées et les sorties. Un circuit contient approximativement 5.10^4 neurones. Étant donné le temps relativement long de transmission de l'information (de l'ordre de $1ms$), ce nombre élevé de neurones se justifie par le besoin d'augmenter le débit en utilisant un parallélisme massif et du multiplexage.
- **Bande cognitive**. La bande cognitive regroupe un ensemble d'opérations élémentaires, préenregistrées, permettant de fournir une symbolique au système et de réaliser des tâches simples. Cette bande temporelle peut être subdivisée en quatre niveaux :
  - le niveau *action délibérée* correspond au niveau où les symboles perçus sont rendus accessibles pour permettre de choisir une opération plutôt qu'une autre.
  - le niveau *embodiment* [BHPR97] correspond au niveau gouvernant les capteurs et les effecteurs du corps dans le but d'acquérir des informations pour alimenter le système symbolique. Les sens peuvent alors être vus comme un moyen d'accès à une base de connaissance extérieure au système : l'environnement.
  - le niveau *opération* permet de réaliser des opérations élémentaires préexistantes. Il permet d'obtenir des réponses rapides dans le cadre d'actions simples à effectuer en correspondance à des stimuli simples.
  - le niveau *tâche unitaire* correspond au premier niveau de composition d'opérations non élémentaires. Avec de l'entraînement, il est possible d'automatiser la composition de certaines opérations ; cela permet de réduire le temps de réponse du système.
- **Bande rationnelle**. La bande rationnelle opère au niveau de la connaissance et crée des hiérarchies de tâches. A ce niveau, le processus décisionnel est orienté par la notion de but.

En utilisant la mémoire et une représentation abstraite du monde, le processus décisionnel peut sortir du contexte actuel en se projetant dans le futur pour analyser les conséquences des choix effectués.

- **Bande sociale**. La bande sociale gère les interactions et les relations entre individus.

Ce découpage, proposé par Newell, met en évidence deux caractéristiques du comportement : la réalisation de tâches simples, apprises et automatisées et la manipulation d'une symbolique associée au monde pour s'extraire d'un contexte donné et raisonner sur les conséquences des actions. Comme le souligne Mallot [Mal97] au travers de sa théorie de la cognition orientée comportement, deux questions peuvent alors être posées :

1. Quels sont les mécanismes et les représentations les plus simples requis pour expliquer un comportement observé?
2. Quel est le comportement le plus simple qui requiert un type donné de représentation mentale?

Il a été montré que des modèles très simples de la forme stimuli-réponse peuvent reproduire un comportement apparemment complexe ; la navigation en est un exemple. Pour naviguer, il n'est pas nécessaire de posséder une représentation mentale évoluée, ce qui soulève le problème de la question 1. Cependant, dans le cas de la détermination d'un chemin pour aller d'un endroit à un autre, un mode de représentation mentale est nécessaire, ne serait-ce que pour déterminer la première direction à suivre, ce qui soulève le problème de la question 2.

Cette différentiation entre des comportements liant la perception à l'action sans passage explicite par une représentation abstraite des connaissances et des comportements plus complexes faisant intervenir une représentation des connaissances se retrouve dans les divers modèles informatiques de description du comportement. Dans la suite de ce document, nous allons utiliser les termes de comportement réactif et de comportement cognitif :

- un *comportement réactif* va qualifier un comportement qui ne fait pas explicitement appel à une modélisation des connaissances. Ce type de comportement englobera les comportements réflexes mais aussi toutes les tâches qui peuvent être effectuées sans réfléchir par un être humain.
- un *comportement cognitif* va qualifier un comportement manipulant de manière explicite une représentation des connaissances. De tels comportements incluent les mécanismes de raisonnement qui se basent sur une représentation symbolique du monde.

Ces définitions sont celles que nous utilisons, n'étant ni psychologue, ni cogniticien, nous n'avons pas pour prétention de donner une définition universelle et exacte mais plutôt de décrire le contexte dans lequel cette terminologie sera utilisée.

Approche écologique de la perception

Pour adapter les actions à l'état du monde, le processus décisionnel a besoin d'être alimenté en informations ; il s'agit du rôle de la perception. Cette perception s'effectue par le biais de capteurs fournissant des informations brutes qui nécessitent d'être interprétées. Gibson, dans ses travaux sur l'approche écologique de la perception, effectue une mise en relation entre l'environnement et un organisme animal. L'environnement est défini comme un ensemble de choses qui entourent l'animal et qui apportent quelque chose en matière de comportement. Sa théorie, nommée la théorie des

affordances[6] [Gib86, Gib97], est fondée sur ce que l'environnement « permet » à un être vivant. D'après cette approche, la perception serait une focalisation sur la nature de l'observateur plutôt que sur l'observé. L'*affordance* de quelque chose est une combinaison spécifique des propriétés de substance et surfaces, prises par référence à un animal donné. L'environnement, par l'intermédiaire de la perception, fournit donc à l'animal un ensemble de comportements possibles à adopter vis à vis des objets le peuplant. Gibson classe tout ce qui se trouve dans l'environnement de l'animal en établissant une typologie des *affordances* :

- le milieu : il s'agit du milieu dans lequel vit un animal. Pour prendre un exemple simple, l'*affordance* de l'air est de permettre de respirer.
- les substances : deux types de substances sont distinguées, l'eau sous ses différentes formes qui peut être bue ou bien servir au lavage, et les substances solides. Ces dernières possèdent la caractéristique de pouvoir être saisies et éventuellement, si leur composition le permet, de servir de nourriture.
- les surfaces et leur disposition : les surfaces à peu près planes peuvent servir à se tenir debout, alors qu'une surface verticale va constituer un obstacle pour la locomotion.
- les objets : les *affordances* qui leurs sont associées sont extrêmement variées, ils peuvent être manipulables, manufacturables...
- les autres animaux : ils constituent, au même titre que les objets, une catégorie dans laquelle les *affordances* sont nombreuses. Elles dépendent du lien entre les catégories d'entités.
- la dimension : une porte permet de passer d'un lieu à un autre, une fenêtre permet d'observer.
- les lieux : les *affordances* associées au lieu, traduisent leur utilité. Un lieu peut permettre à un être vivant de se cacher, ou au contraire de posséder un large champ de vision.

Il est possible de distinguer plusieurs caractéristiques sur le lien avec l'environnement. Les notions de surfaces et de lieux introduites par Gibson, caractérisent l'être vivant dans son environnement que l'on peut considérer comme statique et son importance dans le comportement de navigation. Cet environnement permet des choses mais n'offre pas d'interaction directe, il est plutôt exploité ou contraignant. Pour leur part, les autres catégories caractérisent des choses avec lesquelles des interactions sont possibles. L'être vivant peut alors utiliser son environnement, impliquant par là même une modification de ce dernier.

Cette théorie des *affordances* situe donc la perception au stade de l'interprétation et en ces termes offre des opportunités pour modélisation du comportement. Lors de la modélisation du comportement, il n'est pas nécessaire de modéliser l'intégralité du canal perceptif pour exhiber un comportement cohérent. Il est possible de fournir une information pré-interprétée destinée à une typologie d'organisme vivant (l'être humain dans le cas qui nous intéresse). En terme de modélisation informatique, il est alors possible de voir l'environnement comme une base de données fournissant des informations sur les typologies de comportements pouvant être adoptées par un humanoïde de synthèse. Comme nous allons le voir par la suite, cette propriété est exploitée dans certains modèles comportementaux.

6. Ce terme a été inventé par J. Gibson et n'est donc pas traduisible.

Les diverses théories qui viennent d'être évoquées, relatives à l'architecture du processus décisionnel et à l'interprétation des informations perçues, se retrouvent dans divers modèles informatiques proposés dans la littérature. Dans la suite de cette partie, nous allons, dans un premier temps, étudier les différentes catégories de modèles décisionnels dans lesquelles la différentiation entre comportement réactif et comportement cognitif se retrouve. Dans un second temps, nous traiterons de la relation entre l'humanoïde et son environnement au travers de la représentation de l'espace et du processus de navigation qui constitue un comportement primordial de par son utilisation continuelle. Comme nous le verrons, certains des modèles présentés exploitent la théorie des *affordances* pour créer des environnements informés. Dans un dernier temps, nous présenterons trois architectures complètes dédiées aux humanoïdes virtuels et étudierons le lien entre les modèles qui les constituent.

Chapitre 1

Modèles décisionnels

Nous allons nous intéresser aux modèles décisionnels associés aux entités. Comme nous l'avons décrit dans le préambule, il est possible de distinguer deux grands types de systèmes : les systèmes réactifs et les systèmes cognitifs. Cette différenciation se retrouve dans les modèles de comportements utilisés en informatique.

1.1 Modélisation de comportements réactifs

Comme nous le décrivions dans le préambule, les comportements réactifs permettent de modéliser des tâches plus ou moins complexes, n'utilisant pas directement de modélisation de connaissances. Nous allons distinguer trois grandes approches : les systèmes stimuli-réponses, pour une grande part inspirés de la biologie, les systèmes à base de règles et enfin les automates.

1.1.1 Systèmes stimuli-réponses

Les systèmes stimuli-réponses font partie des premières générations de modèles de comportements. Ils sont issus des travaux développés en éthologie (comportement des animaux) [Bro91] dans lesquels les comportements sont le résultat direct d'une série d'interactions avec l'environnement. De façon générale, la représentation s'effectue sous la forme de réseaux de nœuds interconnectés. Les nœuds en entrée du réseau propagent l'information perçue, alors que les nœuds de sortie sont connectés aux effecteurs. L'idée est donc de traduire par l'intermédiaire de ce réseau une fonction mathématique permettant de corréler directement la perception à l'action.

Le connexionisme s'inspire des théories biologiques sur l'organisation du cerveau sous forme de réseaux de neurones artificiels [1]. Un neurone formel est défini sous la forme d'une fonction mathématique (sigmoïde, gaussienne ...) de la forme $y = f(x)$ où x représente l'activation en entrée du neurone et y l'activation en sortie du neurone. Généralement, ces fonctions sont du type $\mathbb{R} \rightarrow [0; 1]$ ou $\mathbb{R} \rightarrow [-1; 1]$. Les réseaux de neurones correspondent ensuite à l'interconnexion de ces neurones

1. ***connexionisme*** *:n. masc. PSYCHOL. Courant de la psychologie cognitive issu de la théorie de l'intelligence artificielle, et cherchant à analyser le système cognitif sous forme de réseaux de connexions entre neurones «formels».*
Encyclopédie Hachette.

par des liens pondérés (corrélés à la notion de synapse en biologie). L'activation en entrée du neurone est ensuite calculée comme une combinaison linéaire de l'activation en sortie des neurones connectés et les pondérations associées aux liens. Les activations des neurones dépendent donc de deux facteurs, la topologie du réseau d'une part et les poids associés aux liens d'autre part. En jouant sur les poids des connections entre les neurones, il est possible de modifier la fonction du réseau et donc le comportement qu'il fait adopter à une entité. Il s'agit du rôle des algorithmes d'apprentissage (algorithmes génétiques, rétropropagation...), qui en fonction de la situation, influent sur les pondérations des liens de manière à configurer automatiquement le réseau en vue d'obtenir un comportement cohérent. Cette notion d'apprentissage permet aux réseaux de neurones d'être adaptatifs et de généraliser (dans une certaine mesure) les réactions apprises.

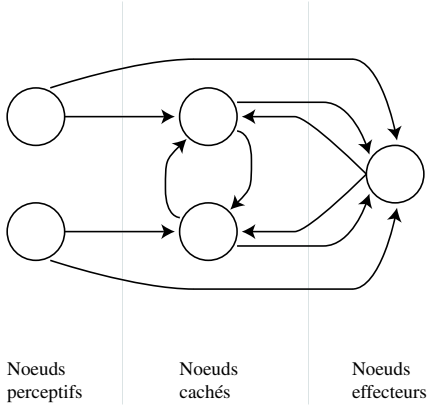


FIG. 1.1 – *Exemple de réseau SAN.*

Van de Panne utilise les réseaux SAN[2] pour contrôler le déplacement d'entités [vF93]. Des algorithmes stochastiques d'apprentissage sont utilisés afin de configurer le réseau pour obtenir un contrôleur sachant faire bouger l'entité. La figure 1.1 montre un exemple d'un tel réseau, l'architecture se décompose en trois niveaux : les nœuds perceptifs, les nœuds cachés et les nœuds effecteurs. Chaque nœud perceptif est connecté à tous les nœuds cachés et effecteurs du réseau, alors que les nœuds effecteurs sont connectés uniquement aux nœuds cachés. Cette utilisation d'un réseau rebouclé permet d'assurer une certaine continuité en prenant en compte la dernière action effectuée lors du prochain calcul.

Mathématiquement, les réseaux de neurones sont des approximateurs universels, autrement dit, ils peuvent permettre de faire une approximation, à n'importe quel degré de précision, d'une fonction mathématique continue. Grzesczuck, dans NeuroAnimator [GTH98], exploite cette propriété pour simuler les règles de la physique régissant l'environnement ainsi que le comportement des entités le peuplant. Pour maîtriser la taille des réseaux, la notion de hiérarchie de réseaux de neurones est utilisée. Chaque sous réseau possède une fonction bien définie (animation d'une jambe par exemple), et les réseaux de niveaux supérieurs permettent de contrôler (via des entrées) les réseaux de niveaux inférieurs. L'utilisation de ce système permet aussi d'effectuer un apprentissage spécialisé, centré sur des fonctionnalités bien identifiées.

Malgré leur puissance en terme d'apprentissage, ces réseaux posent différents problèmes. D'une part, l'apprentissage n'est pas simple à maîtriser. Alors que son intérêt réside dans la possibilité

2. Sensor Actuator Networks

de généraliser des notions à partir d'exemples, l'apprentissage par cœur des exemples est facile à obtenir. Dans ce cas, le réseau répond parfaitement aux exemples proposés, mais n'est plus à même de généraliser, exhibant alors des comportements incohérents face à des situations nouvelles. D'autre part, ces réseaux ne peuvent pas être interprétés, ils peuvent être vus sous la forme de boîtes noires corrélant des valeurs en entrée à des valeurs en sortie. Toute introduction d'une nouvelle fonctionnalité requiert alors la génération d'un nouveau réseau par l'intermédiaire d'une nouvelle phase d'apprentissage. Le lecteur intéressé est invité à lire l'article de Ziemke [Zie98] contenant une discussion très intéressante sur les comportements adaptatifs, tels que ceux qui peuvent être obtenus avec l'utilisation des réseaux de neurones.

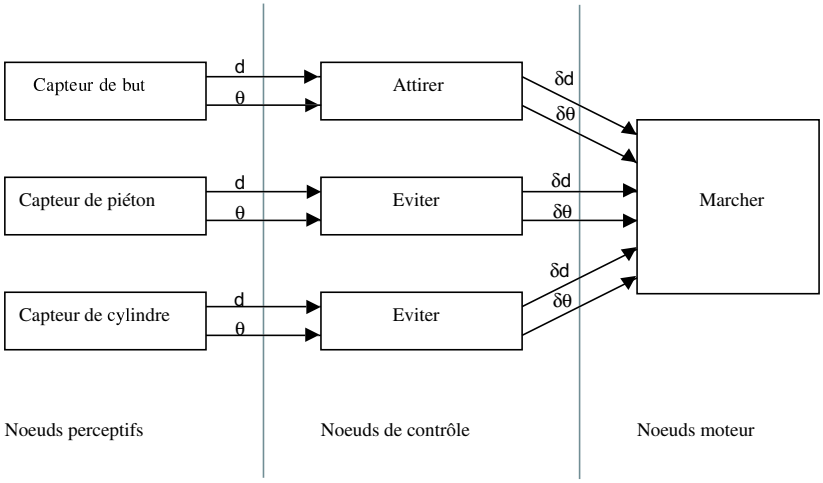


FIG. 1.2 – *Exemple de boucle SCA pour la navigation d'une entité.*

Dans [GBR+95], les boucles SCA [3] sont utilisées pour modéliser le comportement de navigation d'un agent. Il s'agit de réseaux de nœuds qui peuvent être de trois types :

- les nœuds perceptifs permettent de représenter la perception que l'agent peut avoir sur le monde qui l'entoure. Ces capteurs calculent les distances et les angles de vues entre l'agent et les objets peuplant l'environnement.
- les nœuds de contrôle permettent de gérer l'attraction de l'agent vers certaines zones de l'environnement ou la répulsion en vue d'éviter les obstacles.
- les nœuds moteurs permettent de synthétiser l'information fournie par les nœuds de contrôle en vue de générer les mouvements de l'entité à l'intérieur de l'environnement.

La figure 1.2 montre un exemple d'un tel réseau pour gérer le déplacement d'une entité à l'intérieur d'un environnement. Les nœuds perceptifs sont reliés aux nœuds de contrôle en spécifiant si la chose perçue attire l'entité (capteur de but) ou l'éloigne en vue d'un évitement (capteurs de piétons ou de cylindres). Finalement, les informations générées par les modules de contrôle sont fournies en entrée du module de marche pour permettre un déplacement de l'entité vers son but, en évitant les obstacles sur son passage.

3. Sense Control Action

Les approches de type stimuli-réponses sont particulièrement utiles pour programmer un comportement donné, nécessitant beaucoup de réactivité. Elles permettent de prendre en compte très rapidement la dynamique de l'environnement et d'agir en fonction. Cependant, elles sont utilisées pour la réalisation de tâches spécifiques relativement peu complexes. De part le faible niveau d'abstraction dont elles disposent, elles ne permettent pas de spécifier des comportements complexes de haut niveau. De fait, elles sont souvent combinées avec des modèles permettant la description de tâches complexes comme l'enchaînement de tâches simples qui peuvent être réalisées par l'intermédiaire d'un système stimuli-réponses.

1.1.2 Systèmes à base de règles

Le comportement des entités est modélisé sous la forme d'un ensemble de règles. Le choix du comportement à adopter est alors défini par une pré-condition sur l'environnement stipulant dans quel contexte adopter le comportement correspondant. Cette approche présuppose donc l'existence d'un ensemble de règles couvrant l'ensemble des situations possibles.

Reynolds, pionnier dans le domaine de l'animation de nuées, propose un modèle décentralisé, basé sur ce modèle [Rey87]. Le comportement de chaque entité appartenant à la nuée est régi par trois règles : s'éloigner des voisins pour éviter les collisions, rester proche de ses voisins pour garder une cohésion et enfin réguler la vitesse de l'entité sur les entités voisines. Le respect de ces règles donne lieu à un comportement de groupe émergent. Les entités évoluent sous la forme d'une nuée se dirigeant vers un point cible qui peut être dynamiquement modifié. Dans cette approche, l'autonomie de l'entité est discutable dans la mesure où son comportement est uniquement défini en fonction du comportement de son voisinage et pas en fonction d'une volonté propre.

Le modèle de décision associé à l'architecture de SOAR [LNR87] s'appuie sur la notion d'opérateur qui peut être une action primitive (se déplacer, tourner...), une action interne (se rappeler de quelque chose) ou des buts plus abstraits à satisfaire (passer une porte, prendre un objet) qui sont à leur tour décomposés en opérateurs plus simples jusqu'à obtenir une action primitive. Ces actions primitives sont représentées par des règles de type si-alors. Une caractéristique de ce modèle réside dans sa faculté à traiter simultanément plusieurs opérateurs qui sont mis en concurrence et filtrés grâce à un système de préférence. Son autre caractéristique est de disposer d'une mémoire de travail assurant la persistance d'un état. Cela permet d'assurer une certaine cohérence temporelle dans le comportement exhibé. A titre d'exemple, ce système a été utilisé pour décrire le comportement de *quakebots* (des agents autonomes évoluant dans le jeu quake) dotés de la capacité d'anticiper les actions du joueur [Lai01].

Coderre, au travers de Petworld [Cod89], propose de définir le comportement d'une entité par l'intermédiaire d'arbres de décision. Chaque feuille de l'arbre de décision représente un comportement élémentaire (attaquer, manger...). Les nœuds sont des experts qui choisissent entre les différentes possibilités d'actions représentées par les sous arbres associés. Ce processus permet donc de mettre en concurrence plusieurs actions et de choisir parmi celles-ci celle qui semblera la plus adaptée au contexte.

L'approche éthologique de Blumberg, et de son chien Sylas [Blu97] est similaire. Le chien dispose d'un certain nombre de variables internes pouvant influer sur ses motivations (faim, soif, peur...).

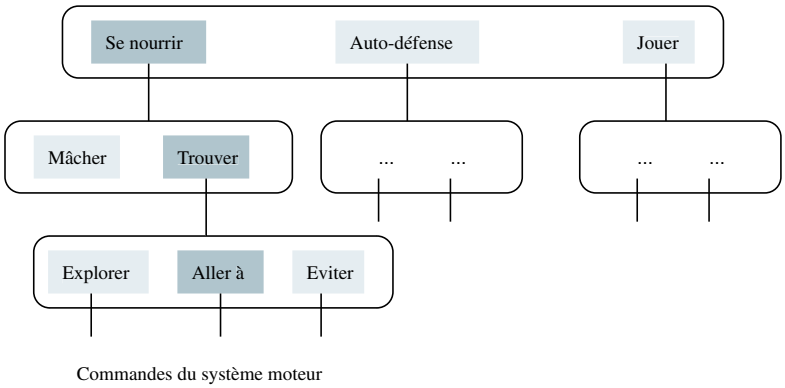


FIG. 1.3 – *Un exemple d'arbre de décision extrait de des travaux de Blumberg [Blu97].*

Les comportements sont ensuite dépendants de l'état de ces variables internes ainsi que de la situation perçue. Ce contexte permet de définir leur adéquation à l'environnement et aux motivations de la créature. Ces informations sont utilisées pour calculer un facteur d'activation ou d'inhibition du comportement. Les comportements sont ensuite organisés sous la forme de groupes de comportements compétitifs. Chaque groupe est ensuite en charge d'élire un unique comportement. La décision finale est alors prise au travers de la hiérarchie de groupes de comportements en choisissant systématiquement le comportement le plus activé, sélectionnant ainsi une branche de l'arbre. Le comportement (représenté par une feuille de l'arbre) alors sélectionné prend le contrôle du système moteur. Une notion intéressante de parallélisme est alors introduite : les comportements appartenant au même groupe que chacun des nœuds de la branche sélectionnée peuvent proposer des commandes au système moteur. Ces commandes pourront être prises en compte si les degrés de liberté concernés ne sont pas utilisés par l'action principale. La figure 1.3 montre un exemple d'une partie d'un arbre de décision. Les comportements en gris foncé représentent la branche de l'arbre sélectionnée. Les comportements en gris clair sont les comportements qui ont perdu face au comportement sélectionné. Ces comportements peuvent alors proposer des commandes secondaires au système moteur. Cette possibilité permet d'augmenter le réalisme de la simulation en offrant la possibilité de faire plusieurs choses à la fois, ce qui est une caractéristique générale du comportement.

Les approches à base de règles permettent de décrire les comportements ainsi que leur concurrence de manière relativement intuitive. Ces systèmes ont aussi la particularité d'être facilement interprétables par le concepteur qui peut les entretenir et les enrichir. Cependant, cet enrichissement demande un certain nombre de précautions pour éviter de définir des règles conflictuelles ou rendant le système incohérent. L'approche du parallélisme d'actions proposée par Blumberg s'avère très intéressante mais manque de précision et de cohérence. La synchronisation ne se fait que sur les degrés de libertés du système moteur. Deux comportements ne peuvent alors pas se partager une ressource qu'ils n'utilisent que temporairement, limitant ainsi le parallélisme. D'autre part, aucune garantie sur la cohérence des commandes secondaires prises en compte n'est assurée. De la même manière, ces systèmes n'assurent pas de cohérence dans l'enchaînement des tâches sélectionnées pour être réalisées.

1.1.3 Les automates

La réalisation d'un comportement consiste en un enchaînement d'un certain nombre de tâches considérées comme élémentaires à un certain niveau d'abstraction. Prenons l'exemple d'un comportement adopté lors d'un jeu de cache-cache par la personne devant se cacher. Dans un premier temps, la personne se cache puis attend. Deux cas peuvent alors se présenter : soit la personne est touchée durant sa phase d'attente et une nouvelle partie recommence, soit la personne se rend compte qu'elle est trouvée, auquel cas elle s'enfuit... Ici, les actions élémentaires sont des déplacements et des attentes. L'enchaînement de ces actions dépend d'une part de la dernière action effectuée et d'autre part du contexte (touché, vu...). Pour traduire ce type de comportement, le formalisme des automates à états finis peut être utilisé. Un état correspond alors à une tâche unitaire et les transitions entre états représentent les conditions d'enchaînement des tâches. Trois types de systèmes basés sur l'utilisation d'automates pour la description de tâches peuvent être distingués : les piles d'automates, les automates parallèles et les automates parallèles hiérarchiques.

Les piles d'automates. Le système de pile d'automates [NT97] permet de modéliser le comportement d'un acteur réagissant à des événements extérieurs. En début de simulation, un certain nombre d'automates peuvent être empilés. Cette pile contient alors une suite d'actions à réaliser. Lorsqu'un automate se termine, l'automate en haut de pile est dépilé et rendu actif. Lors de la réception d'un événement extérieur ou sur demande de l'automate lui-même, un nouvel automate peut s'exécuter, provoquant l'interruption et l'empilement de l'automate courant. Cette technique, permet la transposition de la notion d'appel de fonction à la notion d'automate. D'autre part, la sauvegarde de contexte liée à l'empilement de l'automate, permet de reprendre un comportement cohérent après le traitement de la réaction à un événement (cette notion peut être comparée à la notion d'interruption en système). Ce modèle a été utilisé pour décrire le comportement d'un joueur de tennis dans un environnement où la balle est soumise aux lois de la dynamique et où l'émission d'événements sonores liés à l'arbitre régule le match [NT97].

Les automates parallèles. Les PaT-Nets[4] [BW95] permettent de décrire des comportements complexes sous la forme de plusieurs automates fonctionnant en parallèle. L'ajout de la notion de parallélisme dans le système permet de gérer plusieurs actions simultanées pour un humanoïde et donc d'augmenter le réalisme de la simulation. La communication entre automates est gérée par un système de messages. Pour assurer une certaine cohérence, des sémaphores peuvent être utilisés, permettant ainsi de décrire des exclusions mutuelles entre les différents comportements. Cependant, ces sémaphores doivent être utilisés avec précaution pour être en mesure d'éviter les éventuels inter-blocages. Chaque état d'un automate peut se voir attribué un certain nombre de rôles, parmi ceux-ci:

- l'exécution d'une tâche atomique.
- le lancement d'un nouvel automate, avec éventuellement une suspension de l'automate en cours en attendant la terminaison de l'automate lancé (ce qui permet de simuler des fonctionnalités des piles d'automates décrites précédemment).
- des processus de planification/raisonnement spécialisés sur la tâche à réaliser.

Les transitions entre états sont, soit soumises à un système de priorité, soit probabilistes. Dans le cas du système de priorité, la première transition dont la condition est vraie est choisie. Dans

4. PaT-Nets: Parallel Transition Networks

le cas du système probabiliste, un tirage aléatoire est effectué pour choisir parmi les transitions franchissables celle qui sera effectivement franchie. Dans le système, dénommé Jack, les PaT-Nets sont utilisés en conjonction des boucles SCA pour permettre la navigation dans un environnement. Les automates sont entre autre utilisés pour piloter les boucles SCA. L'exemple de la figure 1.4 montre un automate associé au jeu de cache-cache [TCR+96]. Chaque type de joueur possède son entrée dans l'automate qui décrit, en fonction d'événements perçus, le comportement adopté. La structure de cet automate traduit l'enchaînement de tâches simples définissant le comportement du joueur en fonction du contexte.

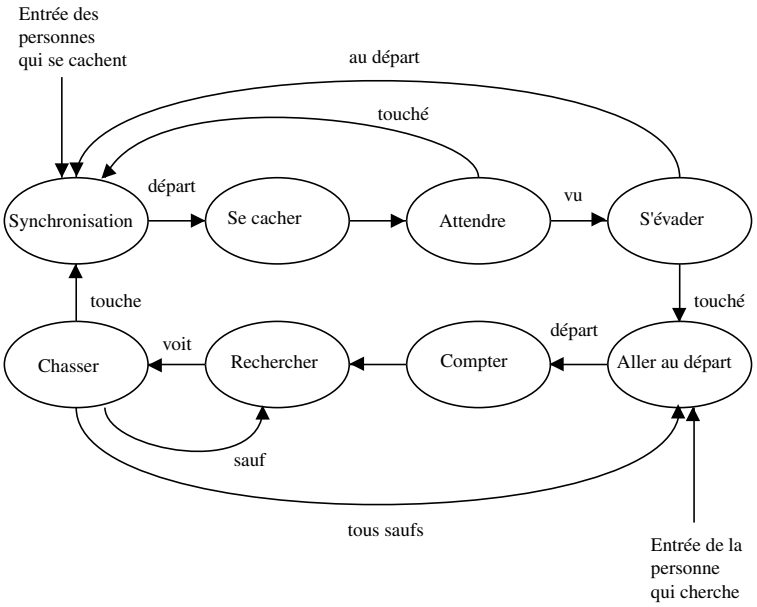


FIG. 1.4 – *Le PaT-Net associé au jeu de cache-cache*

Les automates parallèles hiérarchiques. HCSM[5] [CKP95] est un système qui gère l'exécution d'automates en parallèle et qui ajoute la notion de hiérarchie. Chaque automate du système peut être vu comme une boite noire possédant des flots en entrée et des flots en sortie et un système de messages lui permettant de communiquer avec d'autres automates. Chaque automate représente une fonction d'activité synthétisant des sorties à partir de ses entrées, de son état courant et des informations qu'il peut stocker en interne. Un automate, peut posséder, de manière transparente un certain nombre d'automates fils s'exécutant en parallèle de l'automate père. L'automate père effectue alors une synthèse des propositions des sous-automates en jouant le rôle de régulateur de la concurrence. Ce modèle est utilisé, dans le cadre d'un simulateur de conduite nommé IDS[6], pour modéliser le comportement d'un conducteur de véhicule (voir fig. 1.5). Chaque pilote est ensuite caractérisé par un certain nombre d'attributs : vitesse de conduite préférentielle, distance de sécurité désirée, agressivité.

Diverses approches exploitent ce type de description pour le contrôle des humanoïdes. Dans le domaine commercial, le produit Motivate [KAB+98] intègre divers outils pour la modélisation

5. HCSM : Hierarchical Concurrent State Machines
6. IDS : Iowa Driving Simulator

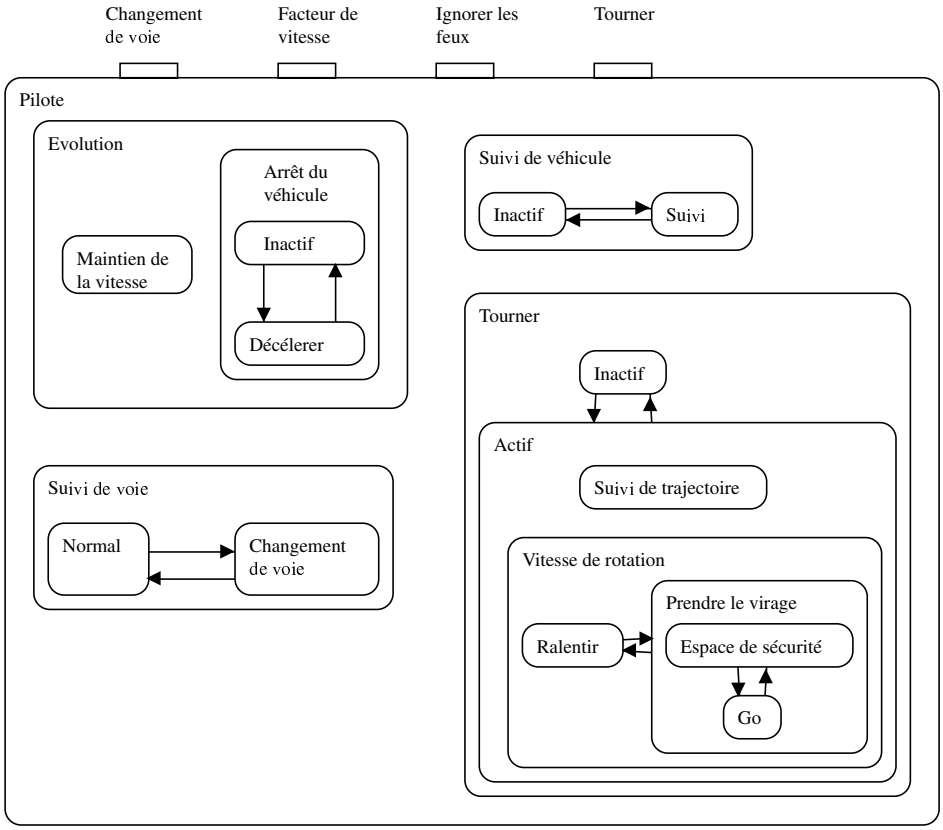


FIG. 1.5 – *HCSM : le modèle décisionnel du pilote de véhicule.*

d'acteurs autonomes et fournit un système de contrôle à base d'automates parallèles hiérarchiques. Pour sa part, le modèle HPTS [7] [Don01], est utilisé dans le cadre de la conduite de véhicules [Mor98] et de la simulation de piétons [Tho99]. Nous reviendrons plus en détail sur ce dernier modèle, car il constitue la base historique d'un partie des travaux présentés dans cette thèse.

1.2 Systèmes cognitifs et orientés buts

Dans cette section, nous allons nous intéresser à la représentation des connaissances. Cette représentation se base sur la notion d'abstraction du monde, permettant de manipuler des concepts. Comme nous allons le voir, cette notion de conceptualisation permet d'obtenir un comportement plus riche, en permettant aux agents, de trouver seuls des suites d'actions cohérentes permettant d'atteindre un but fixé.

1.2.1 Le *situation calculus*

Le *situation calculus* est un formalisme permettant de raisonner sur le monde et ses changements [MH69]. Il permet l'exploration des mondes possibles, autrement dit, l'ensemble des mondes pouvant

7. Hierarchical Parallel Transition Network

résulter de l'exécution d'une ou plusieurs actions.

Les situations. Une situation correspond à l'état complet du monde à un instant donné. Dans la mesure où toutes les informations ne peuvent être stockées, les situations sont décrites par l'intermédiaires de faits décrivant les propriétés du monde. Ces faits peuvent ensuite être utilisés pour déduire d'autre faits dans les situations futures.

Les *fluents*. Les *fluents* décrivent une propriété sur le monde qui peut changer au cours du temps. Ces derniers sont définis comme un fonction prenant une situation en paramètre et pouvant renvoyer l'état de la propriété, permettant ainsi de spécifier des caractéristiques qui peuvent/doivent être présentes dans le monde. Généralement, ils représentent un fait atomique (il pleut) ou encore une relation entre des objets du monde (le parapluie est dans le placard).

Les actions. Les actions permettent d'effectuer des changements sur une situation. Le résultat d'une action donnée est donc une nouvelle situation dans le futur. La situation définit la possibilité de réalisation d'une action. L'ensemble des propriétés qui doivent être présentes dans la monde pour rendre cette action exécutable constitue la pré-condition. L'effet d'une action représente l'ensemble des modifications qui sont effectuées par l'action sur la situation dans laquelle elle a été exécutée pour produire la situation résultante. Lors de leur description, les effets peuvent être conditionnels. Autrement dit, en fonction de la situation, les effets d'une action peuvent varier.

La connaissance. Il existe des logiques permettant de gérer la connaissance du monde. Par ce biais, il est possible de décrire des actions perceptives, permettant d'acquérir une connaissance. Cette gestion de la connaissance permet d'augmenter le niveau d'abstraction du formalisme. Il devient possible de trouver des stratégies permettant d'acquérir de l'information utile pour l'exécution d'un plan, durant sa construction.

L'utilisation de ces concepts permet de calculer des ensembles de mondes futurs, résultats de l'exécution d'une ou plusieurs actions sur la situation courante. La planification consiste alors à trouver une suite d'actions permettant de générer une situation dans laquelle les buts sont satisfaits. Mais ce formalisme souffre d'un problème lié à la description des actions. Ce problème est qualifié de "frame problem" ou problème de la fenêtre [SL03]. Il réside dans la description de tout ce qui ne change pas dans le monde après l'exécution d'une action qui s'avère dans la pratique quasi impossible. La solution à ce problème réside dans l'hypothèse forte du monde clos. Cette hypothèse stipule que tout ce qui n'est pas explicitement décrit comme étant la résultante d'une action reste inchangé après l'exécution de cette même action.

Plusieurs environnements de programmation permettant d'exploiter la puissance du *situation calculs* existent. Parmi ceux-ci on peut citer GOLOG[8] [LRL+97] et plus récemment CML[9] [Fun98] permettant de décrire le comportement de haut-niveau d'une entité autonome. GOLOG et CML sont des langages ayant un grand nombre de similarités, nous allons donc décrire plus particulièrement CML de par son utilisation dans le domaine de l'animation comportementale. Lors de la création de CML, Funge s'est particulièrement intéressé à l'utilisation des *fluents* pour représenter des valeurs

8. alGOL in LOGic
9. CML : Cognitive Modelling Language

- *Déclaration d'une action*
  occurence <ACTION> **results in** <EFFECT> **when** <PRECONDITION>
- *Exécution d'une suite d'actions*
  <ACTION> ; <ACTION>
- *Test d'une expression*
  test(<EXPRESSION>)
- *Choix non-déterministe entre deux actions*
  choose <ACTION> **or** <ACTION>
- *Test de la forme si alors sinon*
  if(<EXPRESSION>) <ACTION> **else** <ACTION>
- *Itération non-déterministe d'une action*
  star <ACTION>
- *Boucle tant que*
  while(<EXPRESSION>) <ACTION>
- *Choix non déterministe des arguments d'une action*
  pick(<EXPRESSION>)<ACTION>
- *Définition d'une procédure*
  void P(<ARGLIST>) <ACTION> <PRECONDITION>

FIG. 1.6 – *Les principaux éléments syntaxiques de CML [Fun98]*

numériques en introduisant la notion d'*interval-valued fluent* [Fun99] permettant de gérer un degré d'incertitude sur une valeur numérique. Avec l'évolution de l'incertitude au court du temps, il est possible de gérer une connaissance estimée sur une valeur précédemment perçue. Lorsque cette estimation s'avère trop imprécise, des actions de perception peuvent alors être déclenchées pour permettre de mettre à jour la connaissance. D'autre part, cette incertitude peut aussi être prise en compte lors de la planification. Le langage CML est à cheval entre un langage déclaratif type PROLOG et un langage impératif. Il offre les structures de contrôle classiques (tests, boucles) et ajoute des choix non-déterministes d'actions ainsi que l'itération non déterministe d'une action. La notion de procédure est aussi présente, permettant d'ajouter des processus spécialisés dans la résolution d'un problème donné. Les principales composantes du langage sont décrites figure 1.6. Ce langage a permis de mettre en œuvre plusieurs démonstrations [FTT99] dont un T-Rex intelligent cherchant à regrouper des Raptors dans une enclave (voir fig. 1.7), un homme des mers cherchant à éviter un requin ou encore un système de caméras intelligentes.

Cette approche dispose d'un très grand pouvoir d'expression. Elle permet de décrire des mondes complexes ainsi que les actions disponibles pour le modifier. Les entités peuvent alors planifier une suite d'actions en raisonnant sur leur conséquences dans le but d'atteindre un but fixé par l'intermédiaire d'un langage mélangeant des aspects déclaratifs avec des aspects procéduraux. L'introduction de la notion de connaissance permet d'augmenter le niveau d'abstraction en permettant de raisonner sur une information « incomplète ». Les plans peuvent alors contenir des actions de perception

FIG. 1.7 – *Un TRex programmé en CML regroupant des raptors dans une enclave. Ces images sont extraites de [FTT99].*

permettant l'acquisition des informations jusqu'alors indisponibles. Plusieurs aspects sont cependant à prendre en compte : quel est l'effort de description d'un problème à partir de ce formalisme et surtout quel est le coût de calcul qui lui est associé. L'exploration des mondes possibles peut s'avérer très coûteuse, mais peut cependant être réduite par l'utilisation des structures de contrôle qui permettent d'introduire une connaissance experte à l'intérieur du système.

1.2.2 STRIPS

STRIPS[10] [FN71] est un langage de description d'actions qui permet d'exploiter un sous ensemble du *situation calculus*. Le monde est décrit sous le forme de propriétés pouvant être présentes ou absentes. Ces propriétés peuvent être corrélées à la notion de *fluent* possédant une valeur booléenne où vrai signifie présent et faux signifie absent. Comme les *fluents*, les faits représentent généralement une propriété atomique ou l'existence d'une relation entre des objets du monde. Les actions sont ensuite définies par l'intermédiaire d'opérateurs possédant l'équivalent d'une pré-condition et des effets. La pré-condition regroupe un ensemble de propriétés qui doivent être présentes dans le monde pour que l'action puisse être exécutée. Les effets d'une action se traduisent par l'intermédiaire de deux listes : une liste d'ajout et une liste de suppression. La liste d'ajout contient l'ensemble des propriétés qui sont ajoutées lors de l'exécution de l'action alors que la liste de suppression traduit l'ensemble des propriétés qui sont supprimées. Ces listes représentent donc des conjonctions de faits. La figure 1.8 montre un exemple de définitions d'opérateurs (ou actions) STRIPS liés au problème des blocs. Les opérateurs définis, consistent à prendre un bloc, pour le poser soit sur la table, soit sur un autre bloc. Les buts qui peuvent alors être exprimés consistent en l'obtention d'une configuration particulière pour les blocs.

La planification dans ce type de domaine correspond donc à trouver une suite d'actions permettant d'ajouter les faits constituant les buts à partir de l'état courant du monde. Un problème de

10. STanford Research Institute Planning System

Opérateurs		
		put(?x,?y)
PRE	:	ontable(?x), clear(?x), clear(?y)
ADD	:	on(?x,?y)
DEL	:	ontable(?x), clear(?y)
		put(?x,?y)
PRE	:	on(?x,?z), clear(?x), clear(?y)
ADD	:	on(?x,?y), clear(?z)
DEL	:	on(?x,?z), clear(?y)
		puttable(?x)
PRE	:	clear(?x), on(?x,?y)
ADD	:	ontable(?x),clear(?y)
DEL	:	on(?x,?y)

FIG. 1.8 – *Exemple d'opérateurs STRIPS définissant le problème des blocs.*

planification STRIPS peut être représenté par un quadruplet $P = (F,O,I,G)$ où F est un ensemble de faits, O est l'ensemble des opérateurs, $I \subset F$ est la situation initiale et $G \subset F$ est la situation à obtenir. Les opérateurs de O sont considérés comme paramétrés, autrement dit, les variables ont toutes été remplacées par des constantes. Les pré-conditions d'un opérateur $op \in O$ seront notées $Pre(op)$, les faits ajoutés seront notés $Add(op)$ et enfin les faits supprimés seront notés $Del(op)$ avec $(Pre(op), Add(op), Del(op)) \subset F^3$. A partir d'un situation $s \subset F$, $A(s)$ représente l'ensemble des opérateurs $op \in O$ tels que $Prec(op) \subset s$, autrement dit, l'ensemble des actions réalisables dans la situation s. La planification va donc consister à trouver une suite d'actions à partir d'une situation initiale $I \subset F$ telle que l'exécution de ces actions produise une situation $s' \subset F$ telle que $G \subset s'$. Plusieurs types de planification ont été proposés pour résoudre ce problème, nous allons plus particulièrement nous intéresser à deux d'entre elles : GRAPHPLAN et HSP [11].

GRAPHPLAN [BF97] est un moteur de planification fonctionnant à partir d'une représentation par l'intermédiaire de graphe : le graphe de planification. Ce graphe est composé de nœuds qui peuvent être de deux types :

- un nœud n de type proposition représente un fait qui peut être présent dans le monde, donc $n \in A$.
- un nœud n de type action représente une action permettant de modifier l'état du monde, donc $n \in O$.

Les arcs pour leur part peuvent être de trois types :

- pré-condition, il s'agit d'un arc reliant un fait f à une action op, stipulant que ce fait constitue l'une des pré-conditions de l'action, donc tel que $f \in Prec(op)$.
- ajout, il s'agit d'arcs reliant une action (op) à un fait (f), traduisant l'ajout de ce fait par l'exécution de l'action, donc tel que $f \in Add(op)$.

11. Heuristic Search Planning

– suppression, il s'agit d'arcs reliant une action (op) à un fait (f), traduisant la suppression de ce fait par l'exécution de l'action, donc tel que $f \in Del(op)$.

Le graphe de planification peut ensuite être construit, sous la forme d'un graphe en couche, alternant des couches de faits et des couches d'actions. La première couche décrit l'état du monde au moment de la demande de planification, elle est donc initialisée avec I. La seconde couche décrit l'ensemble des actions $A(I)$ dont les pré-conditions sont satisfaites dans l'état initial du monde avec l'ajout d'une action spéciale (no-op) stipulant la conservation d'un fait de la couche précédant l'action vers la couche la suivant. La figure 1.9 montre l'exemple d'un tel graphe, alternant couche de faits et couches d'actions contenant l'opérateur *no-op* pour propager les faits d'une couche à l'autre.

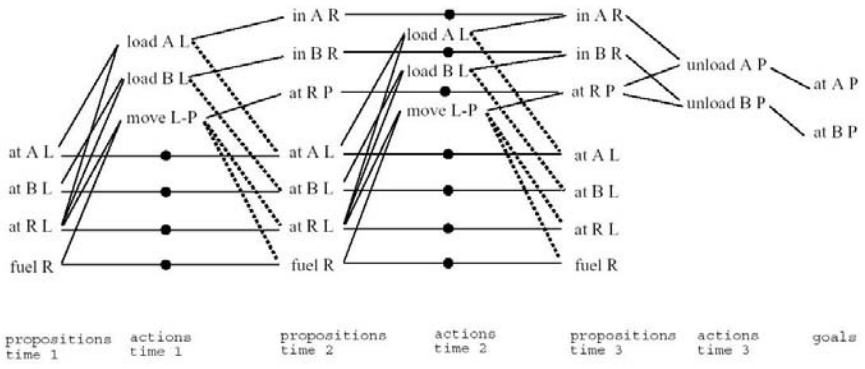


FIG. 1.9 – *Un exemple de graphe de planification généré par GRAPHPLAN.*

Ce graphe, contient des ensembles de faits et d'actions conflictuels. Pour représenter ces conflits, des informations d'exclusion mutuelles sont ajoutées, d'une part entre les actions d'une même couche et d'autre part entre les faits d'une même couche. Deux actions d'une même couche peuvent être marquées comme étant exclusives dans deux cas:

– les interférences : ce cas se produit lorsque qu'une action op_1 supprime une pré-condition d'une autre action op_2 ou un fait ajouté par op_2. Cette condition d'exclusion peut donc être exprimée sous la forme : $(Del(op_1) \cap Pre(op_2) \neq \emptyset) \vee (Del(op_1) \cap Add(op_2) \neq \emptyset)$.
– les besoins concurrentiels : ce cas se présente lorsqu'une pré-condition d'une action op_1 et une pré-condition d'une action op_2 sont marquées comme étant mutuellement exclusives.

Deux faits (p,q) d'une même couche sont marqués comme étant mutuellement exclusifs si toutes les manières d'obtenir p sont exclusives avec toutes les manières d'obtenir q. Ces marquages d'exclusion mutuelle constituent un sous ensemble de toutes les exclusions réelles mais sont utilisées par la suite pour guider le processus de recherche. Le graphe qui vient d'être décrit est généré étape après étape :

– une phase de définition des actions exécutables à partir du niveau courant,
– une phase de calcul des effets de ces actions,
– une phase de détermination des exclusions mutuelles en appliquant les règles qui viennent d'être décrites.

Ce processus est répété jusqu'à trouver une couche dans laquelle tous les buts sont satisfaits et ne possèdent pas de marquage d'exclusion mutuelle. A ce moment, un algorithme, fonctionnant par

chaînage arrière (partant donc des buts) est utilisé pour trouver l'ensemble des actions générant le plan. Le principe consiste à sélectionner à partir du dernier niveau du graphe un ensemble d'actions permettant de satisfaire les buts et qui ne sont pas mutuellement exclusives. Les pré-conditions du niveau précédent deviennent alors le nouvel ensemble de buts à satisfaire et le processus est répété jusqu'à l'obtention du plan. Si à un niveau donné, aucune solution n'est trouvée, l'algorithme renvoie un échec et la combinaison d'actions précédente est remise en cause spécifiant ainsi un nouvel ensemble de sous-buts à satisfaire. Il peut arriver que même si les buts ne sont pas marqués comme mutuellement exclusifs à un niveau donné, aucune solution ne soit trouvée (le mécanisme d'exclusion mutuelles est donc incomplet), dans ce cas, l'algorithme développe un nouveau niveau d'exploration puis répète le processus qui vient d'être décrit.

Cet algorithme garantit de trouver une solution (dans la mesure où elle existe) à un problème donné en minimisant le nombre de couches du graphe utiles à la planification. Les solutions alors fournies s'expriment sous la forme d'actions qui peuvent être menées en parallèle lorsqu'elles appartiennent à la même couche du graphe. Le nombre d'actions n'est donc pas minimisé contrairement au nombre de pas de temps utiles à la réalisation du plan. Le développement des couches d'actions et de faits durant la planification est polynomial en temps et en espace [BF97]. L'algorithme, dans le cas de problèmes difficiles peut donc parfois ne pas fournir de réponse par manque de mémoire.

Une autre approche intéressante est celle développée autour de HSP [BG01]. L'idée est d'utiliser l'aide d'une heuristique pour guider la recherche d'un plan. Cette heuristique est automatiquement extraite de la description du monde sous la forme d'opérateurs STRIPS. L'idée de construction de l'heuristique consiste à considérer un sous problème du problème à résoudre en ne prenant pas en considération les faits supprimés par une action et donc en ne considérant pas les exclusions mutuelles. Pour définir l'heuristique, nous allons utiliser la fonction $next(s) : F \longrightarrow F$ définie comme suit :

$$next(s) = \bigcup_{op \in A(s)} Add(op)$$

Cette fonction calcule l'ensemble des faits accessibles en utilisant une unique opération depuis la situation s. La fonction $g(s,f)$, fournissant le nombre minimal d'actions à exécuter depuis la situation s pour obtenir le fait f, est alors définie comme suit :

$$g(s,f) = min_{f \in next^k(s)} k$$

Cette distance s'avère dans un grand nombre de cas être une sous-estimation du nombre d'actions car les informations de faits supprimés par une action ne sont pas prises en compte. Au mieux elle correspond exactement au nombre d'actions, ce qui signifie que toutes ces actions sont indépendantes. L'heuristique $h(s,G)$ fournissant une estimation de la distance entre une situation s et le but G peut alors être définie par l'intermédiaire de la fonction $g(s,f)$. L'heuristique utilisée dans HSP est définie comme suit :

$$h(s,G) = \sum_{f \in G} g(s,f)$$

Cette heuristique peut être qualifiée de non-admissible dans la mesure où elle considère que tous les faits constituant le but sont indépendants, ce qui s'avère faux en règle générale. Elle a donc tendance à surestimer le coût et ne permet donc que rarement de fournir une solution optimale. Elle est cependant préférée à l'heuristique admissible $h'(s,G) = max_{r \in G} g(s,r)$ car elle contient plus d'information et s'avère donc plus discriminante lors de la recherche.

En utilisant l'heuristique $h(s,G)$, deux algorithmes HSP et HSP2 ont été proposés. HSP fonctionne avec un algorithme de type *hill-climbing search*. En considérant qu'un nœud est une situation s et que les fils de ce nœud sont les situations accessibles en appliquant les opérations de $A(s)$, cet algorithme explore les fils d'un nœuds minimisant l'heuristique $h(s,G)$. Pour sa part, HSP2 fonctionne en utilisant une variante de l'algorithme A^* : l'algorithme WA^*. Cet algorithme associe des poids aux nœuds explorés tels que $v(n) = g(n) + Wh(n)$ où g(n) dénote la distance depuis l'origine, h(n) la valeur de l'heuristique déterminant la distance au but et W un facteur tel que $W \geq 1$. Dans le cas $W = 1$ l'algorithme se comporte exactement comme un algorithme A^*. Dans le cas $W > 1$, l'algorithme fournit une solution dégradée, plus rapidement que l'algorithme A^*, et dont le coût n'excède pas W fois le coût optimal.

Pour ces deux algorithmes, le calcul le plus coûteux est l'évaluation dans chaque nouvelle situation de l'heuristique $h(s,G)$. Pour pallier ce problème, une dernière variante est proposée : l'algorithme HSPr. Plutôt que de partir de l'état courant du monde et de chercher à atteindre le but, il fonctionne en partant des buts et en essayant de trouver des suites d'actions telles que leurs pré-conditions soient satisfaites dans l'état initial I. L'algorithme part donc de la situation G, en explorant les actions op satisfaisant la contrainte suivante : $f \in (G \cap Add(op))$. Les pré-conditions de ces actions deviennent alors un nouvel ensemble de sous-buts à satisfaire. L'avantage de cette approche réside dans l'évaluation de l'heuristique. Les coûts $g(I,f)$ associés à l'obtention du fait f à partir de la situation initiale I peuvent n'être calculés qu'une seule fois dans la mesure où la planification commence à partir du but, contrairement aux algorithmes précédents qui devaient réévaluer ces coûts pour chaque nouvelle situation. L'heuristique associée à chaque situation s devient alors :

$$h_{add}(s) = \sum_{f \in s} g(I,f)$$

Dans la mesure où cette heuristique n'est dépendante que de la valeur de $g(I,f)$ qui est constante, $h_add(s)$ est constante. Cette propriété permet donc de pallier (en partie) le surcoût de calcul associé à l'évaluation de l'heuristique dans les algorithmes HSP et HSP2. Au même titre que GRAHPLAN, HSPr utilise la notion d'exclusion mutuelle pour élaguer l'espace de recherche. Les exclusions mutuelles se traduisent par un ensemble M de paires de faits telles que :

1. (p,q) n'est pas vrai dans I,
2. pour chaque action op telle que $p \in Add(op)$:
   $(q \in Del(op)) \vee (q \notin Add(op) \wedge \exists r \in Pre(op), (r,q) \in M)$.

Cet ensemble étant coûteux à calculer, une approximation en est faite. Cette approximation s'appuie sur le calcul de deux ensembles :

- M_A qui est l'ensemble des paires $(p,q) \in F \times F$, pour lesquelles il existe une action op telle que $p \in Add(op) \wedge q \in Del(op)$.
- M_B qui est l'ensemble des paires (r,q) telles qu'il existe une paire $(p,q) \in M_A$ et une action op telle que $r \in Pre(op) \wedge p \in Add(op)$.

Ensuite, les exclusions mutuelles sont calculées en supprimant de l'ensemble $M' = M_A \cup M_B$ les paires ne satisfaisant pas les deux contraintes qui viennent d'être énumérées et en les stockant dans l'ensemble M^*. Au même titre que GRAPHPLAN, les exclusions mutuelles répertoriées sont un sous-ensemble des exclusions mutuelles réelles, donc $M^* \subseteq M$. L'algorithme HSPr se déroule ensuite en calculant dans un premier temps $g(I,f)$ pour tous les faits de A. Ensuite, la recherche s'effectue en utilisant l'algorithme WA^*. Les situations produites sont filtrées par l'intermédiaire

des informations d'exclusion contenues dans M^*. Si une situation s est considérée comme valide, $h_{add}(s)$ est évalué en s'appuyant sur les pré-calculs de $g(I,f)$.

Les trois algorithmes qui viennent d'être présentés : HSP, HPS2 et HSPr diffèrent en deux points : l'algorithme de recherche (WA^* et *hill-climbing search*) et le sens de recherche (de l'état initial vers le but ou l'inverse). D'après les conclusions de Bonet [BG01], l'algorithme HSP2 serait le plus robuste et globalement le plus efficace malgré le calcul intensif effectué sur l'heuristique. L'un des avantages de cette méthode basée sur le calcul heuristique réside dans le nombre de situations développées lors du calcul. Il s'avère relativement faible et permet à ce système de dépasser GRAPHPLAN sur la taille des problèmes possibles à résoudre.

Le formalisme STRIPS s'avère moins expressif que le *situation calculus*, un certain nombre de choses telles que la prise en compte de la connaissance des faits, les effets conditionnels ne sont pas traduits. La contrepartie réside dans la puissance des algorithmes de planification qui peuvent alors être utilisés. Ces algorithmes exploitent les particularités du mode de description pour restreindre et diriger automatiquement la recherche vers le but, arrivant ainsi à limiter la combinatoire de l'exploration. Contrairement aux programmes développés autour du *situation calculus*, la simple description des actions est suffisante pour permettre la planification, sans description algorithmique supplémentaire sur le mode de recherche.

1.2.3 Planification HTN *(Hierarchical Task Networks)*

La planification HTN [12] diffère des modes de planification présentés jusqu'alors. Plutôt que de décrire un ensemble d'actions atomiques permettant de modifier le monde et de chercher une suite linéaire d'actions satisfaisant un but, la planification par HTN fonctionne de manière hiérarchique [EHN94] en s'appuyant sur trois concepts que nous allons présenter : les tâches, les méthodes et les actions.

Les tâches. Les tâches constituent ce que l'on peut qualifer de but dans les méthodes de planification qui ont été présentées précédemment. Elles peuvent correspondre à la satisfaction d'un ensemble de propriétés sur le monde (comme dans le situation calculus ou les domaines STRIPS), à la réalisation d'un ensemble d'actions, comme par exemple visiter la ville, ou bien à une combinaison des deux. Elles acceptent de prendre un certain nombre de paramètres, rendant ainsi possible la description de tâches générales pour lesquelles les paramètres seront instanciés lors d'une demande de planification. Le pouvoir d'expression associé à la notion de tâche est donc très grand. Elles permettent notamment d'exprimer des comportements complexes n'ayant pas forcément une influence exprimable sous la forme de faits sur l'environnement.

Les méthodes. Les méthodes correspondent à la manière de réaliser une tâche. Autrement dit, une méthode décrit une suite de tâches et/ou d'actions à réaliser pour remplir la tâche à laquelle elle correspond. Une notion intéressante de la planification HTN apparaît alors, il est possible de décrire plusieurs méthodes permettant de réaliser la même tâche. Une méthode est décrite par l'intermédiaire de trois informations :

- la tâche qu'elle permet de réaliser.
- ses pré-conditions, qui décrivent l'état du monde à partir duquel elle peut être utilisée.

12. Hierarchical Task Network

– une liste de sous-tâches ou d'actions, à réaliser successivement, permettant de réaliser la tâche qu'elle remplit.

Ces méthodes permettent donc de décrire la base de connaissance disponible pour la planification. Les pré-conditions permettent d'ajouter des restrictions à l'utilisation d'une méthode, par exemple, un taxi peut être utilisé pour voyager sur des distances relativement courtes, alors qu'un avion peut être utilisé pour voyager sur de très longues distances. La liste ordonnée de sous-tâches décrit une forme de plan plus ou moins abstrait permettant de guider la recherche d'une solution et réduisant ainsi l'espace de recherche.

Les actions. Les actions sont représentées par l'intermédiaire d'un formalisme de type STRIPS, elles possèdent des pré-conditions et des effets. Elles sont directement réalisables, sous réserve de la véracité de leur pré-condition, sans avoir recours à une forme de décomposition.

Les concepts qui viennent d'être exposés dénotent de la conception hiérarchique de la planification HTN. Une demande de planification se traduit donc par la demande de réalisation d'une tâche. Le processus de planification consiste alors à décomposer la tâche par l'intermédiaire des différentes méthodes proposées (voir fig. 1.10). Lorsque la pré-condition d'une méthode n'est pas satisfaite, elle est abandonnée au profit d'une autre et ainsi de suite. Lorsque que la liste des sous-tâches associées à une méthode ne contient plus que des actions, et que ces actions peuvent être enchaînées de manière à ce que leurs pré-conditions soient successivement satisfaites au travers des effets des actions, le processus d'expansion s'arrête car une solution partielle est trouvée. La planification fonctionne donc avec un algorithme de chaînage arrière, similairement à un moteur d'inférence PROLOG [13], explorant les différentes décompositions proposées jusqu'à trouver une solution exprimée sous la forme d'une suite d'actions satisfaisant la tâche à réaliser.

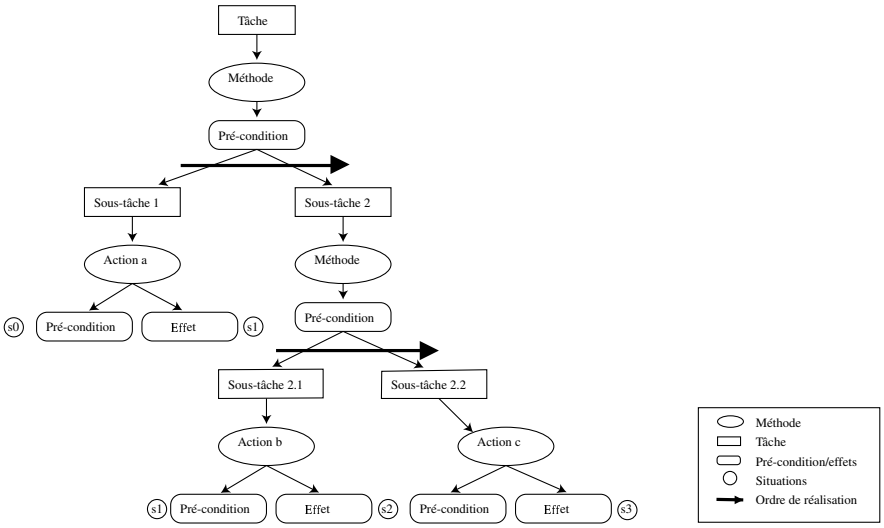


FIG. 1.10 – *Planification HTN : Exemple de décomposition de tâche.*

Ce principe de planification repose sur deux hypothèses : d'une part les tâches doivent être

13. PROgramming LOGic

décomposables en sous-tâches et d'autre part elles doivent pouvoir être ordonnées de façon à ce que les effets des actions n'invalident pas les pré-conditions des actions suivantes. Ces deux contraintes constituent les limitations de ce type d'approche où la description des méthodes doivent être faites avec précaution pour assurer l'absence d'interférences. Cependant elle offre l'avantage de permettre d'introduire une connaissance experte sous la forme de méthodes appropriées. Elle permet ainsi de contrôler de manière fine la réalisation des tâches, ce qui explique son utilisation dans la domaine de la fiction interactive [CCM02]. Le contrôle ainsi offert au travers des méthodes permet de décrire des scénarios de manière fine tout en offrant diverses possibilités de réalisations dépendantes du contexte mais aussi des éventuels échecs de réalisation. Cependant, même s'il a été prouvé que ce formalisme est plus expressif que le formalisme STRIPS, le biais introduit par la décomposition des tâches offre moins de flexibilité quant à la réalisation des but fixé [CLM$^+$03].

1.2.4 Les mécanismes de sélection d'actions

Les mécanismes de sélection d'actions ont été introduits pour gérer deux aspects du comportement : d'une part, la réactivité et donc la possibilité d'exploiter des opportunités offertes par l'environnement ; d'autre part, la recherche d'une suite d'actions permettant de satisfaire un but donné. Ces mécanismes sont typiquement basés sur un graphe d'actions dont les liens entre actions définissent des canaux de propagation d'énergie. Le principe étant alors de calculer l'énergie associée aux actions, sachant que celle possédant la plus grande activation est normalement l'action la plus adaptée.

Dans ce but, Maes a présenté un mécanisme de sélection d'actions [Mae90] basé sur une description des actions par l'intermédiaire d'opérateurs STRIPS, non paramétrés. Chaque action est décrite par un quadruplet (c,a,d,l) dans lequel:

- c représente les faits qui doivent être présents dans le monde pour que l'action puisse être exécutée.
- a représente les faits qui sont ajoutés lors de l'exécution de l'action.
- d représente les faits supprimés lors de l'exécution de l'action.
- l représente le niveau d'activation de l'action.

Ces actions sont ensuite connectées sous la forme d'un réseau possédant trois types de liens. Soient (c_i,a_i,d_i,l_i) et (c_j,a_j,d_j,l_j) deux actions :

- il existe un lien de succession de i à j par proposition $p \in (a_i \cup c_j)$.
- il existe un lien de précédence de i à j par proposition $p \in (c_i \cup a_j)$.
- il existe un lien de conflit de i à j par proposition $p \in (c_i \cup d_j)$.

Un exemple d'un tel réseau est fourni sur la figure 1.11, dans laquelle I correspond aux liens d'inhibitions, B aux liens de précédence et F aux liens de conséquence. Les buts, décrits sous la forme d'une liste de propositions, sont ensuite connectés au réseau d'actions. Ils deviennent des émetteurs d'énergie positive. Cette énergie est ensuite répandue dans le réseau, par l'intermédiaire des liens qui viennent d'être décrits. Les actions dont les pré-conditions sont satisfaites envoient une partie de leur énergie à leurs successeurs. Les actions dont les pré-conditions ne sont pas satisfaites envoient une partie de leur énergie à leur prédécesseur. Enfin, une fraction d'énergie négative est répandue via les liens de conflits. Ces liens sont pondérés par un coefficient stipulant la fraction d'énergie qu'ils laisseront passer ; celui-ci est inversement proportionnel au nombre de liens possédant la même action comme source. Un algorithme itère sur le réseau ainsi construit pour répandre l'énergie dans les

nœuds d'action tout en prenant en compte les éventuels changement dans l'environnement. Lorsque l'activation d'une action dépasse un certain seuil et que ses pré-conditions sont satisfaites, elle est exécutée. Par le système de dispersion, l'énergie générée est concentrée sur les actions permettant de satisfaire les buts. D'autre part, lors de la phase de dispersion, les nouvelles informations acquises sur le monde sont prises en compte et le système est mis à jour en conséquence. Le mélange de ces deux propriétés permet d'obtenir un algorithme de planification réactive, pouvant exploiter les opportunités offertes par l'environnement dès que celles-ci sont perçues.

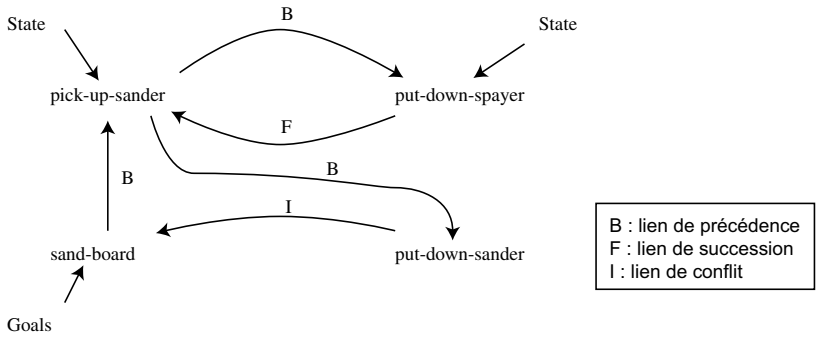


FIG. 1.11 – *Exemple réseau de sélection d'actions suivant l'approche de Maes.*

Cependant, dans le cadre de la spécification de plusieurs buts, le mécanisme montre ses limites [Tyr94]. Le système peut osciller entre plusieurs actions sans jamais effectuer de choix définitif. Dans ce cas, il s'avère incapable de trouver une solution au problème. D'autre part, le mode de calcul de la dispersion d'énergie défavorise les plans ayant trop de possibilités de réalisation, au profit de ceux possédant peu de degrés de liberté. Par ce biais, des buts, qui peuvent être considérés comme secondaires, peuvent être favorisés au détriment de buts plus importants.

Rhodes, pour pallier certains défauts du système, propose une extension de ce mécanisme de sélection d'actions dénommée PHISH-Nets [14] [Rho96]. D'une part, dans la description des actions, il autorise l'utilisation d'un paramètre. Lors de la création du réseau, le paramètre est déterminé pour permettre la construction de liens de précédence entre les actions. La phase de dispersion d'énergie a été modifiée : elle prend désormais en compte une notion de distance entre le but à atteindre et l'action, et, pour pallier le problème de la sur-division de l'énergie, un mécanisme d'identification de la source l'ayant produite a été ajouté. Ce système gagne en facilité de description par l'introduction du paramètre dans les actions et évite de défavoriser les plans possédant beaucoup de possibilités de réalisation.

Decugis et Ferber proposent une autre forme de mécanisme de sélection d'action[DF98]. Les nœuds du réseau sont de deux types :

- les nœuds de type perception qui représentent les faits décrivant l'environnement. Ils possèdent une valeur de vérité permettant de savoir si le fait est présent ou non dans l'environnement.
- les nœuds de type comportement qui correspondent à une action exécutable.

14. Planning Heuristically In Situated Hybrid Networks

Pour créer un réseau, ces nœuds sont connectés les uns aux autres par l'intermédiaire de deux types de liens :

- les liens perception-action, qui correspondent à la spécification de préconditions pour l'exécution de l'action. L'action ne peut en effet être exécutée que si les nœuds perception sont vrais.
- les liens action-perception, qui correspondent à la spécification des conséquences d'une action.

Ces liens, comme dans le mécanisme de Maes, possèdent un facteur, dans l'intervalle $[0; 1]$, permettant de stipuler la proportion d'énergie qu'il peut transmettre. Les buts sont alors considérés comme des émetteurs d'énergie pouvant être connectés aux nœuds perception ou action. Dans le premier cas, le but préconise la réalisation d'un fait, dans le second la réalisation d'une action. Les nœuds perception dont la valeur de vérité est vraie sont eux aussi des émetteurs d'énergie. L'énergie est ensuite propagée de deux manières :

- directe : l'énergie est propagée, dans le sens des liens définis, permettant ainsi d'activer les actions dont les pré-conditions sont vraies. La proportion d'énergie diffusée est pondérée par le poids du lien et un facteur associé à la propagation directe.
- indirecte : l'énergie est propagée dans le sens inverse des liens définis, permettant ainsi de prendre en compte l'influence des buts dans l'activation des actions. La proportion d'énergie diffusée est pondérée par le poids du lien et un facteur associé à la propagation indirecte.

Les activations sont ensuite normalisées pour chaque nœud du réseau pour éviter des phénomènes de divergence. Finalement, l'action possédant le plus fort degré d'activation est choisie. Ce système possède les même propriétés que celui de Maes, et donc les mêmes défauts. Mais pour permettre de gérer les buts conflictuels, la notion de hiérarchie de mécanismes de sélection d'action a été introduite. Une action considérée comme atomique à un certain niveau d'abstraction représente en réalité un autre mécanisme de sélection d'action spécialisé dans la résolution d'un problème donné. Cette approche semble être efficace pour éviter les phénomènes d'hésitation et d'oscillation. Cependant, la manière de hiérarchiser les réseaux d'actions n'est pas formalisée et s'avère être le fruit d'expériences. Le problème du choix des nœuds perception à utiliser au plus haut niveau de la hiérarchie n'est pas simple et doit être spécifié à la main par l'utilisateur lors du processus de conception de la hiérarchie.

La comparaison des réseaux d'actions de Maes et de Decugis, sans prendre en compte la notion de hiérarchie, montre une conséquence intéressante du mode de représentation sur le nombre de liens. Supposons qu'un fait décrivant une propriété sur l'environnement soit conséquence de i actions et précondition de j autres actions. Dans le modèle de Maes, ce fait va générer $i \times j$ liens entre actions. Dans le modèle de Decugis, il va générer $i + j$ liens, réduisant ainsi considérablement le nombre de liens présents dans le réseau. Cette réduction améliore les performances du mécanisme de sélection d'action en diminuant le nombre de calculs effectués lors de la propagation de l'énergie.

Les mécanismes de sélection d'actions, malgré des possibilités d'oscillations liées aux buts conflictuels, présentent de très bonnes propriétés pour l'animation comportementale. Le monde est considéré comme dynamique et peut donc changer sans l'intervention de l'agent. Du point de vue de l'agent, ces changements peuvent d'une part remettre en cause la validité d'un plan mais aussi offrir de nouvelles opportunités. Ces mécanismes prennent en compte ces caractéristiques en alliant la réactivité et la planification ; dès qu'un changement est perçu, il est pris en compte dans la sélection d'action.

1.2.5 Agents BDI

Le mode de raisonnement associé aux agents BDI (Beliefs Desires Intentions) s'inspire du raisonnement pratique de l'être humain [Bra87]. La première implémentation de ce type de système, au travers d'une formalisation logique est attribuée à Rao [RG95]. Les initiales BDI réfèrent à trois concepts : les croyances, les désirs et les intentions.

Les croyances. Elles symbolisent la connaissance que l'agent possède de l'état du monde ou plus précisément ce que l'agent croit vrai. Elles sont caractérisées par une incomplétude due aux capacités de perception réduites de l'agent et à l'hypothèse de départ sur la dynamique du monde et l'impossibilité de prévoir son état.

Les désirs. Ils représentent les motivations de l'agent. Cette notion peut être assimilée à un but, mais ne possède pas une formalisation telle que dans le *situation calculus*.

Les intentions. Elles regroupent l'ensemble des plans que l'agent exécute en vue de réaliser ses désirs.

L'hypothèse de départ des systèmes BDI est que l'agent possède un certain nombre de plans, qui fournissent une méthodologie à appliquer pour satisfaire ses désirs. Ces plans possèdent des conditions d'application qui doivent être satisfaites pour leur permettre d'être adoptés comme des intentions. Ces conditions peuvent porter sur divers paramètres tels que l'état du monde ou référer à l'état interne de l'agent. Lorsque l'agent veut satisfaire un désir, il recherche un plan le réalisant dont les conditions d'application sont remplies. Si plusieurs plans peuvent être utilisés, un mécanisme d'arbitrage peut permettre de choisir celui qui semblera le plus adapté.

Lorsqu'un plan devient une intention, il est exécuté par étapes, jusqu'à ce qu'une nouvelle sélection soit requise ; elle peut provenir d'un changement de l'environnement, de la réussite ou de l'échec du plan. Une étape d'un plan peut prendre trois formes :

- l'ajout d'un nouveau désir, qui correspond à la notion de sous-but et permet de raffiner les plans.
- la modification des croyances de l'agent.
- l'exécution d'une action atomique.

Ce système s'avère efficace car il exploite une connaissance experte fournie par l'intermédiaire des plans. L'enchaînement des actions est géré automatiquement au travers de la demande de satisfaction de désirs, qui peuvent alors être assimilés à des sous-buts. Les plans se réfèrent donc à d'autres plans et ce raffinement est effectué jusqu'à l'obtention d'actions atomiques pouvant être exécutées.

Ce mode de sélection permet de rapidement prendre en compte les changement de l'environnement qui valident ou invalident de nouveaux plans, par l'intermédiaire de leur influence sur les conditions d'applicabilité. L'utilisation des désirs pour décrire les sous-buts permet de décrire des plans généraux, qui durant les différentes phases de sélection vont se raffiner en fonction de l'état des connaissances de l'agent et donc de l'environnement. Cependant, l'agent est peu « inventif », il ne peut résoudre des situations imprévues, car il n'est pas doté de capacités de raisonnement sur ses actions. Si un désir ne dispose pas de plan pouvant directement le satisfaire, il est simplement

omis. D'autre part, ce système pose le problème de la compatibilité des désirs [RG91], qui par leur manque de formalisation réelle s'avère difficile à vérifier [TPH02].

Conclusion

Les différents modèles qui viennent d'être présentés permettent de décrire différents aspects du comportement allant de la modélisation de comportements réactifs, à la modélisation de processus de raisonnement en vue de trouver des enchaînements d'actions cohérents pour atteindre un but fixé. La table de la figure 1.12 fournit un résumé des caractéristiques des modèles décisionnels abordés, dans laquelle *réactivité* stipule la rapidité de réaction du système à un changement extérieur, *orienté but* stipule que le système gère explicitement des buts, *connaissance opératoire* stipule l'utilisation d'une connaissance opératoire fournie lors de la description pour aider le système, *niveau d'abstraction* stipule le niveau d'abstraction utilisé lors de la conception du système, *concurrence* la capacité du système à gérer la concurrence de plusieurs actions ou buts et enfin *cohérence temporelle* indique si l'état du système prend en compte son état précédent de manière à assurer une continuité dans le comportement.

type de Modèle	réactivité	orienté but	connaissance opératoire	niveau d'abstraction	concurrence	cohérence temporelle
stimuli réponses	bonne	non	variable	faible	non	faible
base de règles	bonne	non	oui	moyen	oui	moyenne
automates simples	bonne	non	oui	moyen	non	bonne
automates parallèles	bonne	non	oui	moyen	oui et non	bonne
automates hiérarchiques	bonne	non	oui	moyen	oui	bonne
situation calculus	faible	oui	non	très bon	non	bonne
GRAPHLAN/HSP	faible	oui	non	bon	non	bonne
planification HTN	moyenne	oui	oui	bon	non	bonne
Sélection d'actions	moyenne	oui	variable	moyen	oui	bonne
BDI	bonne	oui	oui	bon	oui	bonne

FIG. 1.12 – *Résumé des caractéristiques des modèles décisionnels.*

A l'analyse des système présentés, les approches à base d'automate semblent bien adaptées à la description de systèmes réactifs. La raison est qu'elles permettent de prendre en compte rapidement les changements de l'environnement tout en offrant un formalisme permettant d'assurer une continuité temporelle du comportement (par la notion d'état courant de l'automate), et en permettant d'effectuer des choix spécifiques à chaque états caractérisant le système. D'autre part, elle offre un certain niveau d'abstraction pour la manipulation des comportements, tout en ne restreignant pas le champ d'expression de l'utilisateur.

En terme de systèmes gérant des notions de buts, les approches sont diverses et possèdent toutes leurs avantages et inconvénients. Le formalisme associé au *situation calculus* permet de prendre en compte beaucoup d'informations sur l'environnement et offre un niveau d'abstraction supplémentaire en permettant de raisonner sur la connaissance. Cette propriété est importante pour les agents situés, qui ne possèdent qu'une perception, et donc une connaissance, partielle de leur environnement. Cependant, les calculs associés peuvent s'avérer très coûteux et ne pas forcément permettre d'obtenir une certaine réactivité, nécessaire pour les agents. La sélection d'action, même si elle souffre de problèmes d'instabilité, tente d'offrir un compromis entre réactivité et planification. Certes, le

niveau d'abstraction est inférieur mais la notion de but reste présente et la réactivité aux changements de l'environnement est bonne. La planification HTN présente un intérêt de par son approche hiérarchique, la décomposition des tâches lui permet de garder une certaine réactivité par une planification relativement rapide. D'autre part, elle permet d'inclure une connaissance experte, biaisant le processus de raisonnement. Il s'agit d'une qualité car elle permet de traduire une cohérence dans l'enchaînement des tâches, mais aussi un défaut car elle nécessite une conception particulière. En effet les actions sont dépendantes de tâches mais pas directement des faits décrivant l'environnement lors de la planification. Cela rend donc la description plus complexe que pour le situation calculus ou le formalisme STRIPS qui se basent sur une description indépendante des actions. Les systèmes BDI souffrent du même problème, ils se basent sur une description exhaustive des plans permettant de satisfaire les désirs (ou buts). Ils ne raisonnent pas mais réagissent à leurs désirs et aux modifications de l'environnement. La stratégie adoptée est alors de produire des désirs pour forcer une orientation vers un but. Cependant, leur grande qualité réside dans leur bonne réactivité face aux changements perçus.

En conclusion, un système adapté à la simulation du comportement semble devoir, en corrélation avec l'approche de Newell, posséder une architecture réactive et une architecture cognitive. L'architecture réactive permet de rapidement réagir à l'environnement, mais doit être contrôlée par un processus plus lourd en calculs, manipulant les connaissances et permettant d'enchaîner des comportements de manière cohérente en vue de réaliser un but de plus haut niveau. D'autre part, pour mieux reproduire le comportement humain, l'architecture réactive doit inclure des notions de parallélisme. Cela permet de gérer l'exécution simultanée de plusieurs tâches, à la fois en réaction à l'environnement, et donc sans passage par un processus cognitif, et à la fois en action sur l'environnement et donc émanant du processus cognitif. Cette notion implique donc l'existence d'un système à même de mélanger les comportements mais offrant une abstraction suffisante pour ne pas avoir à prendre en compte la gestion de la concurrence des comportements réactifs dans le processus cognitif.

Chapitre 2

Représentation de l'environnement et navigation

La possibilité de se déplacer est la condition première de l'autonomie. Le déplacement permet aux formes de vie animales d'être actives dans leur environnement, de survivre, de chasser pour se nourrir... Il permet aussi de mieux appréhender ce même environnement ; de se placer pour mieux percevoir et donc, indirectement, de mieux comprendre les mécanismes qui régissent le monde. En animation comportementale, le déplacement d'un humanoïde revêt la même importance. Il est à la base d'un grand nombre d'activités traduisant des comportements d'interaction avec l'environnement : l'humanoïde se déplace pour prendre quelque chose, pour rejoindre un autre humanoïde... Dans ce chapitre, nous allons nous focaliser sur le comportement de navigation entre deux points de l'environnement avec évitement d'obstacles.

Deux choses contraignent le déplacement : la structure intrinsèque de l'environnement et les autres entités peuplant ce même environnement. La structure de l'environnement représente l'ensemble des obstacles statiques qui contraignent la navigation en bouchant des passages directs d'un point à un autre. Une grande densité d'obstacles contraint donc grandement la navigation et rend ce processus complexe à gérer. Un humanoïde a donc besoin d'une structure de données représentant l'environnement, qui lui permet de détecter rapidement les obstacles statiques gênant sa progression. D'autre part, dans le cadre d'environnements complexes, tels que les villes, les intérieurs de maisons ou d'immeubles, il doit pouvoir raisonner sur leur structure pour trouver un chemin, exempt d'obstacles, lui permettant d'atteindre un but fixé. Enfin, lors de la navigation, les autres humanoïdes jouent eux aussi le rôle d'obstacles. L'entité doit donc être capable de détecter, dynamiquement, les éventuelles collisions avec ses voisins pour pouvoir changer sa route en vue de les éviter. Ces points soulèvent donc deux problèmes de détection de collision : l'un statique, inhérent à la structure de l'environnement, l'autre dynamique, dépendant des autres humanoïdes en déplacement.

Dans ce chapitre, nous allons étudier ces aspects de la navigation. Dans un premier temps, nous présenterons des structures de données permettant de représenter l'environnement. Leur rôle est double, d'une part elles permettent de détecter rapidement les obstacles proches d'une entité, d'autre part, elles fournissent une structure permettant de raisonner pour planifier un chemin. Dans la continuité, nous étudierons donc les méthodes généralement utilisées pour calculer ces chemins. Enfin, nous aborderons les modèles de comportement spécifiques à la navigation, ainsi que les méthodes utilisées pour détecter les collisions éventuelles.

2.1 Représentation de l'environnement

L'environnement dans lequel les entités évoluent est représenté par une géométrie statique. Cette géométrie traduit la structure de l'environnement et donc les contraintes imposées lors de la navigation. De manière générale, cette géométrie représente les obstacles sous la forme de polygones en 2d et sous la forme de polyèdres en 3d. Cette hypothèse sert de base à un certain nombre de méthodes de représentation de l'espace et s'avère être corrélée avec les méthodes de modélisation d'environnement. Que cet environnement ait été produit avec un logiciel de modélisation 3d ou un outil dédié, pour permettre une interprétation rapide du point de vue des entités le peuplant, il doit être représenté dans une structure de données appropriée. Dans ce domaine, plusieurs méthodes ont été proposées dans le monde de la robotique [Lat91]. Ces méthodes consistent à discrétiser l'espace de manière à identifier les zones dans lesquelles l'entité peut naviguer. Cette discrétisation peut ensuite être utilisée pour représenter la topologie des lieux au travers de la notion d'accessibilité entre zones. Cette propriété s'avère nécessaire pour la recherche d'un chemin entre deux points.

Nous allons étudier deux méthodes : d'une part la décomposition en cellules, qui peut être approximative ou exacte, et d'autre part la construction de cartes de cheminement.

2.1.1 Décomposition en cellules

Cette classe de méthode de discrétisation cherche à extraire une carte topologique de l'environnement en décomposant l'espace en cellules représentant des nœuds topologiques. Une fois cette décomposition calculée, un graphe topologique peut être extrait ; chaque nœud représente une cellule alors que chaque arc représente une relation de connexité et d'accessibilité entre deux cellules. Deux types d'approches sont alors à distinguer, d'une part la décomposition approximative qui recouvre partiellement l'espace de navigation et d'autre part les méthodes exactes qui recouvrent exactement l'espace de navigation.

2.1.1.1 Approximation de l'espace libre

Les méthodes d'approximation de l'espace libre se basent généralement sur une discrétisation en cellules de forme fixée telle qu'un carré ou rectangle en deux dimensions ou sous la forme de cubes ou parallélépipèdes en trois dimensions.

Grilles uniformes. Le méthode de décomposition en grille uniforme se base sur la notion de cellules carrées en deux dimensions et cubiques en trois dimensions. Le principe est donc de quadriller l'espace, de manière uniforme, avec ces cellules. Les cellules sont alors vides si elles appartiennent à la zone de navigation, partiellement obstruées si elles contiennent une partie d'un obstacle et obstruées si elles sont inclues à l'intérieur d'un obstacle (voir fig. 2.1).

La précision de ce mode de représentation de l'environnement dépend de la taille de la cellule de base. Plus les cellules sont grandes moins la représentation est précise et inversement. Supposons que l'environnement soit un carré de n cellules de coté, la taille mémoire nécessaire pour le stockage de la grille est en $O(n^2)$; plus généralement, en dimension d, le nombre de cellules est n^d. La complexité mémoire constitue l'un des problèmes majeurs de cette méthode pour laquelle il faut trouver un compromis précision/taille mémoire dépendant de chaque environnement à représenter. Outre son impact sur le coût mémoire, l'augmentation de précision grossit d'autant la taille du

graphe topologique et possède donc un impact non négligeable sur le coût de la recherche d'un chemin [TB96].

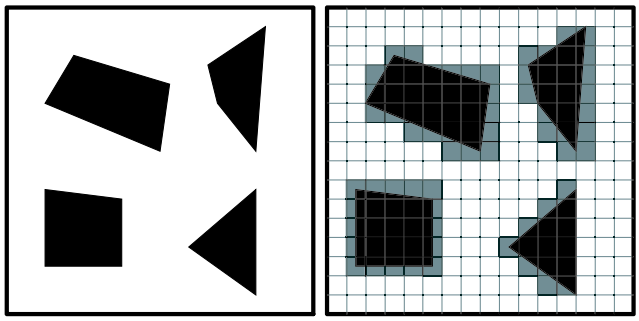


FIG. 2.1 – *Discrétisation de l'espace (à gauche) avec une grille régulière (à droite). Les cases grisées représentent les cases partiellement obstruées par un obstacle (en noir)*

Dans le cas d'une représentation $2d/2d^{\frac{1}{2}}$ les possibilités des cartes graphiques actuelles peuvent être utilisées pour calculer cette carte en une passe de rendu, en utilisant peu de temps processeur [Kuf98]. La méthode consiste à effectuer un rendu de la scène en vue de dessus avec une projection orthogonale. L'image rendue, couplée à l'information de Z-Buffer permet alors d'obtenir une représentation de l'environnement sous la forme d'une grille régulière en $2d^{\frac{1}{2}}$ en identifiant rapidement les zones de navigation et les obstacles. Son utilisation en $2d/2d^{\frac{1}{2}}$ dans le cadre de l'animation comportementale est assez répandue [Kuf98, TC00] dans la mesure où elle est simple d'utilisation et très rapide en terme d'accès. En trois dimensions, elle permet à des humanoïdes de synthèse de naviguer dans des environnements assez complexes [BT98] en prenant automatiquement en compte les informations de hauteur dans la planification. Elle simplifie le problème de la gestion des pieds d'appui car la sélection des cellules représentant le sol lors de la planification fournit toute l'information nécessaire.

Grilles hiérarchiques. Le principe des grilles hiérarchiques consiste à décrire l'espace sous forme d'un arbre de cellules qui subdivisent récursivement l'espace. Dans le cas en deux dimensions, le *quad-tree* (voir fig. 2.2) subdivise récursivement l'espace en décrivant une cellule carrée à un niveau comme la réunion de quatre cellules carrées disjointes recouvrant tout l'espace de la cellule mère. Dans le cas en trois dimensions, l'*octree* fait de même avec une forme de base cubique subdivisée en huit sous cubes. Lors de la phase d'initialisation de cette structure, une subdivision s'arrête à un niveau donné de l'arbre si la profondeur de la cellule est égale à la profondeur maximale de l'arbre (directement corrélée à la taille maximale de la cellule la plus précise) ou si toutes les sous cellules sont libres pour la navigation ou toutes à l'intérieur d'un obstacle. En terme d'occupation mémoire, au pire cas il est du même ordre de grandeur que pour les grilles régulières. Dans les faits, elle s'avère souvent moins coûteuse pour les environnements peu denses en obstacles, ce qui explique, en partie son utilisation en robotique [YSSB98].

Le graphe topologique associé à ce type de représentation est du même type que pour la grille régulière mais est uniquement constitué des feuilles de l'arbre (*quad-tree/octree*). Au pire, la taille du graphe est donc égale à celle du graphe associé à la grille régulière. Cependant, dans un très grand nombre de cas, la précision de la grille régulière est augmentée pour obtenir une meilleure approximation de la géométrie des obstacles. Cette augmentation du nombre de cellules est uniforme

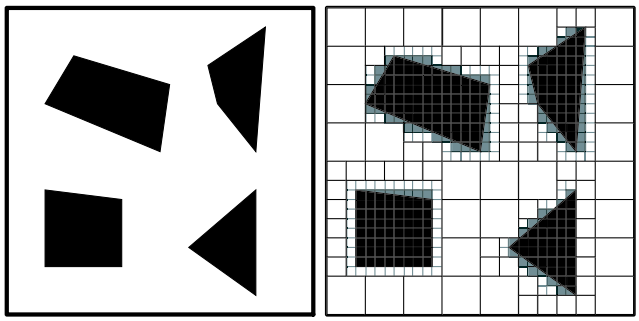


FIG. 2.2 – *Discrétisation de l'espace (à gauche) avec une grille hiérarchique type* ***quadtree*** *(à droite). Les cases grisées représentent les cases partiellement obstruées par un obstacle (en noir)*

sur toute la carte, la même augmentation de précision dans une grille hiérarchique, d'après le mode de construction n'augmente la précision que sur le bord des obstacles en conservant des cellules les plus grandes possibles pour des zones homogènes. L'utilisation de cette structure permet donc de réduire considérablement, par rapport à la grille régulière, le nombre de cellules et donc la taille du graphe topologique associé.

Décomposition par rectangles. Cette décomposition représente l'espace sous la forme d'une collection de rectangles, alignés sur les axes, de taille différente [Lat91]. Ces rectangles recouvrent tout l'espace et ne partagent que des frontières, sans se recouvrir. Au même titre que dans les méthodes précédentes, ces rectangles (ou cellules) représentent une zone de navigation libre, une zone contenue à l'intérieur d'un obstacle ou bien une zone comprenant à la fois une zone de navigation et une partie d'obstacle. La figure 2.3 montre cette décomposition sur l'environnement déjà utilisé précédemment, pour la comparaison, cette subdivision a été générée avec la même précision que pour le quadtree de la figure 2.2.

Par rapport aux méthodes précédentes, ce type de décomposition apporte l'avantage de permettre de créer des cellules de taille variable en fonction de la complexité géométrique de l'environnement. La grille régulière force une discrétisation en cellules de taille fixe, la décomposition hiérarchique permet d'utiliser un facteur d'échelle (fois deux sur chaque dimension à chaque niveau de l'arbre) sur la taille de la cellule alors que la décomposition en rectangle permet de régler la taille en x et en y de chacune des cellules. Elle permet donc d'essayer de maximiser la taille des cellules représentant un espace homogène, permettant ainsi de diminuer la taille de la structure de données utilisée pour représenter l'espace.

2.1.1.2 Décomposition exacte

Les méthodes de décomposition exacte ont pour but de représenter exactement l'espace libre, généralement sous la forme de cellules convexes de différentes formes (triangles, polygones, trapèzes...). Au cours des explications qui vont suivre, nous allons nous contenter du cas de la représentation dans un espace à deux dimensions. Il est cependant à noter, que l'extension de ces méthodes en 3d existe.

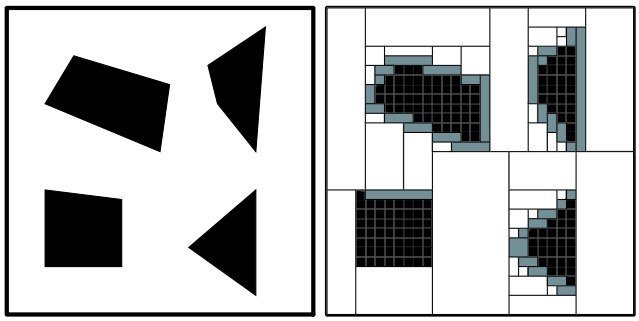


FIG. 2.3 – *Décomposition par rectangles*

Triangulation de Delaunay contrainte en $2d$**.** La triangulation de Delaunay contrainte permet de représenter l'espace sous la forme d'une collection de triangles dont les bords peuvent être de deux types : contraints ou ajoutés par la triangulation. Avant de donner une définition de la triangulation de Delaunay contrainte, nous allons exposer les propriétés de la triangulation de Delaunay pour ensuite voir comment celles-ci peuvent être modifiées pour en faire une triangulation contrainte.

La triangulation de Delaunay [BY95] prend en entrée un ensemble P de points du plan et fournit en sortie un ensemble de triangles dont les sommets sont formés par les points fournis en entrée. Ces triangles respectent la contrainte suivante : le cercle circonscrit au triangle ne contient pas de points de P autre que les sommets de ce même triangle. Cette triangulation peut être construite par le biais d'algorithmes ayant une complexité de l'ordre de $O(n \ln n)$, pour n points fournis en entrée, s'appuyant soit sur la contrainte de cercle, soit sur une propriété angulaire. La figure 2.4 montre un quadrilatère (A,B,C,D) convexe qui peut être triangulé de deux manières. La triangulation maximisant l'angle minimum des deux triangles est dite de Delaunay. Autrement dit, en se basant sur la figure 2.4, la triangulation de droite est une triangulation de Delaunay car $min(a,b_1,b_2,c,d_1,d_2) > min(a_1,a_2,b,c_1,c_2,d)$. Cette règle peut être utilisée localement pour transformer une triangulation en triangulation de Delaunay, la pré-condition d'application étant que l'union des deux triangles considérés crée un quadrilatère convexe. Une autre conséquence très intéressante de cette triangulation est que chaque point est relié à son plus proche voisin par une arête d'un triangle. Enfin, dans le cas d'un ensemble de points ne contenant pas plus de trois points co-circulaires, cette triangulation est unique.

Dans le cas de la subdivision spatiale, il est utile de pouvoir spécifier qu'un ensemble d'arêtes doivent être présentes dans la triangulation finale, sans que celles-ci respectent les contraintes de la triangulation de Delaunay. Dans ce cas précis, la triangulation de Delaunay contrainte [Che87] peut être utilisée. Cette triangulation modifie la contrainte du cercle circonscrit de la triangulation de Delaunay, qui devient : tout point de P inclus dans le cercle circonscrit à un triangle ne peut être relié à tous les points de ce même triangle sans entrer en intersection avec une contrainte. L'ajout de cette notion peut s'apparenter à la notion de visibilité. Si l'on considère que les arêtes contraintes coupent la visibilité d'un point à un autre, alors la propriété portant sur le plus proche voisin de la triangulation de Delaunay, devient dans le cadre de la triangulation de Delaunay contrainte : chaque point est relié à son plus proche voisin visible.

Cette triangulation peut être utilisée pour effectuer une subdivision spatiale en cellules triangulaires [KBT03]. Les contraintes expriment alors les arêtes des polygones délimitant les obstacles peuplant les environnements, permettant d'effectuer une subdivision de l'environnement sous la

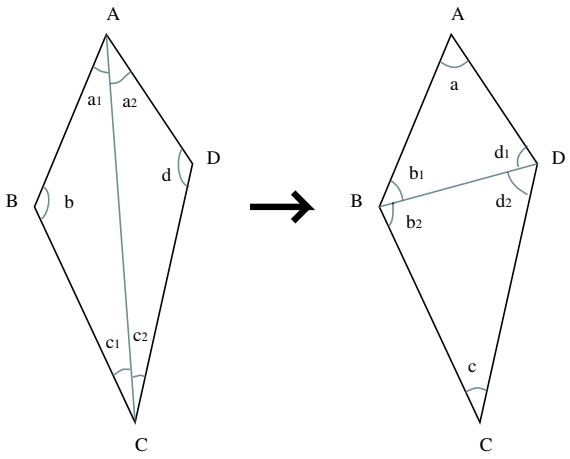


FIG. 2.4 – *Contrainte angulaire de la triangulation de Delaunay : régulation locale.*

forme de triangles (voir fig. 2.5). Le nombre d'arêtes et de triangles alors générés respectent la relation d'Euler [BY95]. Pour simplifier l'expression, nous allons nous placer dans le cas où un triangle englobe l'espace contenant les points de la triangulation. Le nombre de points à trianguler est alors $n + 3$, n correspondant au nombre de points fournis et 3 au nombre de points constituant le triangle englobant. Dans ce cas, le nombre d'arêtes contenue par la triangulation est de $3(n + 1)$ et le nombre de triangles est de $2n + 1$. Le nombre de triangles utilisés pour la subdivision spatiale est donc linéaire en fonction du nombre de points, ce qui constitue une très bonne propriété par rapport aux méthodes de discrétisation approximative pour lesquelles le nombre de cellules utilisées n'est pas fonction de la complexité géométrique mais plutôt de la précision de la représentation.

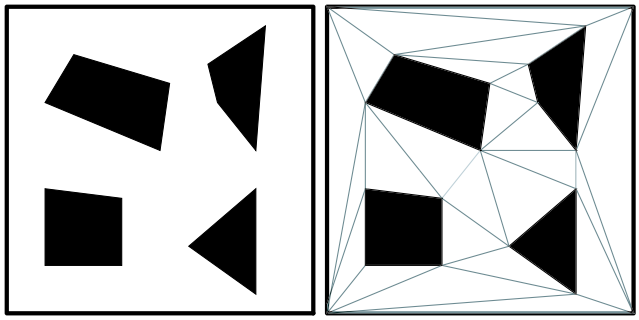


FIG. 2.5 – *Discrétisation de l'espace (à gauche) avec une triangulation de Delaunay contrainte (à droite).*

Décomposition en trapèzes. Cette décomposition permet de décomposer l'environnement sous forme de cellules trapézoïdales [Lat91]. Elle est basée sur un algorithme de balayage [BY95]. Les points délimitant la géométrie de l'environnement sont triés suivant l'axe des ordonnées. Puis un maximum de deux segments est généré, ayant pour origine le point sélectionné et pour extrémité la prochaine intersection avec le bord d'un polygone délimitant un obstacle (voir fig. 2.6).

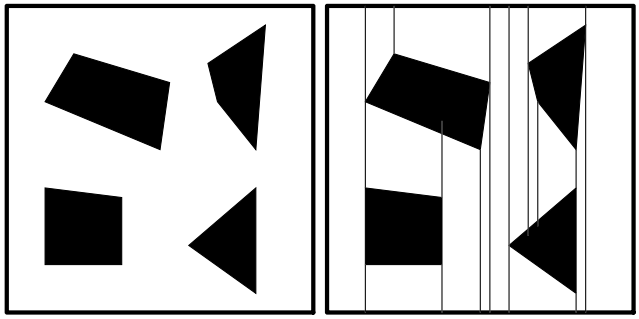


FIG. 2.6 – *Décomposition en trapèzes.*

L'algorithme de subdivision possède une complexité de $O(n \ln n)$, n étant le nombre de points utilisés pour décrire la géométrie des obstacles. Le nombre de cellules alors générées est en $O(n)$, tout comme le nombre de segments reliant ces cellules.

La principale qualité des méthodes de discrétisation exactes réside dans le nombre de cellules qu'elles génèrent. Ce nombre est linéaire en fonction du nombre de points utilisés pour décrire la forme des obstacles. Il ne dépend donc pas de la précision de représentation voulue, mais uniquement de la complexité intrinsèque de la géométrie représentant l'environnement. La deuxième propriété se situe dans l'adaptation automatique de la taille des cellules à la densité des obstacles. Ces méthodes combinent donc les avantages des discrétisations hiérarchiques et par rectangle tout en conservant l'information géométrique exacte.

2.1.2 Cartes de cheminement

Les méthodes à base de cartes de cheminement discrétisent l'espace sous la forme d'un réseau de chemins permettant aux entités de naviguer en évitant les obstacles. Pour ce faire, un certain nombre de points clefs sont répartis à l'intérieur de l'environnement et connectés s'il existe un chemin en ligne droite les reliant sans rencontrer d'obstacle. Différentes méthodes, basées sur ce concept existent. Elles diffèrent principalement par le mode de génération des points clefs et par la façon de les relier.

Graphe de visibilité. Le graphe de visibilité utilise les sommets des polygones décrivant les obstacles de l'environnement, comme points clefs de la carte de cheminement. Par la suite, deux points clefs sont reliés si et seulement si ils sont mutuellement visibles. Autrement dit si et seulement si il existe un chemin les reliant en ligne droite et exempt d'obstacle (voir fig. 2.7).

Le réseau de chemins ainsi créé possède la propriété de maximiser la longueur des parcours en ligne droite, permettant ainsi de minimiser la distance parcourue [ACF01]. Cependant, la taille du graphe généré est directement dépendante du nombre de paires de points mutuellement visibles. Considérons que dans l'environnement, n points soient mutuellement visibles. Chacun de ces points est relié aux $n-1$ autres points. En considérant que le graphe est non orienté, ces n points gérèrent donc $\frac{n(n-1)}{2}$ liens de visibilité. Un environnement ouvert, de grande taille, avec des obstacles ponctuels disparates, possédant donc un grand nombre de relations de visibilité, constitue l'un des pires cas pour la construction de ce type de graphe de cheminement.

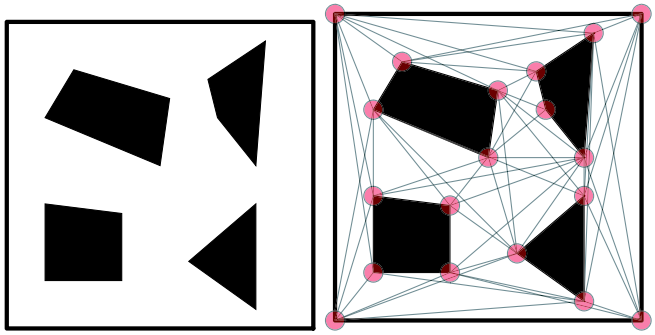


FIG. 2.7 – *Graphe de visibilité (à droite) calculé sur l'environnement (à gauche). Chaque cercle (rouge) représente un sommet de la géométrie de l'environnement et chaque segment une relation de visibilité entre ces sommets. Il est à noter que les bordures des obstacles appartiennent au graphe de visibilité.*

Diagramme de Voronoï généralisé. Le diagramme de Voronoï généralisé est construit sur la base d'un ensemble de sites notés A_1,A_2,...,A_n de différentes formes. Pour chaque point de l'espace, noté p, $dist(p,A_k)$ renvoie la distance de p au site A_k. La région de l'espace où A_i domine A_j est définie comme l'ensemble des points de l'espace plus proches de A_i que de A_j :

$$Dom(A_i,A_j) = \{p \mid dist(p,A_i) \leq dist(p,A_j)\}$$

La région de Voronoï associée au site A_i est alors définie par :

$$V(A_i) = \bigcap_{i \neq j} Dom(A_i,A_j)$$

La partition de l'espace en $V(A_1)$,$V(A_2)$,...,$V(A_n)$ constitue le diagramme de Voronoï généralisé. Les frontières des zones $V(A_k)$ ont des formes qui varient en fonction de la nature du site A_k. Généralement, il s'agit de portions de ligne droites reliées par des courbes. La calcul du diagramme de Voronoï généralisé peut s'avérer complexe, cependant des méthodes utilisant les cartes graphiques, permettent d'en calculer rapidement [HKL$^+$99] une approximation.

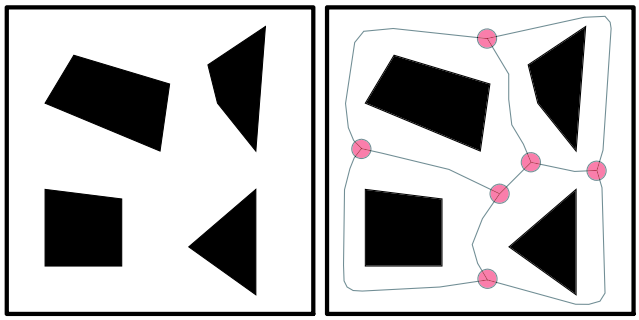


FIG. 2.8 – *Diagramme de Voronoï généralisé.*

Dans le cadre de la génération de cartes de cheminement[Lat91], chaque obstacle de l'environnement constitue un site A_i. Les frontières de chaque zones $V(A_i)$ constituent alors la carte de

cheminement (voir fig. 2.8). Les points clefs sont définis comme étant les point équidistants d'au moins trois obstacles, donc à l'intersection d'au moins trois zones. Les chemins alors générés possèdent la bonne propriété de maximiser la distance aux obstacles.

Cartes de cheminement probabilistes. Plutôt que de se baser directement sur les informations géométriques pour construire un carte de cheminement et identifier les points clefs, les cartes de cheminement probabilistes [Ove02] s'appuient sur une génération aléatoire de points clefs pour couvrir l'espace libre. Deux points peuvent ensuite être reliés s'il existe, entre eux, un chemin libre de collision. Ce test constitue l'opération la plus coûteuse. Il exploite la géométrie de l'environnement et de l'entité en déplacement pour vérifier que le chemin généré ne provoque pas de collision.

Cette méthode est très utilisée dans le cadre de la planification de chemin pour des objets poly-articulés tels que des bras [Kuf99] par exemple. Dans le cadre de la locomotion cette technique a permis la génération de mouvement très complexes [CLS03, PLS03] adaptés à la difficulté de l'environnement, tels que des sauts, marcher sur des piliers, se baisser pour éviter une branche...

2.2 Recherche de chemin

Le déplacement des humanoïdes à l'intérieur d'un environnement virtuel est, le plus souvent, lié à la volonté d'atteindre une position particulière. Il s'agit du problème traité par la recherche de chemin : comment trouver un chemin depuis une position donnée allant vers une position voulue, en évitant les obstacles statiques de l'environnement. Pour générer un chemin plausible, l'approche la plus communément adoptée est de rechercher un chemin minimisant certains critères comme la distance, un coût énergétique... Même si cette recherche d'un chemin optimal n'est pas forcément corrélée avec la navigation humaine [WM03], elle permet de générer des chemins plausibles et évitant des détours excessifs qui s'avèrent beaucoup moins réalistes. Deux grands types d'approches peuvent être distingués : les approches à base de graphe et les approches à base de champs de potentiel.

2.2.1 Calcul sur les graphes

L'utilisation de l'algorithmique des graphes pour le calcul d'un chemin nécessite l'utilisation d'une discrétisation de l'espace car un nœud du graphe est assimilé à un point de l'environnement. Les arcs représentent alors une notion d'accessibilité entre deux points et sont valués par une estimation de l'effort à fournir (il s'agit le plus souvent de la distance) pour relier ces points. La recherche d'un chemin va donc se résumer à la recherche d'une suite de nœuds, reliés par des arcs dont la somme des valeurs associées minimise le coût global du chemin.

2.2.1.1 Discrétisation de l'espace et construction du graphe

Avant de pouvoir appliquer les algorithmes de calcul sur les graphes, il faut disposer d'une représentation de l'environnement sous cette forme. Les cartes de cheminement exposées précédemment fournissent directement cette information : chaque point clef de l'environnement devient un nœud alors que chaque arc représente la relation d'accessibilité entre ces nœuds. Dans le cas d'une discrétisation de l'espace sous la forme de cellules, les informations de cellules et de connexité sont utilisées pour générer une carte de cheminement. Traditionnellement, les points clefs sont choisis soit comme étant le centre de la cellule, soit comme appartenant aux bords franchissables de cette même cellule. Les arcs du graphe sont alors construits en fonction des relations de connexité entre les cellules. La

figure 2.9 montre les deux formes de graphes de cheminement qui peuvent être construites en utilisant comme base une triangulation de Delaunay contrainte ; la figure de gauche utilise les centres de gravité des triangles comme points clefs ; la figure de droite utilise le milieu des segments non contraints reliant deux triangles.

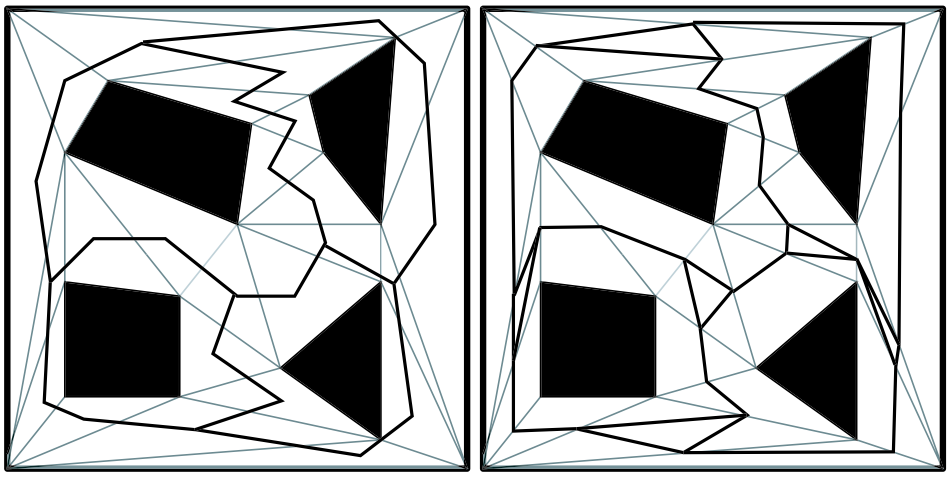


FIG. 2.9 – *Exemple de cartes de cheminement générées à partir d'une discrétisation par triangulation de Delaunay contrainte.*

Une autre approche, utilisée dans le cadre d'environnement intérieurs, consiste à travailler sur le graphe d'adjacence des cellules (extraites de la subdivision spatiale) pour caractériser des espaces (pièces, couloirs, portes) [Lau83, Lau89]. Chaque espace étant un agrégat de cellules, le graphe topologique des espaces est plus abstrait et donc moins complexe que le précédent. Cela permet de définir une stratégie de navigation hiérarchique s'appuyant sur le graphe topologique et raffinant les calculs, à posteriori, à l'intérieur de chaque espace.

2.2.1.2 Les algorithmes utilisés

Diverses formes d'algorithmes de calcul sur les graphes peuvent être utilisées pour effectuer un calcul de plus court chemin. Parmi celles-ci l'algorithme de Dijkstra permet de trouver l'ensemble des meilleurs chemins issus d'un sommet et permettant d'atteindre les autres sommets du graphe. Schématiquement, cet algorithme stocke pour chaque nœud exploré sa distance par rapport à l'origine et son prédécesseur dans le chemin ayant permis l'obtention de cette distance. L'algorithme commence par le nœud d'origine en explorant successivement tous les successeurs tout en privilégiant ceux étant les plus proches de l'origine. Cet algorithme travaille uniquement sur la structure du graphe sans utiliser d'informations supplémentaires. De fait, pour trouver le plus court chemin d'un point a à un point b, sachant que la longueur réelle du chemin de a à b est d, il va explorer tous les nœuds d'une distance inférieure à d avant de pouvoir fournir un chemin.

Dans la mesure où dans les problèmes de planification de chemin, il est possible d'exploiter des propriétés sur l'environnement, l'algorithme A^* lui est souvent préféré. La démarche est légèrement différente, chaque nœud exploré se voit attribuer une valeur correspondant à l'estimation du coût total du chemin passant par ce nœud. Cette valeur est calculée en fonction de la distance réelle par rapport à l'origine plus une estimation (fournie par une heuristique) de sa distance au but. La

valeur $v(n_k)$ associée au nœud n_k a alors la valeur suivante :

$$v(n_k) = d(n_k,o) + h(n_k,b)$$

où $d(n_k,o)$ représente la distance réellement calculée entre l'origine et le sommet n_k et $h(n_k,b)$ la distance estimée (heuristique) du but. L'algorithme s'initialise avec le nœud de départ et stocke dans une queue triée l'ensemble des nœuds successeurs. L'algorithme fonctionne ensuite en gardant une queue triée de nœuds (suivant l'ordre décroissant de la distance estimée au but), que l'on peut qualifer d'ouverts car ils n'ont pas encore été explorés, explorant systématiquement le nœud promettant d'obtenir le plus court chemin. La puissance de convergence de cet algorithme dépend de la qualité de l'heuristique h. Dans le cas de la recherche du plus court chemin dans un environnement, cette heuristique est le plus souvent une estimation de la distance. Un certain nombre de variantes existent par rapport à l'algorithme A^*. Parmi celles-ci l'algorithme ABC (A^* with Bounded Cost) est proposé par Logan [LA98]. Il s'agit d'une généralisation de l'algorithme A^* permettant d'ajouter des contraintes molles à respecter (contraintes de limitation de temps et d'énergie par exemple) lors de la planification.

L'algorithme IDA^* (Iterative-Deepening A^*) utilise une approche différente en effectuant une recherche en profondeur d'abord, supervisée par une heuristique [Kor85]. Dans un premier temps une borne de longueur de chemin B est initialisée avec la distance estimée au but. L'exploration commence par le nœud d'origine du chemin et explore successivement tous les successeurs, ainsi que les successeurs des successeurs..., de ce nœud. Cette recherche s'arrête dans deux cas : soit le chemin est trouvé, dans ce cas il s'agit du chemin optimal, soit un nœud est trouvé tel que sa distance réelle depuis l'origine sommée à sa distance estimée au but soit supérieure à B. Dans le cas où aucun chemin n'est trouvé durant une exploration, la borne B est remise à jour avec la plus petite estimation de la distance au but trouvée au cours des explorations. Certaines améliorations en terme de temps d'exécution peuvent être obtenues en utilisant un cache de taille fixe des estimations déjà calculées pour les nœuds précédemment explorés [RM94]. Ces estimations permettent un élagage plus rapide de l'arbre de recherche.

En règle générale, l'algorithme de Dijkstra est peu utilisé dans le cadre de la recherche de chemin, car il s'agit d'une domaine dans lequel il est souvent possible d'utiliser une heuristique qui réduit grandement le coût de la recherche. Le choix entre l'algorithme A^* et l'algorithme IDA^* est plus difficile. L'algorithme IDA^* est préféré à l'algorithme A^* dans le cas de domaines de taille exponentielle[1] car la taille de la liste triée conservée par l'algorithme A^* devient elle-même exponentielle [RM94]. Cependant, dans le cadre de la planification de chemin, cette considération n'a pas vraiment de signification si l'on considère que l'on dispose déjà du graphe en mémoire. D'un point de vue théorique, ces deux algorithmes possèdent des complexités équivalentes mais l'algorithme IDA^* nécessite moins de mémoire.

2.2.2 Méthodes à champs de potentiel

Contrairement aux approches utilisant les graphes, les méthodes à base de champs de potentiel ne travaillent pas forcément sur une représentation discrète de l'environnement. Les obstacles sont

1. Le domaine dans lequel est utilisé l'algorithme définit le nombre de nœuds appartenant au graphe. Dans le cadre de certains problèmes de recherche de plus court chemin, comme la résolution du problème du puzzle, le graphe peut être implicite, un arc représentant une modification d'un état, un nœud, une configuration.

vus comme des émetteurs de forces répulsives, alors que le but est un attracteur [Lat91, RKBB94]. Un potentiel est donc défini pour chaque point de l'environnement comme la somme des potentiels de répulsion liés aux obstacles avec le potentiel d'attraction lié au but. L'entité étant localisée dans l'environnement, la direction à prendre pour converger vers son but est alors opposée au gradient de potentiel défini en ce point.

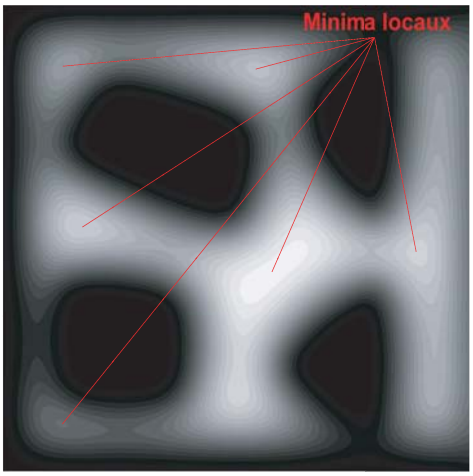


FIG. 2.10 – *Une carte de champ de potentiel, les parties noires représentent les obstacles, les parties plus claires les zones de navigation. Les dégradés de couleurs représentent pour leur part la valeur du potentiel associé au point de l'environnement. Cette image montre six minima locaux caractérisés par des couleurs plus claires.*

Ces méthodes s'avèrent simples et efficaces mais posent le problème des minima locaux (Cf. fig. 2.10). La méthode de navigation associée pousse l'entité à se déplacer vers le minimum local le plus proche qui n'est pas forcément le minimum de la surface et en conséquence qui ne représente pas le but. Pour pallier ce type de problème, des méthodes à base de marche aléatoire sont utilisées. Par exemple, dans RPP (*Random Path Planner*) [BL91], lorsqu'un minimum local est atteint, un ensemble de configurations aléatoires sont tirées puis testée avec une phase de sortie du minimum et une phase de convergence vers le prochain minimum. Les informations sont alors stockées dans un graphe dont les nœuds sont les minima locaux et les arcs traduisent des chemins entre deux minima. D'autre méthodes existent à base de tirage aléatoire de direction à suivre [CRR01], plutôt que de configuration, pour échapper du minimum local.

Ces méthodes ne sont cependant pas très adaptées à l'animation comportementale. Les comportements induits par les tirages aléatoires, s'ils sont acceptables pour des robots, ne le sont pas pour des humanoïdes car ils sont en dehors de la logique de navigation humaine.

2.3 Modèles de simulation de comportement de navigation

La représentation de l'environnement ainsi que la possibilité d'atteindre un but fixé en évitant les obstacles statiques de l'environnement sont suffisants pour gérer la navigation d'un seul piéton à l'intérieur d'un environnement. Dans le cas où plusieurs piétons naviguent, un nouveau facteur est à prendre en compte : l'évitement de collision avec les entités dynamiques. Deux problèmes se

posent alors : comment détecter efficacement les collisions et quelle réaction adopter ? La détection de collision pose le problème de la complexité des calculs ; la réaction à adopter pose le problème du réalisme du comportement. Nous allons aborder cette problématique au travers de la présentation de résultats d'études sur le comportement piétonnier, puis nous présenterons les modèles informatiques dédiés à la simulation de ce comportement.

2.3.1 Résultats d'observation sur le comportement piétonnier

Un certain nombre d'études ont été conduites sur le comportement piétonnier. Dans le domaine de l'étude des flots, des statistiques sont effectuées en vue d'analyser le comportement piétonnier pour assurer, à plus long terme, une certaine qualité de navigation. Au cours de ces études, un certain nombre de caractéristiques de ce comportement ont pu être mises en évidence [Hel01]:

1. Les piétons montrent une certaine aversion à prendre des détours ou à marcher dans une direction opposée à leur direction désirée (celle la menant vers leur but), même dans des environnements peuplés. Cependant, ils ont tendance à choisir la route la plus rapide qui n'est pas forcément la plus courte. En général, les notions de détour et de confort de navigation sont prises en compte, en vue de minimiser l'effort à fournir.
2. Chaque piéton possède sa vitesse individuelle correspondant à la vitesse la plus confortable, autrement dit, celle qui permet de minimiser la dépense d'énergie. Du point de vue de la personne, cette vitesse est corrélée à plusieurs facteurs comme le sexe et l'âge par exemple. Le respect de cette vitesse dépend aussi de facteurs externes tels que la raison du déplacement (la personne est elle pressée ou non ?) ou encore de la densité de la foule. La figure 2.11 résume les résultats observés, en fonction de la densité de la foule, en utilisant une classification par niveau de service. Cette vitesse, que l'on peut qualifier de vitesse de confort, suit une distribution Gaussienne de moyenne 1.34 ms^{-1} avec un écart type de 0.26 ms^{-1}.
3. Les piétons tentent de conserver une certaine distance avec les autres piétons et les obstacles de l'environnement. Cette distance varie en fonction du contexte, elle réduit si le piéton est pressé ou si la foule devient dense. Dans le cas de piétons stationnaires (attente d'un train, bronzage sur la plage...), la tendance est à se répartir uniformément dans l'espace. Cependant, autour des endroits attractifs (statue, monument...), la densité des piétons augmente. Les individus se connaissant peuvent former des groupes ; ces groupes ont alors tendance à exhiber un comportement similaire à celui d'un piéton seul.
4. Dans les situations de panique, les individus ont tendance à développer des comportements "aveugles" et essayent de se déplacer beaucoup plus vite que de normale. Les interactions entre les piétons deviennent alors physiques, allant jusqu'à la bousculade. Les personnes blessées ou tombées constituent alors des « obstacles » gênant la progression. Finalement, un comportement mimétique est mis en œuvre ; les piétons ont tendance à faire comme les autres, à aller où vont les autres, au détriment de l'exploitation d'opportunités de sorties différentes.
5. Lorsque la densité des piétons devient grande, la tendance générale ressemble à l'évolution d'un fluide. Au milieu de foules statiques, on peut remarquer une tendance à créer des flots ; en environnement dense, on peut remarquer la formation de zones de navigation regroupant des piétons allant dans la même direction.

Ces résultats exposent les tendances générales associées à la navigation. D'autres études se sont plus particulièrement intéressées à la manière d'éviter une collision et à la direction adoptée. Selon

Niveau de service	Espace (m^2/piéton)	Flot (piéton/min/m)	Vitesse moyenne (m/s)
A	≥ 12	≤ 7	≥ 1.32
B	$3.7 - 12$	$7 - 23$	$1.27 - 1.32$
C	$2.2 - 3.7$	$23 - 33$	$1.22 - 1.27$
D	$1.4 - 2.2$	$33 - 49$	$1.14 - 1.22$
E	$0.6 - 1.4$	$49 - 82$	$0.76 - 1.14$
F	≤ 0.6	*variable*	≤ 0.76

FIG. 2.11 – *Niveaux de service offerts aux piétons en fonction de leur densité. Cette table met en rapport la vitesse moyenne des piétons avec la densité [MRHA00].*

Goffman [Gof71], en accord avec le point 1 sus-cité, les piétons sont réticents aux changements de direction, et lorsqu'ils y sont obligés, ils privilégient les plus petits changements. C'est ce que l'on qualifie de loi des changements minimaux. Un piéton aura donc tendance à effectuer un évitement de collision en minimisant l'angle de déviation. Cependant, comme le fait remarquer Lee [LW92], l'organisation globale des flux piétonniers est similaire à celle observée pour des véhicules respectant le code de la route. Autrement dit, en Angleterre nous trouverons une tendance à l'évitement à gauche, alors qu'en France, la tendance sera à l'évitement à droite. Dans le cas de situations relativement denses, l'application de cette règle permet d'obtenir un consensus global. Les flots s'organisent, ayant pour conséquence une circulation plus fluide. Des détails supplémentaires sur le comportement de navigation en environnement urbain (traversée de rue, attention visuelle...) peuvent être consultés dans la thèse de G. Thomas [Tho99].

2.3.2 Modélisation à base de particules

Dans un cadre général, les modèles à base de particules cherchent à retranscrire le comportement d'une particule dans un environnement soumis aux lois de la physique. Son comportement suit donc la loi de Newton :

$$\frac{d}{dt}mv = \sum_i F_i$$

où m est la masse de la particule, v sa vitesse et F_i une force provenant de l'environnement appliquée à cette particule. Outre leur utilisation dans des domaines tels que la simulation de fluides ou l'astrophysique, il est apparu que les modèles à base de particules pouvaient être utilisés pour la modélisation du comportement de piétons et donc de foules. Cela provient certainement de l'analogie souvent faite entre le mouvement d'une foule et les mouvements fluides. La modélisation du comportement consiste donc à définir les forces auxquelles un piéton est soumis lors de la navigation.

L'approche de Bouvier [BCN97] consiste à associer à chaque type de piéton une classe de particule. Ces particules peuvent changer d'état en fonction de leur environnement, traduisant ainsi une décision associée au comportement du piéton. Elles peuvent ensuite être soumises à plusieurs champs tels que des champs attracteurs (modélisation de buts) et des champs de décision permettant de changer leur état, et donc de modéliser des comportements dépendants de la localisation. Enfin, en fonction de leur proximité respective, elles exercent des forces de répulsion permettant d'éviter les collisions. Ce modèle a notamment été utilisé pour conduire des simulations de grande envergure telles que la simulation de foules dans le stade de France.

Helbing, pour sa part, s'est attaché à modéliser les interactions liées au comportement piétonnier au travers d'un système de forces socio-psychologiques appliquées aux particules représentant les entités. Ces forces traduisent la tendance à garder une certaine distance avec les autres piétons lors de la circulation ou le comportement de friction associé au contact avec un autre piéton mais tout en prenant en compte une force attirant le piéton vers un but fixé. Ce modèle a été utilisé dans le cadre de la modélisation du comportement de panique [HFV00]. Toujours dans ce cadre et sur la base du modèle d'Helbing, Braun [BMdOB03] a défini l'influence de la connaissance entre les entités lors de situation de panique par l'intermédiaire de forces. Lorsque les entités se connaissent, elle peuvent, en fonction de paramètres d'altruisme et de dépendance, s'entraider pour sortir d'une situation critique ; le but étant de pouvoir étudier l'impact de ce type de comportement sur l'évacuation de lieux sinistrés. La modélisation de ce comportement se fait par l'intermédiaire de forces d'attraction pondérées par un facteur d'altruisme et un facteur de dépendance.

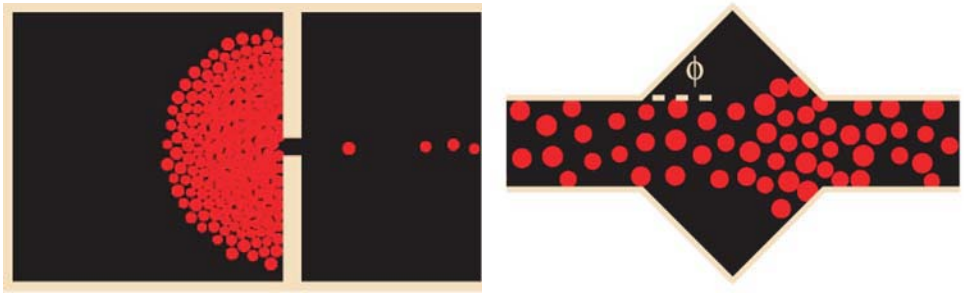


FIG. 2.12 – *Un exemple de comportement de piétons issu des travaux de Helbing.*

Les modèles à base de particules, particulièrement le modèle d'Helbing, exhibent des tendances générales réalistes en terme de comportement (la figure 2.12 montre la répartition des entités dans le cas de bouchons au niveau des goulets d'étranglement). Cependant, elles ne prennent pas en compte un certain nombre de facteurs tels que la perception de l'environnement et les règles sociales [LW92] qui peuvent grandement influer sur le comportement de navigation.

2.3.3 Modélisation comportementale

La modélisation comportementale diffère quelque peu de la modélisation par système de particules dans la mesure où le comportement d'une entité est défini par rapport à un certain nombre de règles, qui, dans certains cas, s'inspirent des études sur le comportement humain. Nous évoquerons plusieurs méthodes : les approches s'appuyant sur la représentation de l'environnement sous la forme d'une grille régulière, les approches plus générales considérant des déplacements continus et enfin, les approches utilisant des environnements informés pour permettre l'exploitation de résultats issus de l'analyse du comportement humain. Dans un dernier temps, nous aborderons la gestion du comportement de groupe.

2.3.3.1 Modèles à base de grilles régulières

PEDFLOW [KKWH01] est un environnement permettant de modéliser le comportement de piétons. Il se base sur une représentation de l'environnement sous la forme d'une grille régulière en

deux dimensions. Chaque cellule de la grille peut contenir un obstacle ou un agent. Le comportement des agents est défini par l'intermédiaire d'un ensemble de règles lui permettant de se déplacer en fonction de ce qu'il perçoit de son environnement, autrement dit, de l'état des cellules voisines. Pour simplifier les ensembles de règles, les variables d'entrée représentant la perception de l'environnement (type d'entité, direction, vitesse, distance, position relative en terme de voie de navigation) sont discrétisées, tout comme le résultat de la règle (direction, vitesse). Une propriété intéressante de ce système réside dans la possibilité de décrire des agents actifs, envoyant des événements aux agents proches pour leur permettre de changer leur comportement. Ceci permet notamment de simuler des points attractifs dans l'environnement (statue, peintre...). Cependant, dans ce système, le déplacement des entités se fait case par case, ce qui permet de traduire un comportement général de navigation mais ne permet pas d'obtenir une animation réaliste de la navigation des piétons.

L'approche adoptée par Dijkstra, au travers des automates cellulaires [DTJ00], est très proche. Les automates cellulaires se basent sur une discrétisation de l'espace (sous la forme de cellules) et du temps. Chaque cellule est active, à chaque pas de temps, elle peut modifier son état en fonction de l'état des cellules avoisinantes. Cette propriété est utilisée dans le système pour calculer localement des densités de piétons. De plus, les cellules sont typées et des cellules particulières, de type décision, possèdent des informations de direction à adopter en fonction des intentions des piétons qui pourront les traverser. Chaque piéton se déplace alors case par case, en prenant des décisions en fonction de l'état de la cellule sur laquelle il se trouve, de la densité de piétons à son voisinage et de l'état des cellules connexes. Le comportement de navigation s'avère cependant simpliste : si un piéton ne peut se déplacer dans la cellule en face de lui car elle est occupée, il teste la cellule en face sur la gauche (ou la droite), si celle-ci est aussi occupée, il s'arrête. La notion de cellule de décision s'avère intéressante, elle permet de configurer l'environnement de simulation en fonction des comportements attendus ou observés des piétons.

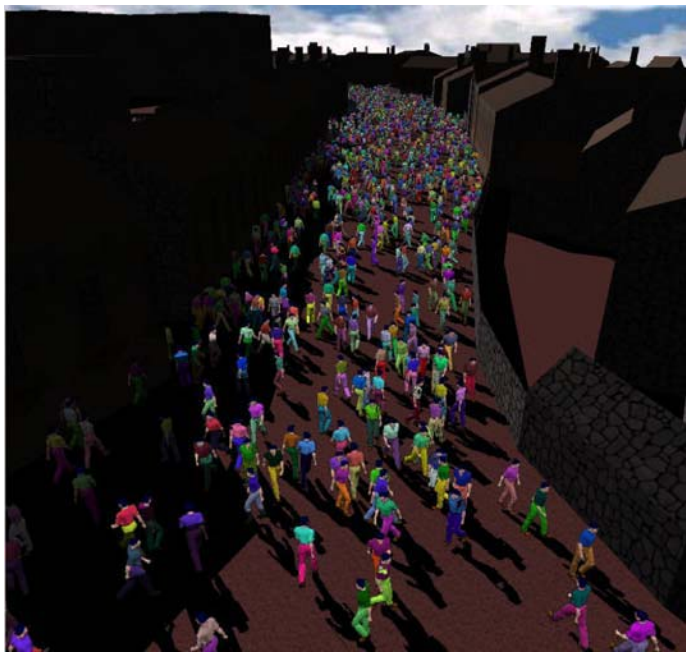


FIG. 2.13 – *Une vue d'une rue peuplée extraite des travaux de Tecchia et Loscos.*

Dans le cadre de la navigation de piétons à l'intérieur d'environnements 3d, Tecchia propose une méthode de détection de collision simple et rapide [TC00]. Dans un premier temps, une carte de l'environnement, contenant des informations de hauteur est calculée en utilisant un rendu 3d de l'environnement en vue de dessus et en projection orthogonale. La carte obtenue est donc une grille régulière en $2d^{\frac{1}{2}}$. Cette grille est utilisée pour prédire les collisions des piétons avec les murs, au travers de l'exploration d'un certain nombre de cellules dans la direction de déplacement. Loscos a étendu ce travail en y incluant des comportements piétonniers plus réalistes [LMM03]. La grille est désormais utilisée pour prédire les collisions avec les autres piétons (avec exploration des cellules connexes et prédiction temporelle) et une table de réaction est utilisée en cas de détection de collision. Cette table contient un certain nombre de réactions typiques, qui prennent en compte le comportement en cas d'embouteillage, les configurations de collision (de face, arrière et perpendiculaire). D'autre part, la structure de grille est utilisée pour stocker une information sur la dernière direction adoptée par un piéton à l'intérieur de cette cellule. L'organisation des piétons en flots se fait alors automatiquement, au travers d'une moyenne entre la vitesse stockée dans la case contenant le piéton et sa vitesse courante (modulo une atténuation temporelle). L'application de ce modèle permet de gérer un très grand nombre de piétons (approximativement 10000 piétons à 10 images seconde sur un pentium à 2Ghz équipé d'une GeForce4 Ti4600) naviguant dans une ville ; une vue d'une rue peuplée est présentée sur la figure 2.13.

Les approches qui viennent d'être décrites, permettent la gestion d'un grand nombre d'agents (particulièrement pour l'approche de Loscos et Tecchia), grandement due à l'utilisation sous jacente de la grille régulière qui facilite les calculs. La détection du voisinage des obstacles (agent ou environnement) s'effectue par l'exploration d'un nombre fixé de cellules autour de l'agent. Cette propriété permet de stabiliser le coût de calcul des détections de collisions, qui devient rapide par l'accès direct fourni par la grille. Cependant, l'utilisation d'une grille limite grandement la taille des environnements qui peuvent être gérés et l'augmentation de la précision possède un impact sur le coût de détection des collisions. Lors de cette augmentation, sans modification de la distance de prédiction, le coût du parcours des cellules voisines augmente proportionnellement au nombre de cellules représentant l'environnement, donc de façon polynomiale. Ces modèles sont dédiés à la simulation d'environnement peuplés, au détriment du comportement de navigation, qui reste relativement simpliste. Il est cependant intéressant de remarquer que la représentation de l'environnement est utilisée pour influer sur le comportement du piéton, cette propriété permet de délocaliser une part du processus décisionnel, allégeant ainsi le coût global de calcul.

2.3.3.2 Modèles généraux

Les modèles que nous allons présenter ne présupposent aucune représentation particulière de l'environnement dans lequel évoluent les humanoïdes. Ils modélisent les réactions des agents à des collisions, mais ne fournissent pas de modèle permettant de maîtriser le coût de la détection de collision au travers d'un calcul de voisinage.

Pour être en mesure de gérer l'interaction entre des entités lors de leur navigation, des méthodes basées sur la planification temporelle peuvent être utilisées. Généralement, elles planifient une trajectoire dans l'espace (x,y,t) où (x,y) représente la position de l'entité et t le temps. La planification d'une trajectoire se réduit alors à un problème en trois dimensions ou le mouvement des objets définit des formes géométriques dans cet espace. Ce type de calcul est utilisé en robotique mais ne reflète pas le comportement d'un piéton. Feurtey s'inspire de cette technique pour construire un

modèle de simulation de comportement piétonnier [Feu00]. Le déplacement d'un piéton est contraint par une vitesse maximale de déplacement. Elle permet de définir un cercle représentant l'ensemble des positions que le piéton peut atteindre au prochain pas de temps de simulation. En effectuant une prédiction sur le mouvement des entités proches, un ensemble de zones triangulaires sont définies sur ce cercle. Elles caractérisent des situations spatiales et temporelles provoquant une collision. Un algorithme d'optimisation est alors utilisé pour trouver une position à l'intérieur de ce cercle. Il minimise une fonction de coût traduisant une préférence dans un comportement de navigation. Cette fonction est définie comme la somme de trois coûts distincts : le coût associé au fait de dévier de la direction de préférence, le coût associé à un changement de direction, le coût associé à un changement de vitesse. Un comportement de navigation se traduit alors par l'association d'un coefficient à chacun de ces coûts. La solution obtenue favorise les réactions de moindre coût et respecte le profil de navigation décrit par les trois paramètres.

Pour traduire la concurrence de plusieurs règles à respecter dans le comportement de navigation, Hostetler propose un système à base de vote [HK02]. La navigation du piéton est définie en utilisant un espace discret de valeurs permettant de traduire une modification de la direction (tourner à gauche, tourner à droite, ne pas tourner) et la vitesse du piéton (accélérer, décélérer, conserver la vitesse). Ainsi, un espace de neuf configurations est défini. Ensuite, un ensemble de modules dotés de la capacité de voter pour ou contre chacune des neuf solutions est défini. Ces modules sont par exemple un module de suivi de trajectoire, le maintien d'une formation pour la navigation en groupe, l'évitement de piétons, l'évitement d'obstacles, le centrage par rapport aux entités environnantes... Ce système permet donc de continuellement essayer d'adopter une réaction tentant de satisfaire au mieux l'ensemble des contraintes en prenant la décision la plus proche du consensus global.

L'approche de Metoyer [MH03] diffère quelque peu des approches présentées jusqu'ici. Les entités possèdent un comportement de navigation par défaut. Dans un deuxième temps, un utilisateur cherchant à reproduire une animation réaliste a la possibilité d'intervenir sur la simulation en spécifiant le type de réaction attendue à un évitement de collision. Ensuite, le contexte de la réaction (position et vitesse relatives des entités, obstructions liées à l'environnement, direction de déplacement souhaitée) est stocké et utilisé dans un modèle bayésien pour déduire des probabilités de réaction face à une situation. Par la suite, lorsqu'une collision potentielle est détectée, le modèle est utilisé pour choisir la réaction la plus probable en fonction de la situation courante. Au fur et à mesure de la phase d'apprentissage, le comportement piétonnier gagne en réalisme, grâce à la supervision par un être humain. Les résultats d'apprentissage peuvent cependant être stockés et utilisés par la suite sans autre intervention humaine.

2.3.3.3 Environnements urbains informés

Les modèles présentés jusqu'alors ne prennent que peu en compte la relation qui existe entre l'environnement et les entités le peuplant. Lorsque l'on parle de navigation en environnement urbain, non seulement plusieurs entités peuvent entrer en interaction, mais leur comportement dépend des modalités offertes par l'environnement ainsi que de leur perception des lieux. Relieu [RQ98] a caractérisé le concept d'*affordance* de Gibson [Gib86] dans le cadre de la navigation urbaine. Ce concept permet de traduire l'influence de la typologie de la zone de déplacement sur le comportement du piéton. Pour permettre l'exploitation de tels résultats, le concept d'environnement urbain informé a été introduit.

Parmi les modes de modélisation, Farenc propose une décomposition hiérarchique de l'environnement [FBT99] en proposant une représentation fournissant de l'information géométrique et

sémantique. La décomposition se fait par l'intermédiaire d'entités spatiales typées (les *ENV*) elles-mêmes constituées d'autres entités spatiales (voir fig. 2.14). Ces entités peuvent représenter des quartiers, des rues, des tronçons de rue, des voies de navigation (route/trottoir), des immeubles... Le découpage hiérarchique et l'information s'effectue durant la phase de conception de l'environnement, en exploitant la possibilité d'utiliser des nœuds pour la décomposition hiérarchique et de les nommer. Pour gérer la navigation et la planification de chemin, les zones de navigation sont informées avec des points d'entrée et de sortie, traduisant ainsi la connexité. Ces informations permettent de construire une carte de cheminement, fournissant aux entités la possibilité de planifier leur chemin à l'intérieur de l'environnement. Le typage des zones permet par exemple de gérer une navigation cohérente au travers de piétons possédant les notions de trottoir et de passage clouté.

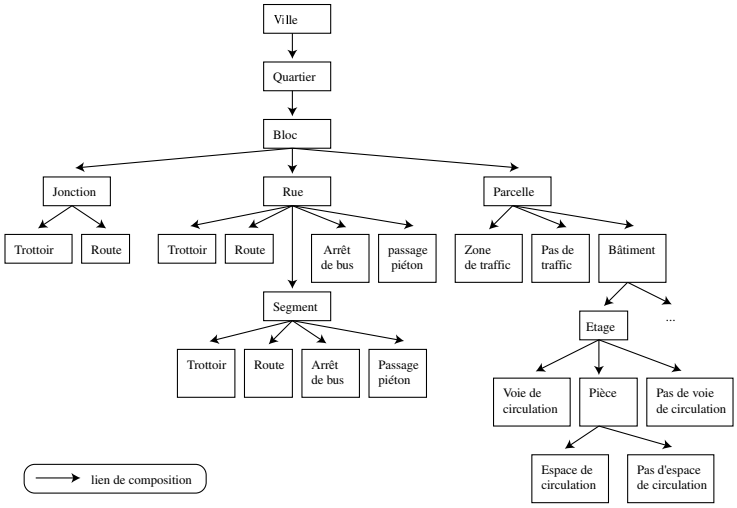


FIG. 2.14 – *Un exemple de décomposition hiérarchique d'une ville.*

Le logiciel VUEMS [2] [Don97] a été initialement conçu pour modéliser des environnements urbains pour la simulation de conduite. Thomas, dans le cadre de ses travaux [TD00a] a étendu ce logiciel pour y inclure des informations sur les zones de circulation piétonnières comme les trottoirs, les passages piétons... La décomposition est hiérarchique, mais les différentes zones, typées, sont ensuite interconnectées pour créer un graphe topologique planaire fortement connexe (voir fig. 2.15). Outre les informations sémantiques, les zones contiennent des informations d'axiales (axiales simples pour les trottoirs, diagramme de Voronoï généralisé pour les espaces libres). Elles permettent de corréler les informations topologiques aux chemins suivis par les humanoïdes lors de la navigation. Le modèle de navigation s'inspire des travaux issus de l'analyse du comportement humain en environnement urbain, et notamment de notre perception de l'espace [RQ98]. Il prend en compte la notion de danger dans la traversée des rues et la notion d'*affordance* spatiale pour gérer la perception des autres entités. De par la représentation typée utilisée dans l'environnement, il est possible de réduire les calculs liés à la perception en exploitant la connaissance sur l'environnement, tout comme les calculs de prédiction de collision. Par exemple, un piéton ne se soucie des voitures que dans le cas d'une traversée de route. Dans le même cadre, une collision entre deux piétons n'est testée que s'ils naviguent sur la même voie de circulation. Lors d'un évitement de collision, deux types de règles sont

2. Virtual Urban Environment Modeling System

prises en compte en fonction de la distance entre les deux piétons [TD00b]. La règle des changements minimaux [Gof71] est utilisée à courte distance, assurant ainsi un minimum d'interaction entre les deux entités. A plus longue distance (une dizaine de mètres), la règle du code de la route est utilisée [LW92].

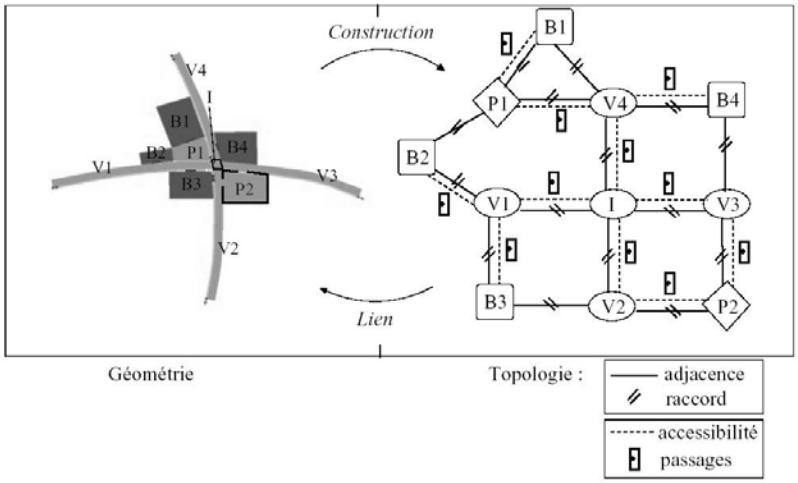


FIG. 2.15 – *Un graphe topologique généré par VUEMS [Tho99].*

Ces modèles permettent de moduler le comportement des humanoïdes en fonction de leur environnement. Le comportement reproduit peut alors être contextuel et s'inspirer d'études sur le comportement humain. Cependant, l'utilisation d'un logiciel de modélisation dédié, ou d'une nomenclature sur les noms, restreint leur utilisation. Ils imposent des contraintes qui ne permettent pas, sans traitement manuel préalable, d'utiliser tous les modèles géométriques disponibles dans les sociétés ou sur internet par exemple. La notion de décomposition hiérarchique offre de bonnes propriétés pour la planification de chemin. Il devient possible, par exemple, d'exploiter la notion de quartier pour effectuer des raccourcis durant la planification, ce qui peut permettre de grandement réduire le coût de calcul.

2.3.3.4 Foules et comportements de groupe

Pour le moment, cette partie de l'état de l'art s'est principalement focalisée sur le comportement d'un piéton par rapport aux autres piétons. Autrement dit, chaque piéton est considéré de manière unitaire. Cependant, les piétons ou plus généralement les entités peuvent naviguer en groupe. Cette notion de groupe a diverses conséquences sur le comportement et notamment la tentative de conservation d'une cohésion.

Pionnier dans ce domaine, Reynolds a proposé un système de gestion de nuées [Rey87] basé sur le respect de trois règles simples, traduisant une modification de la vitesse en fonction de l'influence du voisinage :

- La cohésion : il s'agit d'une règle attirant le membre de la nuée vers le centre de gravité du groupe d'entités voisines.

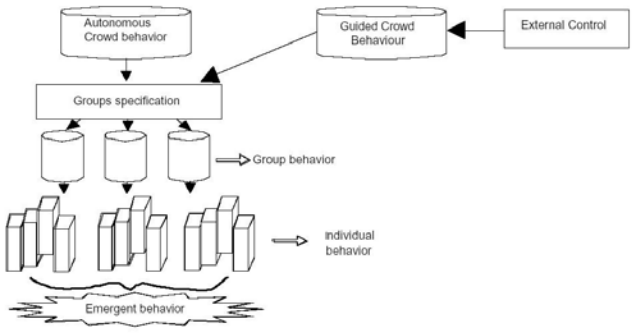


FIG. 2.16 – *Architecture de contrôle des foules issue des travaux de Raupp Musse.*

- La séparation : cette règle permet d'éviter les collisions entre les entités. Dans ce cas, une force repousse l'entité de ses entités voisines pour maintenir une certaine distance.
- L'alignement : cette règle permet d'harmoniser les vitesses des membres d'un groupe. Elle moyenne la vitesse de l'entité avec celle de entités voisines pour permettre d'aligner les déplacements.

Reynolds [Rey99] a ensuite répertorié un certain nombre de comportements qui peuvent être exprimés sous une forme similaire : le suivi de trajectoire, le suivi d'un leader... Ces règles sont largement utilisées pour traduire le comportement d'un groupe doté d'une certaine cohésion. Brogan, par exemple, les utilise pour diriger des groupes d'entités simulées dynamiquement [BH97, BMH98], comme des coureurs cyclistes.

Ces règles sont exprimées en fonction du voisinage. Un algorithme naïf, peut chercher à comparer la position de l'entité avec toutes les entités de l'environnement pour trouver les plus proches. Cette méthode possède une complexité en $O(n^2)$, dans le cas de n entités. Elle est donc difficilement exploitable pour un grand nombre d'entités. Différentes méthodes ont été proposées pour réduire cette complexité. Reynolds propose l'utilisation d'une grille régulière $3d$ [Rey00], permettant ainsi d'exploiter les mêmes propriétés que dans les environnements représentés sous cette forme (voir section 2.3.3.1). Pour sa part, O'Hara préconise l'utilisation de K-d arbres [O'H00]. Cette méthode cherche à construire une structure de données permettant d'accéder rapidement aux plus proches voisins. Le coût de construction et d'accès au voisinage devient alors dépendant du nombre d'entités mais pas de la représentation géométrique de l'environnement. D'autre part, elle permet de contrôler les distances de prise en compte en fonction de la distance réelle des entités voisines et non d'une zone de rayon fixe définie autour de l'entité. En s'appuyant sur cette structure, O'Hara utilise la notion de stabilité des groupes pour réduire le coût en calcul du modèle de nuées [O'H02]. Lorsqu'un groupe est détecté comme stable, autrement dit lorsque les entités sont restées voisines suffisamment longtemps, les calculs liés à la navigation sont effectués pour le groupe complet et non plus entité par entité. Cet algorithme optimise le temps de calcul tout en conservant un comportement cohérent des entités. Cette approche met en évidence les avantages d'une gestion par niveaux de détails du comportement des entités.

Dans le même ordre d'idée, Raupp Musse s'est intéressée à l'adjonction de niveaux de détails dans la gestion de la navigation des groupes d'humanoïdes virtuels. Ceux-ci agissent sur deux points : d'une part, sur la détection de collision et, d'autre part, sur le niveau d'autonomie laissé à la foule. En terme d'évitement de collision, une méthode basée sur la distance à l'observateur est proposée

[MT97]. Deux niveaux de comportement d'évitement sont alors définis en fonction de la distance (l'un plus complexe que l'autre et donc plus coûteux) et le comportement d'évitement est supprimé à grande distance. Pour la gestion des foules, plusieurs niveaux de contrôle [MT01] peuvent être distingués, chacun d'entre eux ayant une influence sur l'autonomie des personnages la constituant mais aussi sur le temps de calcul global de l'application (voir fig. 2.17). La foule est ensuite décrite de manière hiérarchique (foule, groupes, individus) au travers de ViCrowd, l'architecture de simulation (voir fig. 2.16). Elle prédéfinit un certain nombre de comportements liés à des algorithmes de gestion de nuée [Rey00] pour la gestion d'un groupe d'entités, le groupe est alors considéré comme une entité à part entière avec ses propres buts. Les entités ensuite ont le choix de suivre le groupe ou bien de changer de groupe.

Contrôle du comportement	Foules guidées	Foules programmées	Foules autonomes
Niveau d'autonomie	Bas	Moyen	Haut
Niveau d'intelligence	Bas	Moyen	Haut
Rapidité d'exécution	Haute	Moyenne	Basse
Complexité des comportements	Basse	Variable	Haute
Niveau d'interaction	Haut	Variable	Variable

FIG. 2.17 – *Les comparaisons des différentes caractéristiques comportementales des foules en fonction du type de contrôle qui leur est imposé. Ce tableau est extrait des travaux de Raup Musse [MT01].*

Conclusion

La simulation du comportement de navigation d'humanoïdes virtuels pose plusieurs problèmes et s'avère être un domaine pluridisciplinaire, ceci en plusieurs sens : d'une part en terme de domaines de recherche où les informations issues de la socio-psychologie sont importantes à prendre en compte pour la modélisation informatique du comportement, d'autre part en terme informatique où elle touche à plusieurs problèmes tels que la représentation de l'environnement, la planification de chemin, la détection de collision et la définition de comportements de plus ou moins haut niveau. Pour obtenir une architecture efficace alliant à la fois autonomie et rapidité, les compromis à faire sont nombreux.

La représentation de l'environnement joue un rôle capital, d'une part pour la planification de chemin, mais aussi dans certains cas pour la détection de collision. L'utilisation d'algorithmes liés aux champs de potentiels permet, dans beaucoup de cas de calculer rapidement et éventuellement de manière incrémentale un chemin d'un point à un autre de l'environnement mais est cependant soumis au problème des minima locaux [Lat91]. L'utilisation d'algorithmes aléatoires pour la sortie des minima rend difficile son utilisation dans le cadre de la simulation de comportements humains car ce facteur est en dehors de toute logique de navigation humaine. La représentation discrète

de l'environnement semble plus adaptée mais pose le problème du compromis mémoire/précision dans le cas des cartes approximatives. Dans ce type de modèles, il est d'une importance capitale de maîtriser la taille du graphe utilisé pour la planification de chemin dans le cadre d'une simulation interactive, contenant un grand nombre d'entités. Finalement, les approches à base de discrétisation exacte semblent être les plus adaptées. D'une part, le nombre de cellules générées ne dépend que de la complexité géométrique de l'environnement. D'autre part, elles fournissent une représentation précise, permettant aux humanoïdes de repérer rapidement les obstacles proches. Elles offrent donc une base riche en informations géométriques, utiles à la gestion d'une navigation fine.

Dans le domaine de la modélisation du comportement de navigation, beaucoup de paramètres sont à prendre en compte. La phase la plus importante et certainement la plus coûteuse en terme de calcul est la détection de collision. En effet, elle se base sur des relations de voisinage entre entités et nécessite l'utilisation de structures de données permettant de calculer rapidement ce voisinage. Dans certains cas, il est possible d'utiliser la structure de représentation de l'environnement pour permettre de filtrer les entités à prendre en compte [TC00, Tho99, KKWH01], ou alors des structures de données dédiées peuvent être mises en œuvre [Rey00, O'H00]. L'utilisation de ces structures dépend grandement du nombre d'entités que l'on cherche à simuler, plus leur nombre est grand plus l'on doit faire attention à la complexité d'accès au voisinage. D'autre part, la notion de représentation hiérarchique du comportement des entités et la notion de réunion des entités par groupes permet de limiter les calculs [O'H02] mais au détriment de l'autonomie [MT01]. Enfin, la définition des règles de navigation proches de la navigation humaine est utile pour le réalisme mais, pour être réellement exploitée, requiert l'utilisation d'un environnement informé pour s'adapter aux situations réelles [Tho99, FBT99].

Chapitre 3

Des architectures

Dans ce chapitre, nous allons nous attacher à présenter des architectures existantes permettant de simuler le comportement d'humanoïdes virtuels autonomes. Ces architectures s'appuient sur l'intégration du travail de plusieurs personnes, autour de plate-formes orientées vers la simulation du comportement. L'intérêt est ici de voir comment différentes approches de systèmes décisionnels sont utilisées et couplées pour traduire diverses caractéristiques du comportement humain. L'autre intérêt réside dans la gestion des interactions entre l'humanoïde et son environnement.

3.1 Jack

Jack est le nom de l'architecture de simulation pour les humanoïdes virtuels utilisée à l'université de Pensylvanie [BPB99]. Elle intègre différents niveaux de contrôle allant de l'animation des humanoïdes de synthèse jusqu'à l'interaction en langue naturelle.

Transom Jack Toolkit. Le Transom Jack Toolkit est une boite à outils, équipée d'une interface graphique évoluée permettant de décrire des humanoïdes de synthèse pouvant évoluer dans des environnements virtuels. Elle fournit un système d'animation complet allant de la cinématique inverse à la gestion de mouvements capturés. Elle offre, d'autre part, un interfaçage avec Python (un langage de script interprété) et C++, permettant ainsi de contrôler les animations. Cette propriété est utilisée dans les modules de plus haut niveau, développés à l'université de Pensylvanie pour gérer le comportement des entités virtuelles.

Emote. Le système Emote [CCZB00] se base sur le modèle de Laban, et fournit une sur-couche de haut niveau pour la description des mouvements. Après diverses observations des mouvements humains, dans différentes situations, Laban a extrait un certain nombre de caractéristiques permettant de les qualifier : forme, amplitude, vitesse... En s'appuyant sur ces travaux, le système Emote offre une description de haut-niveau des caractéristiques des mouvements, de manière beaucoup plus intuitive que le paramétrage mathématique de leur calcul. Les mouvements peuvent alors être paramétrés en fonction de la situation, de la raison pour laquelle le geste est effectué et éventuellement des émotions que peut ressentir l'humanoïde [AB02]. Ce système est connecté à Jack, et effectue le lien entre les paramètres qualitatifs du mouvement et sa réalisation effective, au travers du paramétrage mathématique des animations.

PaT-Nets et boucles SCA. Les PaT-Nets, que nous avons décrit à la section 1.1.3 permettent de décrire les comportements qui peuvent être adoptés par l'humanoïde sous la forme d'automates parallèles. Couplés avec les boucles SCA (voir section 1.1.1), ils permettent de traduire des comportements de navigation réactive, en offrant un contrôle d'ordre supérieur. Dans les autres cas, ces automates contrôlent les mouvements de manière à exhiber des comportements cohérents, en fournissant une base de comportement pour le plus haut niveau : les PAR.

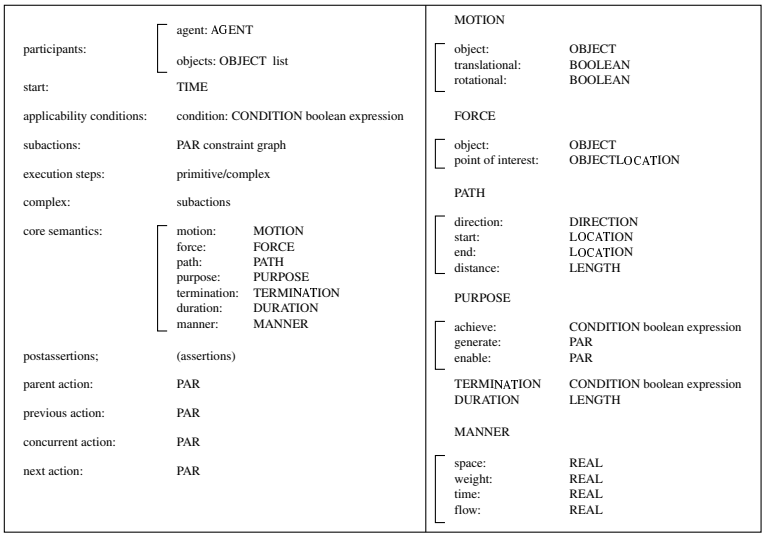


FIG. 3.1 – *Les informations associées aux PAR.*

PAR. Les PAR[1] [BSA+00] constituent le plus haut niveau d'abstraction pour la description des comportements des humanoïdes de synthèse. Ils s'agit de structures décrivant les actions ainsi que différentes caractéristiques qui leur sont associées (voir fig. 3.1), tout en offrant l'opportunité d'effectuer une forme de planification. Les éléments principaux de description sont les suivants :

- *Les objets* : il s'agit de la liste des objets physiques qui vont être concernés par l'action.
- *L'agent* : ce paramètre stipule l'agent qui va devoir exécuter l'action.
- *Les conditions d'application* : il s'agit d'une expression décrivant les conditions qui doivent être nécessairement remplies dans l'environnement pour que l'action puisse être exécutée par l'agent.
- *Les actions préparatoires* : Il s'agit d'une liste de couples (condition, action), dont toutes les conditions doivent être remplies pour exécuter l'action. Si une condition n'est pas satisfaite, l'action associée doit être exécutée, en vue de la satisfaire et en conséquence rendre l'action principale du PAR réalisable. Les actions associées aux conditions possèdent le même format de description que l'action associée au PAR que nous allons décrire.
- *L'action* : Il s'agit de l'action associée au PAR, qui peut être qualifiée de simple ou complexe. L'action simple se traduit par la demande d'exécution d'un PaT-Net traduisant le comportement adopté par l'agent. Les actions complexes décrivent une réalisation par l'intermédiaire de plusieurs sous-actions (ou PAR). Elles peuvent être exécutées en parallèle, en séquence ou

1. Parameterized Action Representation

FIG. 3.2 – *Exemple de scénario d'arrestation.*

une combinaison des deux. Une action complexe est terminée lorsque toutes les sous-actions sont terminées ou que les conditions de terminaison sont satisfaites.

- *Les conditions de terminaison* : Il s'agit d'une liste de conditions permettant de spécifier sous quelles conditions l'action est considérée comme terminée.
- *Les effets* : Il s'agit d'instructions spécifiant les mises à jour qui sont effectuées dans la base de données représentant le monde lorsque l'action est terminée.
- *La manière* : Il s'agit de la manière de réaliser l'action. Elle permet de qualifier la gestuelle qui sera adoptée par l'humanoïde et fait donc le lien avec le modèle Emote.

Les PAR offrent donc une description de haut niveau des actions qui peuvent être effectuées par un humanoïde de synthèse. Au travers de la spécification des actions préparatoires, le formalisme autorise une forme de « planification ». En effet, ces actions permettent de mettre l'humanoïde dans un état où il pourra réaliser l'action. Cette action peut alors être vue comme un but, les actions préparatoires décrivant la manière d'atteindre la réalisation de cette action. Les actions préparatoires étant elles-mêmes des PAR, ce processus peut se propager, générant ainsi une suite complexe d'actions. La description de l'action associée au PAR offre des particularités directement extraites de l'utilisation sous-jacente des PaT-Nets. Cette spécification permet d'organiser la réalisation de plusieurs sous-actions en parallèle et de poser des barrières de synchronisation avant l'exécution de l'action suivante. Les actions décrites peuvent alors s'avérer très complexes et peuvent traduire un grand nombre de comportements humains, caractérisés par la notion de parallélisme de tâches. Enfin, cette utilisation des PaT-Nets à l'intérieur de la description des PAR permet de faire le lien entre la représentation haut-niveau des actions et leur réalisation effective sous la forme de comportements adoptés par l'humanoïde de synthèse (voir fig. 3.2).

Langue naturelle. L'utilisation de la langue naturelle est une des particularités de ce système, elle nous concerne relativement peu, mais mérite d'être citée. Les PAR ont été conçus sur la base d'une relation avec la langue naturelle. Il est possible, au dessus des PAR d'utiliser un logiciel de reconnaissance vocale, qui génère des phrases. Ces phrases sont ensuite analysées par le *XTAG Synchronous Tree Adjoining grammar* [Sch99], un logiciel dédié à l'analyse et à la reconnaissance de la langue naturelle. Sa combinaison avec les PAR permet à un être humain de donner des ordres en langue naturelle aux agents, ces ordres, sont ensuite retranscrits sous la forme de l'instanciation d'un PAR. Les objets contenus dans la phrase ainsi que des adjectifs qualifiant les mouvements sont

FIG. 3.3 – *Des avatars pilotés en utilisant la langue naturelle.*

alors interprétés et servent au paramétrage du PAR. Cette transcription se fait à l'aide d'une base de données permettant de mettre en relation les objets qualifiés par l'utilisateur avec les objets du monde virtuel. Le PAR, une fois instancié, est réalisé par l'humanoïde virtuel. La figure 3.3 présente une scène d'intérieur mélangeant quatre avatars et un acteur semi-autonome effectuant le service. Les utilisateurs peuvent alors interagir, au travers du monde virtuel, en pilotant leur avatar par l'intermédiaire de la langue naturelle.

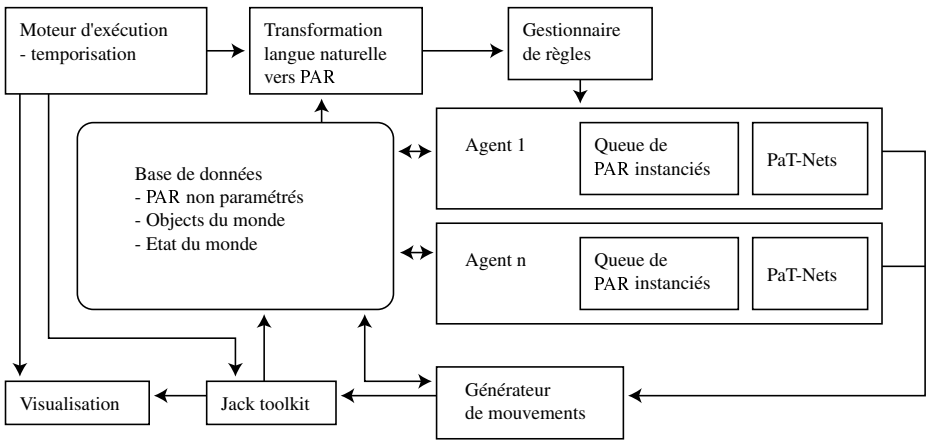


FIG. 3.4 – *Architecture définie autour de Jack.*

Architecture La figure 3.4 montre l'architecture utilisée lors des simulations. Le moteur d'exécution effectue une temporisation de la simulation. La base de donnée centrale regroupe toutes les informations sur l'état du monde, les PAR définis et non instanciés ainsi que les objets de l'environnement. Lorsqu'une instruction est dictée en langue naturelle, elle est transformées en PAR instancié (en utilisant la base de données) qui est fourni au gestionnaire de règles. Ce gestionnaire de règles possède deux rôles, il fournit les PAR traduisant les commandes des utilisateurs aux agents concernés et vérifie un certain nombre de règles régissant le monde comme par exemple : lorsqu'une voiture

arrive devant la barrière, lever la barrière. Chaque agent est un processus qui possède plusieurs rôles :

- il gère l'ajout des PAR dans la queue,
- il peut communiquer avec les autres agents par l'intermédiaire de messages
- il sélectionne des actions sur la base de la personnalité des agents, les messages reçus et l'état de l'environnement,
- il permet la consultation de l'état d'exécution d'un PAR,
- il assure qu'il n'y a pas d'ajout récursif de PAR à partir des règles.

Enfin, l'agent assure l'exécution des PaT-Nets permettant la réalisation des PAR. Les commandes de mouvements envoyées par les PaT-Nets sont passées au générateur de mouvement, permettant ainsi le contrôle de l'humanoïde et la génération des animations.

Ce système est très complet, il permet de créer des comportements en les manipulant à un haut niveau d'abstraction tout en n'omettant pas les informations cruciales pour leur exécution. Cependant, il est à noter qu'il offre peu de sécurité lors de la gestion des PaT-Nets. La notion d'exécution parallèle des actions est directement corrélées aux capacités des PaT-Nets et de leur utilisation de processus légers (*thread*). Dans cette mesure, les mécanismes de synchronisation utilisés sont des sémaphores, il n'existe donc pas de garantie de non inter-blocage entre les divers comportements. D'autre part, la gestion de la description des actions préparatoires semble pouvoir poser problème. Le processus de l'agent vérifie lors de l'utilisation des PAR que les ajouts de nouveaux PAR dus à ces règles ne provoque pas une boucle infinie, mais ne semble pas prendre en compte les éventuelles interactions entre les actions préparatoires. Si ces interactions sont relativement faciles à éviter à un seul niveau de description, il semble beaucoup plus difficile de les prévoir à plus longue échéance.

3.2 ACE : une plate-forme pour les humanoïdes virtuels

Le VRLab [2] (anciennement LIG [3]) est un laboratoire de l'EPFL [4] dont une partie des recherches portent sur l'animation d'humanoïdes virtuels. Ce laboratoire dispose d'un grand nombre d'outils pour la description des humanoïdes de synthèse et de leurs comportements.

ACE. Les divers modèles de comportement se basent sur la plate-forme ACE [5] [KMCT00] et du module *agentlib* [BBET97]. Cette plate-forme fournit un environnement de développement, permettant notamment de répartir la gestion des agents sur plusieurs machines, tout en offrant des possibilités de communication au travers d'un protocole TCP/IP. Elle dispose aussi de la possibilité d'utiliser des processus légers (*thread*) lors de la description des applications et particulièrement des différents modules régissant les simulations. Elle fournit aussi le support d'animation pour les humanoïdes virtuels. Il est basé sur des modules de cinématique inverse [BB98], de possibilité d'utilisation de capture de mouvement et d'animation faciale [Mon02]. En terme de représentation de l'environnement, elle fournit des fonctionnalités pour la navigation des agents, au travers de l'utilisation d'une triangulation de Delaunay contrainte [KBT03]. Lors du chargement de la base géométrique de l'environnement, la triangulation contrainte est calculée, fournissant ainsi une discrétisation exacte de l'environnement sous la forme de triangles, permettant aux humanoïdes de

2. Virtual Reality Lab.
3. Laboratoire d'Informatique) Graphique
4. École Polytechnique Fédérale de Lausanne
5. Agent Common Environment

naviguer et de planifier leur chemin (voir fig. 3.5). Enfin, elle définit aussi une architecture de perception pour les humanoïdes [BBT99], permettant de lister les objets présents dans leur champ de vision. Cette plate-forme fournit donc une base solide pour le développement de modèles permettant de gérer les humanoïdes virtuels.

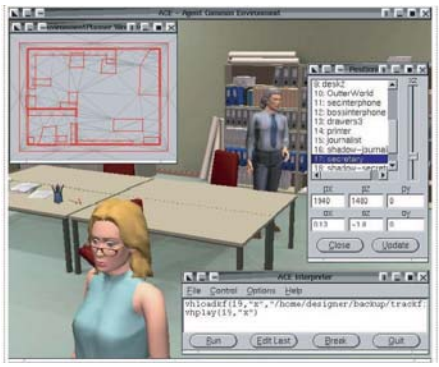


FIG. 3.5 – *Une vue de l'interface de la plate-forme ACE.*

Smart objects. Les *smart objects* [Kal01] décrivent les objets peuplant l'environnement sous la forme des interactions qu'ils peuvent offrir aux humanoïdes de synthèse. Le but est donc de déporter la méthodologie et la connaissance experte de la manipulation des objets à l'intérieur des objets eux-mêmes. Ces objets peuvent être considérés comme semi-actifs, dotés d'un comportement et surtout d'informations d'interaction pour les humanoïdes. Leur description s'effectue par l'intermédiaire de quatre grands types d'informations :

- Les propriétés intrinsèques de l'objet : il s'agit de variables décrivant l'état de l'objet. Elles peuvent décrire les parties mobiles, le poids, le centre de masse, des variables permettant de qualifier son état...
- Les informations d'interaction : il s'agit des informations nécessaires à l'agent pour lui permettre d'effectuer des interactions. A titre d'exemple, elles peuvent identifier les parties mobiles de l'objet, les informations de manipulation (point de saisie, angle d'attaque...), de positionnement ainsi que les mouvements qui peuvent être effectués par l'objet.
- Le comportement de l'objet : ces comportements décrivent les réactions de l'objet aux interactions avec l'humanoïde. Certains comportements peuvent être contraints par l'état interne de l'objet, par exemple, une imprimante ne peut imprimer quelque chose sans être préalablement allumée.
- Le comportement attendu de l'acteur : il est associé à chaque comportement de l'objet et fournit une description du comportement que l'acteur doit adopter pour être en mesure d'effectuer les interactions avec l'objet.

Dans un second temps, il est possible de modéliser au travers d'automates à états des comportements évolués, à la fois pour l'objet et aussi pour l'acteur, consistant à enchaîner plusieurs comportements atomiques de l'objet. Cela rend possible la description d'interactions complexes. L'ensemble de ces propriétés peuvent être décrites au travers d'un outil de modélisation dédié : SOMOD [KT02].

Cette approche, originale, fait le lien entre les objets pouvant peupler l'environnement et les comportements qui peuvent être adoptés par les humanoïdes (voir fig. 3.6). Elle permet de créer

des objets génériques, qui peuvent être utilisés dans différentes simulation, sans pour autant prévoir explicitement leur utilisation lors de la description de l'humanoïde. L'humanoïde peut alors consulter les informations de l'environnement et récupérer automatiquement le comportement d'interaction à partir de l'objet. Cette approche est utilisée pour décrire divers comportements d'interaction plus ou moins complexes tels que l'utilisation d'un ascenseur, la manipulation d'objets...

FIG. 3.6 – *Exemples d'interactions avec les* smart objects*[KT02].*

Piles de tâches. Les piles de tâches [Mon02] permettent de gérer le parallélisme entre plusieurs actions lors de l'animation de l'humanoïde. Une pile représente un groupe d'exclusion mutuelle, dans lequel l'ordre dans la pile correspond à un ordre de priorité sur les tâches. Ces tâches peuvent être dans deux états : actives ou suspendues. Donc, à l'intérieur de chaque pile, une seule tâche peut être exécutée à la fois, et il s'agit de la tâche la plus proche du sommet de la pile et qui est active. Les autres tâches sont bloquées, en attente de terminaison de la tâche de sommet de pile, ou en attente d'un reprise d'exécution pour les tâches suspendues. Cinq types de piles sont utilisés, autorisant un maximum de cinq types de comportements en parallèle :

- La **pile de déplacement** contient toutes les commandes de type : aller d'un point à un autre de l'environnement.
- La **pile de tâches visuelles** contient l'ensemble des tâches visuelles demandées à l'humanoïde. Cela permet de gérer l'attraction du regard par un objet durant le déplacement de l'humanoïde.
- La **pile d'animation faciale** contient l'ensemble des tâches permettant d'animer le visage de l'humanoïde et permettant donc de traduire des informations sur son état d'esprit, par exemple.
- La **pile des interactions avec les *smart objects*** contient l'ensemble des comportements d'interaction adoptés par l'humanoïde avec les *smart objects*.
- La **pile d'animation** contient les demandes d'animation traduisant des gestuelles de l'humanoïde.

D'autre part, chaque tâche contient une information sur les tâches suivantes, qui devront être activées lors de la terminaison de la tâche courante. Cette fonctionnalité permet de traduire des enchaînements de tâches, décrivant ainsi un comportement évolué mais linéaire.

Cette notion permet donc de gérer une forme de parallélisme dans les comportements. Cependant, cette approche tient compte d'une forme de cohérence en terme d'animation de l'humanoïde mais pas réellement en terme de comportement. En effet, un comportement se traduit sous la forme d'un enchaînement de tâches, qui d'une part n'est pas forcément linéaire et qui, d'autre part, dispose

d'une cohérence intrinsèque à respecter. La notion de pile de tâches ne permet donc pas d'assurer la cohérence d'un comportement de préhension, nécessitant à la fois l'utilisation des mains et des yeux pour focaliser sur l'objet d'intérêt, parallèlement à l'observation d'une voiture qui passe dans la rue. Dans ce cas, il est possible que le comportement de préhension se déroule sans l'utilisation de la vue, réquisitionnée pour l'observation de la voiture, ce qui s'avère peu cohérent par rapport au comportement humain. Elle dispose cependant de la qualité d'offrir une gestion simple et efficace de la parallélisation des actions.

Comportements de haut niveau. La description des comportements de haut niveau, associés aux humanoïdes passe par une architecture de type BDI, décrivant un certain nombre de règles dépendantes de l'état du monde et de l'état interne de l'humanoïde [CT00, MCT01]. Le monde est décrit sous la forme de faits, traduisant les croyances de l'agent. Il en existe deux formes : les croyances à long-terme et les croyances à court-terme. Les croyances à long-terme décrivent des propriétés intrinsèques au monde et traduisent une structure statique n'évoluant pas au cours du temps. Les croyances à court-terme traduisent les faits pouvant changer dans l'environnement, en fonction des interactions de l'agent. Le stockage de ces faits s'effectue dans deux bases de données, la base propre à l'agent, qui stipule l'état de ses connaissances internes, et la base propre au monde, qui stipule l'état du monde. Des requêtes perceptives peuvent être effectuées sur le monde, pour mettre à jour la base propre à l'agent. D'un autre coté, les actions de l'agent peuvent mettre à jour la base de donnée qui lui est propre et la base représentant le monde, pour mettre à jour les modifications qui ont pu être effectuées par ses comportements.

Les agents sont caractérisés par un état interne, traduisant leurs émotions et les paramètres physiologiques. Ces états s'expriment par des variables dont la valeur peut évoluer au cours du temps. Cette évolution s'effectue suivant deux facteurs : une évolution temporelle par défaut (évolution de la faim par exemple) ou comme conséquence d'une action (manger réduit la faim).

Les actions réalisables par l'humanoïde se traduisent sous la forme de plans décrits par l'intermédiaire de trois types d'informations :

- Des conditions sur l'état des paramètres internes de l'humanoïde spécifient des intervalles de valeurs sur les paramètres internes qui valident ou invalident le plan. Par exemple, il est possible de spécifier des actions qui ne peuvent être réalisées que par des agents curieux.
- Les pré-conditions de l'action traduisent une certain nombre de faits qui doivent être présents dans l'environnement pour que l'action puisse être exécutée.
- Les effets de l'action qui traduisent les conséquences de l'action. Ces conséquences sont de trois formes :
  - les faits ajoutés ou supprimés de l'état des connaissances propres à l'agent,
  - les faits ajoutés ou supprimés dans la base du monde, traduisant les effets perceptibles des actions,
  - les actions effectives de l'agent, qui se traduisent par des comportements adoptés (sous la forme d'interactions avec les *smart objects* ou sous la forme de tâches décrites précédemment).

Ces plans sont aussi paramétrés par l'intermédiaire de variables, permettant de les réutiliser pour plusieurs catégories d'humanoïdes et différentes formes d'interactions. La sélection des actions s'effectue alors de la manière suivante [Mon02] :

1. Vérification des pré-conditions du plan : les plans sont parcourus et une tentative de paramétrage est effectuée, tout en testant la validité des pré-conditions. Une liste de propositions

FIG. 3.7 – *Un exemple de scénario [Mon02].*

d'action est alors créée, contenant les plans paramétrés dont les pré-conditions sont satisfaites.

2. Vérification de l'état des variables internes : pour les plans possibles, les conditions portant sur l'état des variables internes sont testées.
3. Lorsque toutes les pré-conditions sont satisfaites, le plan est exécuté, provoquant ainsi la mise à jour des connaissances de l'agent, de l'état du monde, ainsi que l'exécution des comportements associés au plan.

Il s'agit donc de l'expression d'un système à base de règles qui sont exécutées dès que leur pré-conditions sont satisfaites. Pour traduire une suite d'actions, permettant de réaliser un but de plus haut niveau, les croyances sont utilisées pour spécifier les sous-buts suivant l'exécution d'un plan. De cette manière, une action ayant pour conséquence la demande de réalisation d'un sous-but sous la forme d'une croyance, pourra être suivie de manière cohérente par une autre action, possédant ce sous-but comme pré-condition. Il s'agit d'une architecture de type BDI, possédant la particularité de représenter les désirs sous la forme de croyances, autorisant ainsi leur persistance.

Ce système n'intègre donc pas de manière explicite la gestion de buts et la recherche d'une suite d'actions en vue de les satisfaire. Il s'agit d'un système hybride, mélangeant les propriétés des systèmes à base de règles avec la notion d'effet, propre à la planification. L'expression de buts se fait alors par la configuration des règles et l'utilisation de croyances particulières conditionnant les enchaînements (en cela similaire à un système de script, mais moins contraignant). Cela le rend moins souple que les systèmes utilisant des notions de planification mais lui permet d'être utilisé relativement aisément pour spécifier des scénarios.

Architecture. La figure 3.8 montre l'architecture globale des applications utilisant les humanoïdes virtuels. Chaque agent est séparé en deux entités distinctes : l'agent incarné et l'agent intelligent. L'agent incarné correspond à l'agent possédant un comportement en cours de réalisation et une représentation graphique. L'agent intelligent est l'agent possédant une représentation des connaissances ainsi que les actions possibles à effectuer. Ces deux parties sont reliées par l'intermédiaire d'une communication réseau, permettant d'exécuter sur des machines différentes les processus de raisonnement et la réalisation des actions. L'agent incarné fournit à l'agent intelligent les résultats de sa perception pour la mise à jour de la connaissance, ainsi que les informations relatives à la terminaison des actions. L'agent intelligent, pour sa part, fournit les actions à exécuter à l'agent incarné. Il est à noter qu'il existe plusieurs formes de parallélisme à l'intérieur de l'architecture. D'une part, les agents incarnés possèdent chacun des processus légers pour leur exécution et d'autre part, les agents intelligents peuvent fonctionner sur des machines distantes par l'intermédiaire d'une

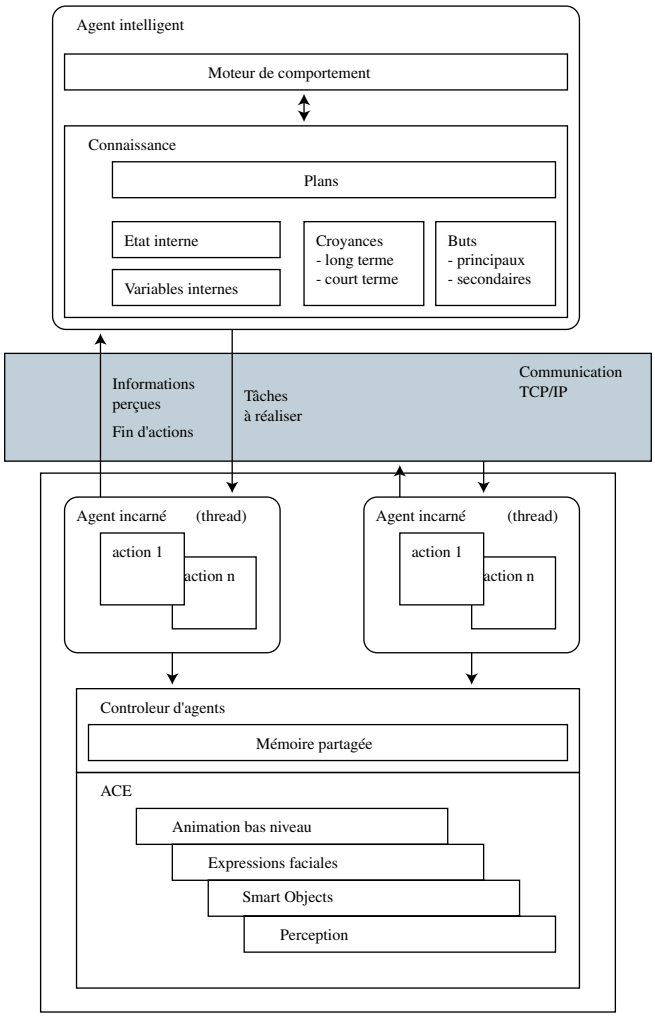


FIG. 3.8 – *Architecture d'exécution pour les humanoïdes virtuels.*

communication TCP/IP. Enfin, la plate-forme ACE gère les comportements, les animations qui leurs sont associées, la relation avec les *smart objects* et la perception, qui sont propres aux agents incarnés.

Ce système est très complet, il offre des fonctionnalités d'animation bas-niveau évoluées, tout en permettant la gestion d'un environnement complexe et une description des interactions qui peuvent être effectuées. L'introduction de la triangulation de Delaunay contrainte pour la représentation de l'environnement offre de bonnes propriétés en terme de représentation de l'espace et permet de gérer des environnements possédant une géométrie complexe. Cependant, nous n'avons pas trouvé d'informations sur le mode de gestion de la navigation et des évitements de collision (ces modèles existaient dans le couplage des travaux de Farenc [FBT99] et de Raupp Muss [MGT99], mais la représentation de l'environnement a changé depuis). Il existe cependant un problème inhérent à la triangulation de Delaunay, qui est la détection des goulets d'étranglement dans l'environnement. L'utilisation d'une triangulation de Delaunay, sans traitement supplémentaire, ne permet pas d'assurer que l'agent peut naviguer d'un triangle à un autre sans problèmes d'envergure. Les modèles de comportement sont évolués et permettent de traduire une grande variété de comportements, ainsi qu'une forme de parallélisme entre les actions, qui cependant n'assure pas une cohérence parfaite des comportements comme nous l'avons précisé dans la partie sur les piles de tâches. Le mécanisme de sélection d'actions utilise une représentation des connaissances issue des domaines STRIPS, et s'avère relativement peu coûteux en mémoire. Son mode de fonctionnement est orienté but par conception des actions, mais n'exhibe cependant pas de possibilités de planification réelles. L'expression des buts doit donc se faire étape par étape, pour forcer l'enchaînement des actions. Cependant, les résultats sont concluants et permettent de gérer des situations complexes, en n'omettant pas de faire le lien avec le modèle de comportement sous-jacent.

3.3 OpenMASK et animation comportementale

Dans cette partie, nous allons décrire les travaux de l'équipe SIAMES de l'IRISA [6], qui constituent la base (et l'historique) des travaux qui ont été effectués durant cette thèse. Nous allons donc effectuer une description détaillée de certains points, pour expliciter les bases des travaux qui seront présentés dans la deuxième partie de cette thèse.

OpenMASK Les travaux sur l'animation comportementale se basent sur la plate-forme OpenMASK [7] [MAC+02], qui est une refonte de l'ancienne plate-forme GASP [8] [CD97, DCDK98]. Cette plate-forme offre un environnement générique pour la modélisation d'applications en réalité virtuelle, en simulation dynamique et en animation comportementale pour le cas qui nous intéresse. La description d'une application se fait sur la base de modules, pouvant communiquer avec d'autres modules sous la forme de flots de données et de messages, tous deux typés. Cette notion de communication, permet de gérer une indépendance de description des modules ainsi que des possibilités de répartition automatique de l'application sur plusieurs machines distantes. Chaque module décrit se voit attribué, lors de la phase de simulation, une fréquence de calcul. Cette notion constitue la base de fonctionnement de la plate-forme ; la notion de parallélisme entre les différentes entités qui

6. Institut de Recherche en Informatique et Systèmes Aléatoires
7. Open Modular Animation and Simulation Kit
8. General Animation and Simulation Platform

peuvent être simulées se base donc sur un mode de calcul par pas de temps, synchronisé sur les fréquences de calcul. Le parallélisme est synchrone et simulé, avec un temps de latence, possible à définir lors de la répartition sur plusieurs machines. Sur la base de cette plate-forme, divers modules ont été développés comme un module de visualisation graphique permettant d'effectuer le rendu des animations. Tous les travaux qui vont être présentés par la suite s'intègrent dans cette plate-forme, permettant la mise en commun des outils et des modèles.

Humanoïde virtuel L'humanoïde virtuel, développé durant la thèse de S. Menardais [Men03], constitue désormais la base du travail sur l'animation comportementale en s'insérant comme un module de la plateforme OpenMASK. Il offre des fonctionnalités diverses allant de la cinématique inverse aux mouvements capturés. Les mouvements possibles à effectuer sont fournis sous la forme d'actions spécifiques : mouvement du bras, locomotion (avec ou sans pieds d'appui)... Ces actions peuvent être manipulées indépendamment les unes des autres, le pilote humanoïde se chargeant de leur éventuel mélange au travers d'une gestion par priorités. Récemment, des travaux ont porté sur la spécification de tâches visuelles [Cou02], au travers de l'utilisation d'une méthode d'asservissement visuel, issue de la robotique. Ces travaux permettent de gérer la perception sous la forme de contraintes visuelles exprimées dans le plan image (perceptif) en animant la chaîne articulaire partant du bassin et arrivant aux yeux (voir fig. 3.9).

FIG. 3.9 – *Exemple de comportement de perception [Cou02]*

VUEMS La modélisation d'environnements informés nécessite l'utilisation d'outils dédiés pour alléger le processus de création. Le logiciel VUEMS [Don97, Mor98, TD00b] a été conçu dans cette optique. Il permet de décrire des environnements urbains informés, facilitant leur utilisation dans le cadre de l'animation comportementale et permettant d'exploiter les travaux de Gibson [Gib86] sur la perception. Son domaine de description s'étend de la modélisation d'un réseau routier jusqu'aux informations nécessaires à la navigation des piétons (voir section 2.3.3.3). Il génère les fichiers géométriques représentant l'environnement, ainsi que la base de données fournissant les informations à la fois géométriques, topologiques et sémantiques utiles à la gestion des comportements. La figure 3.10 présente l'interface du logiciel ainsi que le comportement de traversée de route d'un piéton. Ce logiciel est dédié à la description de milieux urbains et n'intègre pas de fonctionnalités permettant de décrire et d'informer des milieux intérieurs tels qu'un appartement par exemple.

HPTS Le modèle HPTS[9] [MD98, Don01] est un système réactif multi-agent intégrant la notion de concurrence. Il s'inspire des travaux de Newell [New90] en offrant un modèle décisionnel hiérarchique,

9. Hierarchical Parallel Transition System

FIG. 3.10 – *Exemple d'utilisation du logiciel VUEMS.*

```
SMACHINE Id ;
{
    PARAMS type Id {, type Id}* ; // Les paramètres
    VARIABLES
    {
        { type Id ; }*
    }
    INITIAL Id ; // État initial
    FINAL Id ; // État final
    STATES
    {
        ( Id // Nom de l'état
        { [ réel {, réel} ] } // Temps minimum et maximum à rester dans l'état
        {RANDOM } ; // Choix de transition aléatoire optionnel
        { /* Corps de l'état */ })+
    }
    { TRANSITION Id ;
    {
        ORIGIN Id ; // État origine de la transition
        EXTREMITY Id ; // État extrémité de la transition
        { DELAY réel ; } // Temps de transition
        { WEIGHT réel ; } // Poids de la transition pour le tirage aléatoire
        condition / action
        { /* Corps de la transition */ }
    }
    }*
}
```

FIG. 3.11 – *Grammaire (au format BNF) simplifiée de Gecomp ; les parties en gras correspondent aux mots clefs du langage.*

intégrant une gestion explicite du temps permettant de prendre en compte les délais de réaction. Chaque agent du système peut être vu comme une boite noire communiquant soit par messages, soit par flots de données en entrée et en sortie. Il est utilisé pour décrire la composante décisionnelle des humanoïdes. La description des comportements se base sur la notion d'automates hiérarchiques

et présente beaucoup de similarités avec le modèle HCSM (voir section 1.1.3). Il possède la particularité d'offrir un langage de description dédié, ainsi qu'un compilateur : Gecomp, dont la grammaire est présentée fig. 3.11. Ce langage permet la description de hiérarchies d'automates, statiques, en exploitant lors de la spécification des différentes actions associées aux états et transitions, un sous ensemble de la grammaire de C++. Les facteurs limitants sont nombreux. D'une part, le sous-ensemble accepté de la grammaire de C++ contraint les possibilités d'expression (pas de boucle par exemple) et n'offre qu'un typage de données faible (entiers, flottants, chaînes de caractères). Cela rend son utilisation difficile dans le cadre d'applications complexes, bien qu'il soit utilisé en conjonction avec VUEMS pour gérer le comportement de piétons [Tho99] et de conducteurs de véhicules [Mor98]. D'autre part, la description statique de la hiérarchie d'automates offre peu d'ouverture vers l'extérieur. Elle pose des problèmes pour la connexion dynamique de comportements issus de modèles décisionnels de plus haut niveau ou fournis par l'environnement en utilisant une approche de type *smart objects*. Pour être à même de dérouler plusieurs comportements manipulant le même humanoïde, la description doit contenir une synchronisation explicite pour assurer une certaine cohérence sur les actions manipulant les membres. Même si le pilote humanoïde sous-jacent dispose de notions de priorités sur les actions, elles ne sont pas suffisantes pour assurer cette cohérence. La description de comportements complexes devient alors difficile, et requiert une exhaustivité allant à l'encontre de la modularité ou requiert une exclusivité, ne permettant pas d'exploiter les notions de parallélisme.

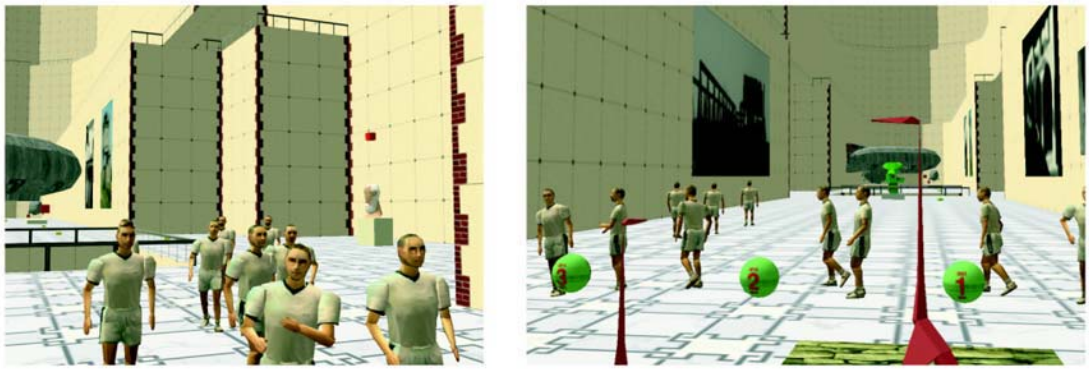


FIG. 3.12 – *Visite scénarisée du musée virtuel.*

SLURGH La scénarisation des simulations s'effectue par l'intermédiaire du langage SLURGH [Dev01, DD03]. Il permet de manipuler plusieurs acteurs semi-autonomes, en influant, à haut-niveau, sur leur comportement. L'une de ses particularités est d'offrir un mécanisme de gestion de contraintes temporelles, au travers d'un sous ensemble calculable de l'algèbre temporelle d'Allen [All81]. L'expression de ces contraintes permet de décrire l'agencement temporel des scénarios, et donc de synchroniser l'exécution. Le langage offre un certain nombre de structures algorithmiques de haut niveau, allant des diverses boucles, aux choix non-déterministes. Un système de réservation d'acteurs est fourni, permettant aux scénarios de se répartir les différentes entités en fonction d'un système de priorité sur les réquisitions. Un scénario prioritaire peut donc voler des acteurs à des

scénarios de moindre importance. La phase de compilation du langage s'effectue par une traduction des scénarios sous la forme d'une hiérarchie d'automates HPTS. La structure des automates permet alors de représenter les diverses structures algorithmiques et les synchronisations temporelles. Ce langage est à la base d'une démonstration de visite d'un musée virtuel peuplé de diverses entités, évoluant de manière semi-autonome.

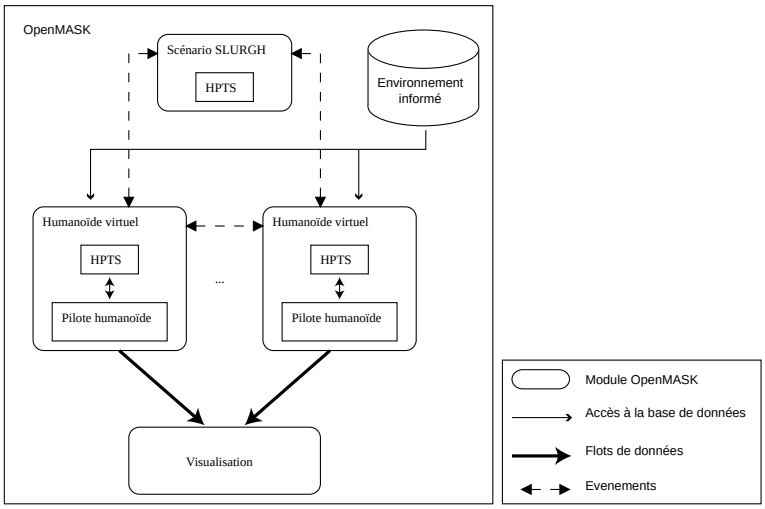


FIG. 3.13 – *Architecture d'une simulation scénarisée sous OpenMASK.*

Bilan. L'architecture d'animation [DDLT02, CDD+02] s'articule autour de la plate-forme OpenMASK et de sa modularité (voir fig. 3.13). Elle constitue l'environnement d'exécution des divers modèles proposés, leur offrant des moyens de répartition sur plusieurs machines au travers du mode de communication par flots de données ou événements. En terme de modélisation d'environnements, les outils s'adressent à la modélisation d'environnements urbains, mais pas d'environnements intérieurs. Le modèle décisionnel des entités, permet de gérer une bonne réactivité par rapport aux changements de l'environnement. Cependant, le pouvoir d'expression du langage est limité et n'offre pas de mécanisme de synchronisation implicite (comme les sémaphores pour les PaT-Nets ou les piles de tâches). La gestion de comportements complexes requiert alors une description exhaustive, qui va à l'encontre de l'utilisation de mécanismes de raisonnement de haut niveau. La gestion de scénarios, introduite au travers de SLURGH, offre des atouts en terme de simulation, en contraignant l'autonomie des agents, tout en gérant une cohérence temporelle au travers des contraintes d'agencement entre scénarios. Cependant, contrairement à l'approche de type BDI présentée au VRLab, la scénarisation est plus contraignante car elle ne prend pas en compte la notion de but au niveau de l'agent. Ce dernier, est certes, semi-autonome, mais ne peut être contrôlé à un haut niveau d'abstraction en spécifiant un but à atteindre, le laissant ainsi libre de choisir les modalités de réalisation.

Conclusion

Les architectures présentées sont certes différentes mais abordent des problèmes communs au travers de concepts qui le sont tout autant. La représentation des environnements dans lesquels évoluent les humanoïdes passe par une phase d'information. Les *smart objects* informent les objets par leurs modalités d'interaction, les environnements urbains informés par la topologie et la sémantique de l'environnement. Cette approche facilite le travail des modèles décisionnels en fournissant une information sémantique utile, rejoignant en cela la vision écologique de Gibson. Le contrôle de l'humanoïde se scinde en deux niveaux : l'un représentant le comportement de l'entité, l'autre contrôlant à un plus haut niveau d'abstraction les enchaînements des comportements. La modélisation du comportement inclue les notions de parallélisme inhérentes au comportement humain. Cependant, ces approches n'assurent que difficilement la cohérence globale du comportement et manquent de moyens de gestion implicite de leur mélange. Les comportements sont alors manipulés sous la forme d'exclusion mutuelle (plus fine dans le cas des PaT-Nets), ne prenant pas en compte, implicitement, une adaptation automatique de leur déroulement en fonction du contexte. Les modèles de contrôle de plus haut niveau ne font pas de planification mais utilisent des systèmes, permettant de décrire des scripts de manière plus ou moins abstraite laissant un niveau d'autonomie variable aux entités. Contrairement à des systèmes de planification de type *situation calculus* ou STRIPS, l'inclusion d'une nouvelle possibilité d'action nécessite une corrélation explicite avec les actions précédemment décrites. Le travail de description devient alors plus ciblé et contraint « l'inventivité » des humanoïdes face à des situations imprévues.

Conclusion

Cet état de l'art s'est focalisé sur les modèles décisionnels des humanoïdes ainsi que leur relation à l'environnement, au travers du processus de navigation. A partir des architectures présentées, il est possible de mettre en évidence trois grands types de modèles qui doivent coexister pour permettre la modélisation du comportement des humanoïdes : des modèles de contrôle de haut-niveau, travaillant sur une abstraction des comportements, des modèles réactifs, permettant de réaliser les comportements issus du contrôle de haut-niveau et enfin, des modèles de représentation de l'environnement permettant la navigation des entités.

Contrôle de haut-niveau. Les modèles permettant ce type de contrôle sont assez variés. Ils s'étendent des systèmes à base de scripts aux systèmes de raisonnement évolués. Ces modèles diffèrent par le niveau d'autonomie laissé à l'entité. Plus l'autonomie est grande, plus le coût de calcul augmente. Cette constatation a tendance à faire pencher la balance vers des architectures de type BDI, offrant un compromis entre la réactivité et une forme d'orientation vers un but. Il s'agit d'une forme de système pouvant se rapprocher du script car il lie différents composants (ou plans) par l'intermédiaire des désirs. Mais, il offre une grande réactivité en prenant en compte à chaque sélection d'un plan l'état de l'environnement. Cependant, ces systèmes, de par leur utilisation d'une connaissance experte s'avèrent assez contraints et ne peuvent, sous peine d'un effort de description considérable du concepteur, prendre en compte toutes les situations exprimables. Dans ce cadre, les systèmes de planification de type *situation calculus* ou STRIPS sont plus inventifs de par le raisonnement qu'ils peuvent effectuer sur les actions et leurs conséquences. Les actions ne sont en effet pas connectées par l'intermédiaire de marqueurs particuliers (désirs, expression de sous-buts, notion de tâche), mais au travers de leur influence sur l'environnement. Cela implique une grande modularité de description et l'exploitation automatique d'une action si ses conséquences permettent de converger vers le but. Leur défaut réside dans leur coût de calcul, qui ne leur permet que difficilement d'atteindre une certaine réactivité. Cependant, les mécanismes de sélection d'actions peuvent peut-être répondre à cette attente car, malgré certaines instabilités numériques, ils sont à la fois réactifs et orientés buts. Enfin, ils peuvent être utilisés à partir d'une formalisation du monde (de type STRIPS par exemple) qui permet de générer automatiquement les graphes d'actions. Cette propriété a notamment été utilisée dans le mécanisme de Maes [Mae90].

Les modèles de planification présentés et les langages qui y sont associés utilisent une modélisation déclarative, n'intégrant pas les objets comme des entités à part entière mais les utilisant comme médium d'accès aux propriétés. Les actions sont alors décrites indépendamment des objets, comme des opérateurs de modification du monde, prenant ces objets en paramètre. Il semble intéressant, dans ce cadre, de fournir des langages de plus haut niveau, mettant l'accent sur la modélisation des connaissances. L'introduction de concepts, tels que ceux de la programmation objet, permettrait de travailler à différents niveaux d'abstraction, tout en offrant une modularité de description accrue.

La notion de hiérarchie de concepts deviendrait alors une partie intégrante du langage, offrant une représentation des connaissances beaucoup plus proche de celle utilisée par les êtres humains.

Modèles réactifs. Les modèles réactifs sont utilisés pour décrire des comportements plus ou moins évolués, en connectant directement la perception à l'action. Leur réactivité provient de cette propriété et explique leur utilisation comme modèle exécutif des actions qui ont pu être sélectionnées par des mécanismes plus lourds, travaillant sur une modélisation des connaissances. Cependant, leur lien avec les modèles décisionnels de plus haut niveau est un point critique. D'un coté, les comportements décrits sont manipulés de manière abstraite, de l'autre, ces comportements sont concrets et ne doivent pas remettre en cause la cohérence des autres comportements éventuellement en cours de réalisation. Les divers modèles admettant l'exécution en parallèle de plusieurs comportements souffrent de lacunes pour offrir une abstraction suffisante. Certaines approches permettent de ne gérer qu'une synchronisation explicite qui va à l'encontre de l'utilisation d'un modèle de contrôle plus abstrait (les automates parallèles hiérarchiques par exemple). L'approche à base de piles de tâches autorise un certain parallélisme, mais au détriment de la cohérence du comportement global, en effet le mécanisme ne prend pas en compte la cohérence intrinsèque d'un comportement ayant besoin d'utiliser plusieurs piles en simultané. L'approche des PaT-Nets et de la synchronisation par sémaphores paraît être la plus fine et cohérente. L'exclusion mutuelle des comportements peut alors être décrite en fonction des états des automates. Cependant, l'utilisation de ce mécanisme requiert certaines précautions pour éviter les inter-blocages, qui sont un problème récurrent de la programmation parallèle. D'autre part, elle ne gère pas l'adaptation des comportements dans leur déroulement sans communication explicite. Un comportement ayant pris un sémaphore, ne le relâchera pas tant qu'il n'aura pas fini son traitement, à moins d'une demande explicite de la part d'un autre comportement. Cependant, cette notion implique un travail de description de la part du concepteur, qui peut s'avérer fastidieux. De plus, elle ne prend pas en compte de manière explicite l'importance relative des comportements pour la gestion de la concurrence sur les sémaphores.

Ces constatations mettent en avant le besoin d'un modèle réactif plus évolué qui devrait permettre la description indépendante des comportements tout en assurant leur cohérence lors de exécution. Cela implique un modèle ne souffrant pas de problèmes d'inter-blocage (PaT-Nets) et ne nécessitant pas de synchronisation explicite (automates parallèles hiérarchiques). Enfin, il devrait prendre en compte une caractéristique des comportements qui n'est pas abordée : l'adaptation cohérente et automatique de leur déroulement en fonction de la disponibilité des ressources corporelles (mains, yeux, bouche...).

Représentation de l'environnement et navigation. En terme de représentation de l'environnement, les méthodes de discrétisation exacte retiennent particulièrement notre attention. Elles permettent de générer un nombre de cellules linéaire en fonction du nombre de contraintes de l'environnement, tout en offrant une représentation précise des obstacles. Cependant, ces méthodes ne prennent pas en compte la notion de goulets d'étranglements, n'assurant pas sans traitement supplémentaire la génération de chemin exploitables par l'humanoïde. Les environnements informés exploitent ces techniques de représentation, mais nécessitent une modélisation particulière ou l'utilisation d'outils dédiés. Cela restreint grandement leur utilisation en n'autorisant pas ou difficilement, l'utilisation de modèles provenant de sources diverses, ne respectant pas la nomenclature de description ou n'étant pas produits à l'aide d'outils dédiés. Cependant, leur mode de représentation hiérarchique présente l'intérêt de pouvoir considérer l'environnement à plusieurs niveaux

d'abstraction. De fait, lors de la navigation, le piéton peut exploiter cette structure pour effectuer la planification d'un chemin abstrait, réduisant alors le coût de calcul.

Pour la navigation, les méthodes de détection de voisinage, utiles aux prédictions de collisions, se scindent en deux groupes : les méthodes exploitant la représentation de l'environnement, et les méthodes de classification dynamique (K-d arbre par exemple). L'indépendance de ces dernières par rapport au mode de représentation de l'environnement leurs confèrent une bonne propriété : la complexité moyenne du calcul et de l'accès au voisinage ne dépend que du nombre d'entités. Cet accès au voisinage est très important dans la mesure où il est à la base des modèles de navigation et plus particulièrement de la détection de collisions. Dans les versions les plus évoluées, les modèles de navigation mettent en concurrence plusieurs règles, s'inspirant de l'analyse du comportement piétonnier, pour obtenir un comportement de navigation réaliste.

La coexistence de ces différents modèles à l'intérieur d'une architecture de simulation du comportement est nécessaire. Chacun d'entre eux traduit différents modes de comportements et de représentation propres à l'être humain : capacités de raisonnement, d'action, de réaction, de locomotion et représentation de l'espace.

Deuxième partie

Modélisation du comportement

Introduction

L'état de l'art qui a été présenté dans la partie précédente, a mis en évidence le besoin de cohabitation de plusieurs modèles, pour permettre la description du comportement d'un humanoïde virtuel. L'architecture de simulation que nous allons présenter, s'inspire des conclusions de la partie précédente. Les divers outils proposés dans le cadre de cette architecture ont pour but de faciliter la description du comportement des humanoïdes virtuels, en mettant l'accent sur la modélisation, la réutilisation et l'automatisation. Elle se scinde en quatre grands points : un modèle cognitif, un modèle réactif, un modèle de navigation et une représentation particulière de l'environnement géométrique.

Modèle cognitif. Le modèle cognitif associé aux humanoïdes offre une représentation abstraite de l'environnement dans lequel ils évoluent. Cette représentation qualifie les objets et leurs propriétés et adopte une approche écologique [Gib97] consistant à décrire l'environnement sous la forme des interactions qu'il peut offrir. Le système propose un langage orienté objet : BCOOL [10], offrant des moyens de description abstraits permettant de mettre l'accent sur la modélisation des connaissances. Les informations décrites peuvent ensuite être exploitées par un mécanisme de sélection d'action travaillant sur l'abstraction des actions possibles. Ces actions sont ensuite concrétisées au travers de leur réalisation par le modèle réactif, en charge de l'animation et du déplacement de l'humanoïde. Cette notion de concrétisation des actions requiert, cependant, de posséder un modèle réactif offrant une abstraction suffisante permettant de les manipuler sans aucune connaissance sur leur déroulement.

Modèle réactif. Le modèle réactif est en charge de la réalisation des actions générées par le modèle cognitif et doit aussi être à même de prendre en compte des comportements réflexes, reliant directement la perception à l'action. Le mélange de ces comportements, provenant de sources différentes, doit pouvoir s'effectuer automatiquement en exploitant, dans la mesure du possible les notions de parallélisme, mais dans un soucis de cohérence. Nous avons abordé cette problématique au travers d'un mécanisme d'ordonnancement automatique de comportements, se basant sur l'hypothèse d'une description sous la forme d'automates parallèles (éventuellement hiérarchiques). Ce mécanisme permet une description indépendante des comportements et de leurs différentes possibilités d'adaptation, en fournissant les effets sans en connaître les causes. Le niveau d'abstraction ainsi offert, autorise la manipulation des comportements sans aucune connaissance sur leur déroulement, offrant ainsi au modèle cognitif le niveau d'abstraction nécessaire. Pour exploiter au mieux cette abstraction, le langage permettant de décrire les comportements sous la forme d'automates,

10. Behavioral and Cognitive Object Oriented Language

inclut des notions de modélisation objet, permettant de décrire des catégories de comportements manipulables à différents niveaux d'abstraction.

Navigation réactive. Le processus de navigation est un comportement nécessitant, d'une part, une modélisation de l'environnement et, d'autre part, la gestion des évitements de collisions entre les entités. Son utilisation continuelle pour la gestion des interactions entre l'humanoïde et son environnement, en fait un comportement qui requiert une attention toute particulière. Dans ce cadre, au même titre que le mélange des boucles SCA et des PaT-Nets, nous proposons un modèle de navigation réactive bas niveau, pouvant être supervisé par le modèle réactif. Dans ce cadre, nous nous sommes inspiré de l'analyse du comportement piétonnier pour définir un système configurable, permettant de prendre en compte divers aspects des règles régissant la navigation. Pour étendre son application à la gestion de grandes foules, un travail a été effectué sur la détection de collisions au travers de l'utilisation d'un graphe de voisinage permettant de limiter la complexité des calculs, tout en offrant une structure riche en informations. Cependant, ce processus requiert une représentation de l'environnement permettant à l'entité de se situer par rapport aux murs, mais aussi de raisonner pour pouvoir atteindre un but qu'elle s'est fixée.

Représentation de l'environnement et planification de chemin. Les modèles évoqués durant la partie d'analyse des modèles existants appartiennent à deux grandes classes : les environnements informés et les environnement géométriques. L'utilisation d'environnements informés possède beaucoup de qualités mais nécessite cependant lors de la conception, l'utilisation d'outils dédiés, ou une phase d'information manuelle, longue et fastidieuse. Les modèles d'environnements tels que des appartements, des immeubles ou des villes sont, dans un très grand nombre de cas, fournis sous la forme d'un modèle géométrique ne possédant pas d'information dédiée au comportement. Nous sommes parti de cette constatation pour proposer un système de subdivision spatiale, s'appuyant sur l'hypothèse de navigation sur sol plat, partant de l'analyse d'une géométrie et générant une structure, riche en informations topologiques. Ces informations topologiques peuvent ensuite être utilisées pour générer des niveaux d'abstraction, possédant d'une part une sémantique et d'autre part permettant, par la suite, de considérablement réduire le coût de calcul associé à la planification d'un chemin. Cette réduction, permet de gérer de très grands environnements pour un coût de calcul raisonnable.

Nous allons donc exposer ces différents modèles, ainsi que les langages qui leurs sont associés. Nous allons utiliser une approche ascendante de description. Dans un premier temps, nous allons nous attacher à la représentation de l'environnement et à la planification de chemin. Dans un second temps, nous aborderons la navigation réactive, au travers de son utilisation de la représentation de l'environnement. Dans un troisième temps, nous présenterons le modèle réactif, qui offre un niveau d'abstraction suffisant pour autoriser l'utilisation du modèle cognitif qui sera ensuite présenté. Dans un dernier temps, nous présenterons l'architecture globale et des réalisations effectuées sur la base des modèles proposés.

Chapitre 1

Subdivision spatiale et planification de chemin

Une entité autonome est définie par sa capacité à décider de ses actions en fonction de l'état perçu de l'environnement dans lequel elle évolue. Ses capacités d'action sont diverses, mais il en est une qui s'avère être primordiale : sa capacité à se déplacer. Pour être en mesure de se déplacer, elle doit posséder ou être à même de percevoir des informations qualifiant les restrictions imposées par la structure de l'environnement dans lequel elle évolue. Ces restrictions sont qualifiées d'obstacles car elles contraignent le déplacement. Ces obstacles peuvent être classés dans deux grandes catégories : les obstacles dynamiques et les obstacles statiques. Les obstacles dynamiques ont une position pouvant évoluer au cours du temps, de manière plus ou moins prévisible. Les obstacles statiques ont, pour leur part, une grande inertie temporelle et leur position ne change pas ou possède une très faible probabilité de changement. Il s'agit en règle générale, d'obstacles naturels, de bâtiments, ou bien de mobilier dont le déplacement est peu probable. De par cette faible probabilité de changement, les informations les concernant sont le plus souvent mémorisées sous une forme permettant par la suite de planifier un chemin en prenant en compte leur évitement.

En animation comportementale, la partie statique de l'environnement est représentée par un fichier géométrique dont la structure est connue avant le lancement de la simulation. Deux cas sont alors à distinguer : le fichier géométrique a-t-il été produit avec un logiciel de modélisation 3d classique ou bien avec un outil dédié générant, en parallèle de la modélisation, des informations décrivant la structure de l'environnement [Tho99]. Le fait est, que le plus souvent, les environnements sont fournis sous la forme de fichiers géométriques. Leur manque d'organisation ne permet que très difficilement d'obtenir une résolution efficace, dans le cas des déplacements, des interactions éventuelles entre l'entité et son environnement statique. Les problèmes liés à ces interactions sont de deux types : comment accéder rapidement à une géométrie des obstacles statiques voisins d'une entité et comment représenter les informations pour permettre une planification de chemin rapide prenant en compte son envergure.

Dans ce chapitre, nous proposons une solution, basée sur l'analyse d'un fichier de géométrie, permettant l'extraction d'une structure de données alliant informations géométriques et topologiques. Ces informations sont ensuite utilisées pour générer des niveaux d'abstraction topologiques permettant d'accélérer les calculs de planification de chemin, rendant ainsi possible l'utilisation d'environnements complexes et étendus, pour un coût de calcul modique.

1.1 Subdivision spatiale

Nous disposons, pour représenter le monde statique dans lequel l'entité évolue, d'un fichier de géométrie. Ce fichier contient une collection de points dans un espace en trois dimensions qui sont utilisés pour caractériser des facettes triangulaires représentant la géométrie de l'environnement. Les entités de type humanoïde, n'étant pas dotées de la capacité de voler, effectuent des déplacements en deux dimensions, utilisant des informations de hauteur lors de la gestion des pieds d'appui. Cette constatation nous a conduit à poser une hypothèse forte sur le mode de subdivision spatiale, qui consiste à considérer que les entités vont se déplacer sur un sol plan. Dans un premier temps, nous allons analyser la base géométrique pour en extraire un plan en deux dimensions. Dans un deuxième temps, nous allons calculer une subdivision spatiale exacte, sous la forme de cellules convexes, offrant la bonne propriété de détecter et conserver localement les informations sur les goulets d'étranglement.

1.1.1 Traitement de la base géométrique

La géométrie de l'environnement induit des contraintes sur la navigation des agents dont l'expression est : "les agents ne doivent pas percuter les obstacles de l'environnement lors de leur navigation". En s'appuyant sur l'hypothèse de départ qui est la navigation sur un sol plat, il est alors possible d'extraire d'une part la partie de géométrie contraignant le déplacement au sol et d'autre part, de projeter cette géométrie de manière à en extraire un plan en deux dimensions. Ensuite, ce plan va être simplifié pour supprimer les informations superflues.

1.1.1.1 Extraction des contraintes

Considérons pour la description suivante, que le repère de l'environnement 3d soit orienté de façon à ce que son axe Z représente la hauteur. Un humanoïde ou toute entité se déplaçant en utilisant des pieds d'appui, va exploiter une zone de navigation, que l'on peut considérer comme une tranche de l'environnement ; cette tranche possédant une hauteur équivalente à celle de l'entité en mouvement.

Le premier traitement consiste donc à découper la base $3d$ suivant deux plans, un plan correspondant au sol, le deuxième correspondant à la hauteur de l'humanoïde. Par ce traitement, l'ensemble de la géométrie appartenant à l'espace de navigation nécessaire à l'agent, donc représentant les obstacles entrant en interaction avec l'entité lors de la navigation, est extrait. Cette géométrie se présente sous la forme d'un ensemble de points et de triangles construits sur ces points. Chaque segment délimitant un triangle est alors projeté sur le plan XY pour obtenir une liste de contraintes en deux dimensions. Ces contraintes esquissent le plan de l'environnement.

Cependant, les segments obtenus par la projection représentant les bords des triangles projetés, ont une très forte probabilité d'entrer en intersection. Pour fournir une liste de contraintes pouvant être facilement exploitée, les intersections entre tous les segments projetés sont calculées, en utilisant un algorithme de type balayage [BY95], tout en supprimant les segments redondants.

1.1.1.2 Filtrage des contraintes

La liste de segments obtenus par la phase de traitement précédente, décrit la projection de la géométrie des obstacles appartenant à la zone de navigation. Pour être à même de naviguer, seules

les contraintes décrivant le contour des obstacles sont utiles ; celles représentant la projection de leur géométrie interne ajoutent de l'information non-pertinente dans le plan généré. Pour ne garder que l'information utile, et ne pas surcharger le plan de l'environnement, une phase de filtrage est appliquée pour ne conserver que les contours des obstacles. Pour ce faire, un graphe représentant la topologie des lieux est extrait puis est utilisé pour filtrer les contraintes en fonction de la ou des composantes connexes représentant la zone de navigation. La figure 1.1 présente le modèle $3d$ du quartier de la cathédrale de Rennes avec la projection de sa géométrie. La zone blanche entre les bâtiments du modèle $3d$ caractérise la zone de navigation exploitable par les entités.

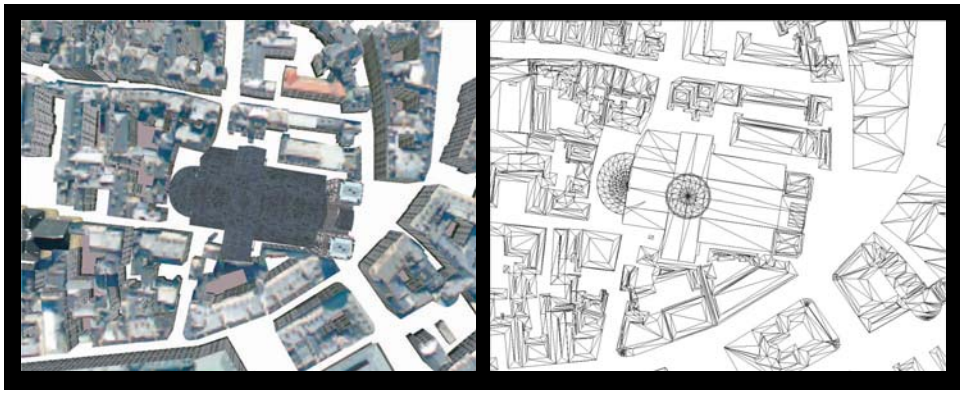


FIG. 1.1 – *Une vue de dessus du modèle 3d du quartier de la cathédrale de Rennes, avec la projection de sa géométrie.*

Dans un premier temps, une triangulation de Delaunay contrainte [Che87] est calculée sur l'ensemble des segments. Il est à noter que ce calcul est rendu possible par la phase de calcul d'intersection des segments effectuée précédemment. La triangulation ainsi obtenue est constituée de deux types de segments :

- les segments contraints : il s'agit des segments représentant la projection de la géométrie appartenant à la zone navigable.
- les segments libres : il s'agit des segments ajoutés par la triangulation de Delaunay mais ne représentant aucune contrainte.

Ce calcul produit une première subdivision spatiale, sous la forme de triangles, qui représente exactement la géométrie de l'environnement. Les segments partagés par deux triangles traduisent une information de connexité et lorsqu'il s'agit d'un segment libre, une information d'accessibilité.

Pour matérialiser l'information topologique obtenue, un graphe pour lequel un sommet représente un triangle et un arc, un segment libre est construit. La zone de navigation exploitée par l'entité se caractérise par l'accessibilité entre les différents points appartenant à cette même zone. De fait, cette zone de navigation est décrite par l'une (ou plusieurs dans le cas de possibilité de téléportation) des composantes connexes de ce graphe. L'utilisateur est donc en mesure de sélectionner la ou les composantes connexes constituant la ou les zones de navigations utiles pour ses entités[1]. Une fois cette sélection effectuée, seules les contraintes délimitant cette zone sont utiles.

1. Il est à noter que dans un grand nombre de cas, la zone de navigation est représentée par la composante connexe contenant le plus grand nombre de nœuds, notamment dans le cas de la subdivision spatiale d'une ville. Cette propriété peut être utilisée comme une heuristique mais n'est cependant pas toujours vérifiée.

Les autres contraintes représentent la projection de la géométrie interne aux obstacles et peuvent donc être supprimées. Ce traitement permet d'obtenir l'ensemble minimal de segments extraits de la géométrie délimitant la zone de navigation de l'agent. La figure 1.2 présente le plan extrait de l'environnement présenté sur la figure 1.1. Ce plan ne contient plus que les contraintes délimitant la zone de navigation des agents.

FIG. 1.2 – *Le plan filtré extrait du modèle 3d du quartier de la cathédrale de Rennes.*

1.1.1.3 Simplification des contraintes

Les segments extraits à l'étape de calcul précédente, sont intimement liés à la modélisation géométrique de l'environnement. De par la nature de cette modélisation et par le calcul de projection, il existe une très forte probabilité pour qu'un certain nombre de ces contraintes soient colinéaires ou bien que leur précision de représentation soit bien supérieure à la précision nécessaire lors de la navigation. Donc, pour tenter de limiter au maximum le nombre de contraintes à prendre en compte, une phase d'analyse visant à rassembler les segments interconnectés et colinéaires modulo une certaine précision est effectuée. De cette manière, deux contraintes considérées comme possibles à fusionner n'en produiront plus qu'une seule.

Soit A,B,C trois points dans l'environnement. Soit AB et BC deux contraintes. Soit B' la projection de B sur le segment AC, définissant la hauteur h du triangle en B (voir fig. 1.3). Le principe de fusion de contraintes s'appuie sur deux critères, d'une part un critère angulaire et d'autre part un critère de hauteur du triangle. Notons ε la hauteur maximale d'un triangle pour autoriser la fusion des contraintes et α l'angle maximal entre les deux contraintes autorisant la fusion. Les contraintes peuvent alors être fusionnées si et seulement si

$$\left(\frac{AB.BC}{||AB|| \times ||BC||} \geq \cos\alpha\right) \wedge (h \leq \varepsilon)$$

Lorsque les segments AB et BC satisfont cette contrainte, ils sont supprimés et remplacés par le segment AC, diminuant ainsi le nombre de segments et simplifiant la carte.

Ce traitement est répété jusqu'à convergence de l'algorithme, autrement dit jusqu'à ce que l'on ne trouve plus de paires de contraintes pouvant être fusionnées. L'ensemble des points superflus,

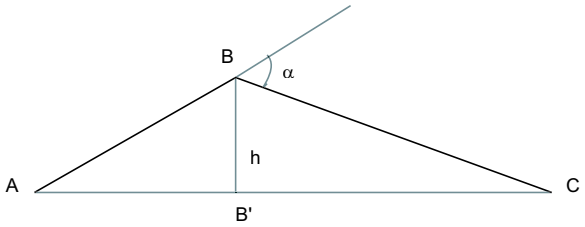


FIG. 1.3 – *Le triangle utilisé pour tester la simplification des contraintes.*

autrement dit ne servant pas réellement à la représentation des contraintes, sont supprimés, permettant ainsi d'obtenir un ensemble minimal de points et de segments représentant le plan de l'environnement. La figure 1.4 montre un exemple de simplification de la géométrie du plan d'un appartement.

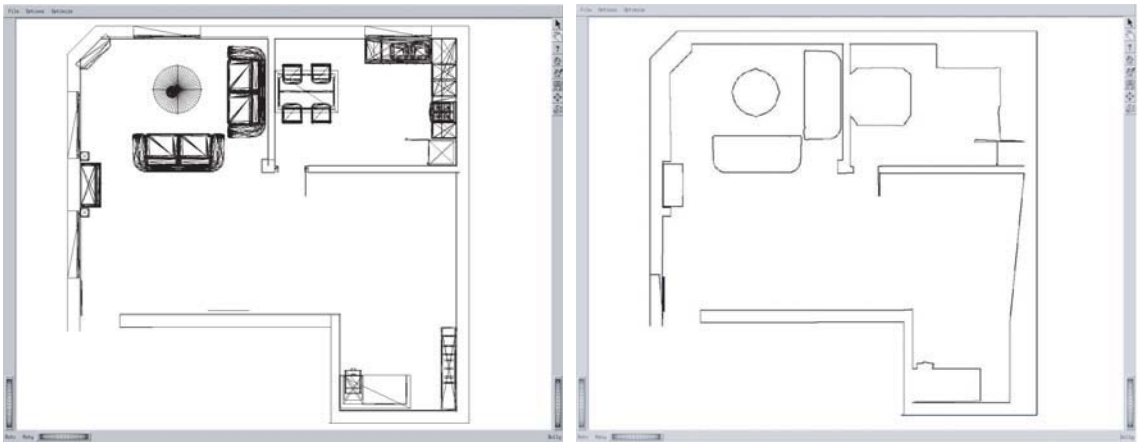


FIG. 1.4 – *Exemple de simplification de la géométrie sur le plan d'un appartement.*

1.1.2 Subdivision en cellules convexes

Les traitements précédents ont eu pour rôle d'extraire un plan simplifié de l'environnement, ne contenant que l'information nécessaire à la caractérisation de la zone de navigation exploitée par les agents. Le rôle de la subdivision en cellules convexes est alors de générer une discrétisation exacte de ce plan. Avant d'effectuer le calcul de cette subdivision, nous allons extraire de cette carte de l'environnement l'ensemble des goulets d'étranglement qu'elle contient. Grâce à cette information, nous serons à même de savoir si une entité, en fonction de son envergure, peut ou non emprunter un passage donné. La subdivision en cellules convexes qui s'en suivra sera donc calculée de manière à conserver localement ces informations.

1.1.2.1 Repérer les goulets d'étranglement

Les entités, pour être en mesure de se déplacer et planifier leur chemin dans un environnement doivent pouvoir savoir si elles peuvent passer ou non dans un passage, aussi étroit soit-il. Dans un

environnement réel, un goulet d'étranglement est caractérisé par la plus petite distance entre un angle de mur et un mur ou bien la plus petite distance entre deux angles de murs. Dans la carte générée, cette propriété se traduit par la plus petite distance d'un point à un segment ou la plus petite distance entre deux points. Nous allons donc présenter un algorithme permettant d'extraire cette information, plus qu'utile pour la navigation.

L'extraction des goulets d'étranglement se base sur une légère modification de l'algorithme de triangulation de Delaunay contrainte. Durant cette triangulation, trois types de segments vont être utilisés :

- C_s : l'ensemble des segments contraints. Ces segments représentent l'ensemble des contraintes extraites de la géométrie de l'environnement.
- F_s: l'ensemble des segments libres. Il s'agit des segments ajoutés par la triangulation de Delaunay.
- D_s : l'ensemble des segments de plus courte distance entre les coins et les murs. Ces segments sont produits par l'algorithme.

Et deux types de points :

- C_p : l'ensemble des points utilisés pour la description des contraintes de l'environnement.
- D_p : l'ensemble de points générés pour la construction des goulets d'étranglement.

Ces ensembles sont ensuite utilisés dans la modification de la triangulation de Delaunay contrainte pour caractériser les situations rencontrées. Seuls les segments de l'ensemble C_s seront considérés comme des contraintes au regard de la triangulation. L'algorithme se déroule en plusieurs phases :

1. Une triangulation de Delaunay contrainte est calculée sur l'ensemble des segments contenus dans C_s. Notons T l'ensemble des triangles consistuant cette triangulation.
2. Notons T' l'ensemble de triangles suivant:

   $$T' = \{(A,B,C) \mid ((A,B,C) \in T) \wedge (BC \in C_s) \wedge (AB \in F_s \cup C_s) \wedge (AC \in F_s) \wedge (A \notin D_p)\}$$

   Cet ensemble contient tous les triangles constitués d'un segment contraint, d'un segment libre et d'un segment contraint ou libre (voir fig. 1.5 (a)). Il ne contient donc pas de triangles dont un segment appartient à D_s.
3. Notons T'' l'ensemble des triangles de T' tels que la projection orthogonale du point A sur la droite BC appartienne au segment BC (voir fig. 1.5 (b)).
4. Pour chaque triangle (A,B,C) de l'ensemble T'', P_A, la projection orthogonale de A sur BC, est calculée (voir fig. 1.5(c)). Puis :
   - P_A est ajouté à D_p,
   - la contrainte BC est supprimée de C_s,
   - les contraintes BP_A et CP_A sont ajoutées dans C_s,
   - le segment AP_A est ajouté dans D_s.

   Le segment AP_A ainsi créé représente la plus courte distance entre le point A et le segment contraint BC.
5. Tant que l'ensemble T'' n'est pas vide, retourner en 1.

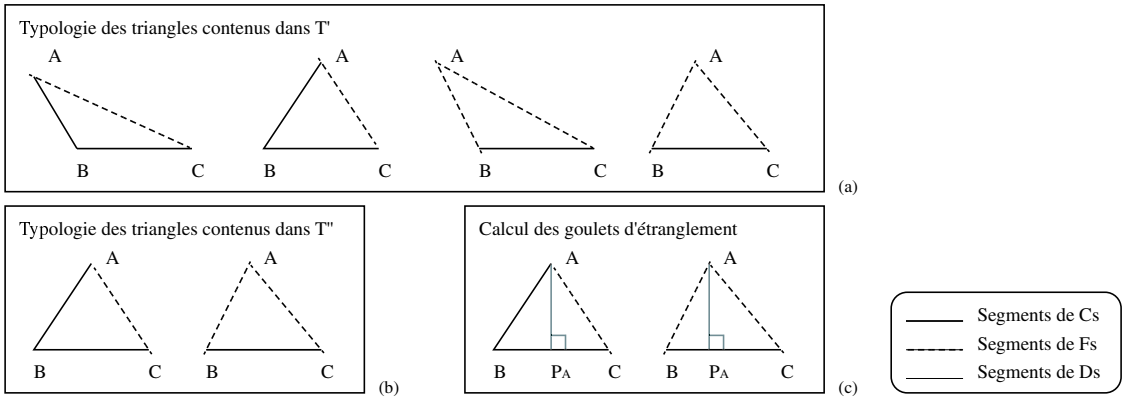


FIG. 1.5 – *Typologies des triangles manipulés durant la détection des goulets d'étranglement.*

Il est à noter que dans la mesure ou les segments de D_s et les points de D_p ne sont plus pris en compte dans l'algorithme au fur et à mesure de leur construction, cet algorithme converge. A la fin de l'algorithme, l'ensemble des contraintes de plus courte distance entre les angles et les murs est généré, identifiant ainsi les goulets d'étranglement déduits de la géométrie de la scène. Les goulets d'étranglement de type point à point sont pour leur part automatiquement générés par la triangulation, cette contrainte faisant partie intrinsèquement des propriétés de la triangulation de Delaunay.

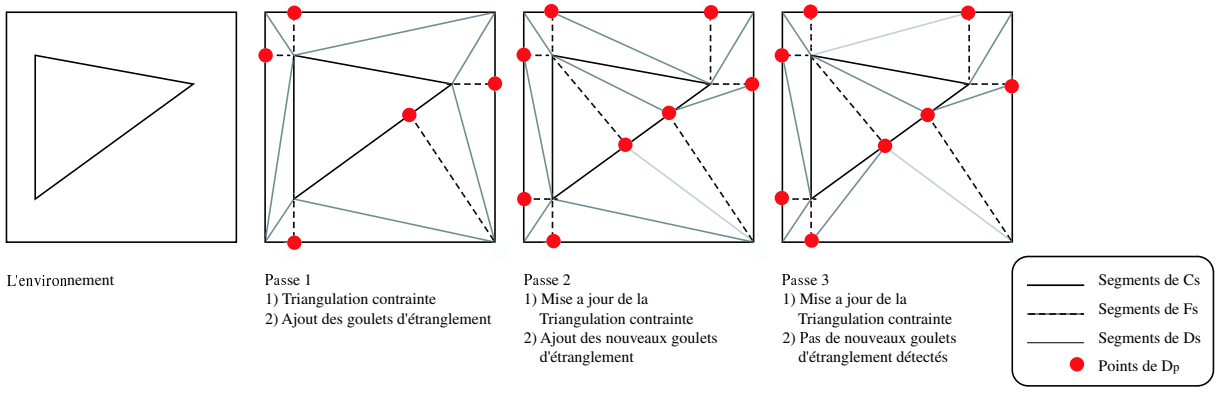


FIG. 1.6 – *Déroulement de l'algorithme de calcul des goulets d'étranglement sur un environnement simple.*

Un exemple de déroulement de l'algorithme est donné en fig. 1.6. L'algorithme se déroule en trois étapes. A chaque étape, la triangulation de Delaunay contrainte est d'abord calculée, puis les nouveaux goulets d'étranglements sont ajoutés. Grâce aux propriétés sur le voisinage des points, les différentes triangulations conservent les informations sur les goulets d'étranglement.

1.1.2.2 Calcul des cellules

En utilisant comme base la triangulation de Delaunay contrainte décrite précédemment, il est possible de réunir les triangles de manière à générer des cellules convexes. L'information existante sur les goulets d'étranglement de l'environnement n'est utile que pour spécifier les contraintes de navigation liées à l'envergure d'une entité. Elles ont donc principalement une portée sur les zones délimitant l'accessibilité d'une cellule à une autre. De fait, seuls les passages localement les plus étroits sont à prendre en compte.

En s'appuyant sur cette remarque, il est possible d'extraire deux contraintes à respecter pour pouvoir réunir des cellules pour n'en former plus qu'une seule :

1. La réunion de deux cellules convexes doit former une cellule convexe. Cette contrainte, inhérente à la forme de subdivision, doit être respectée pour assurer une navigation libre à l'intérieur d'une cellule. En effet, elle assure que deux points appartenant à la même cellule peuvent être reliés en ligne droite sans rencontrer d'obstacle.
2. Lorsque l'on souhaite réunir deux cellules convexes, ces dernières partagent un segment libre (ajouté par la triangulation de Delaunay à l'origine). Pour assurer localement la conservation des informations sur les goulets d'étranglement, la longueur du segment partagé par les deux cellules doit être supérieure à la longueur du plus long segment libre appartenant à l'enveloppe de la cellule convexe ainsi créée. De cette manière, seules les contraintes les plus fortes sont conservées.

L'algorithme qui s'en suit se révèle simple. Dans un premier temps, la triangulation de Delaunay contrainte est calculée. Ensuite, l'ensemble des segments libres utilisés pour définir les triangles sont classés suivant leur longueur. Un segment libre étant partagé par uniquement deux cellules, la liste de segments triée est parcourue dans le sens des longueurs décroissantes. Les deux cellules associées au segment courant sont testées pour savoir si leur union respecte les contraintes (1) et (2) exprimées ci-dessus. Dans le cas où les contraintes sont respectées, le segment est supprimé, les deux cellules sont aussi supprimées et remplacées par leur union.

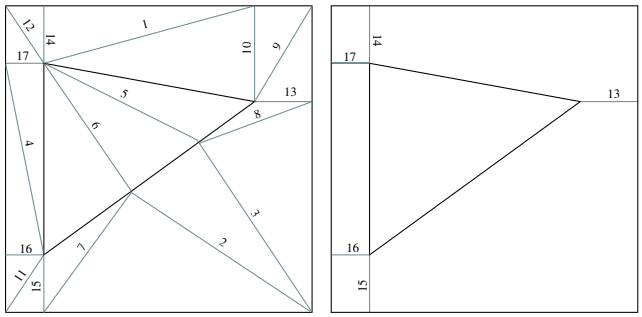


FIG. 1.7 – *Création des cellules convexes à partir de la triangulation de Delaunay contrainte sur l'environnement avec détection des goulets d'étranglement.*

La figure 1.7 montre le déroulement de l'algorithme de création de cellules convexes à partir de l'exemple de triangulation de la fig. 1.6. Le schéma de gauche représente la triangulation de

Delaunay contrainte extraite de l'étape de calcul précédente, contenant les goulets d'étranglement détectés. Les segments sont numérotés suivant l'ordre décroissant leur longueur et donc l'ordre dans lequel ils sont testés par l'algorithme de création des cellules convexes. Le schéma de droite montre le résultat obtenu après la fusion. Les goulets d'étranglement {13, 14, 15, 16, 17} ont été conservés, alors que le goulet 3 a été supprimé par la fusion des cellules. Ceci s'explique par le fait que ce goulet d'étranglement n'était pas localement le plus contraignant comparativement aux goulets 13 et 15.

Dans un cadre plus concret, la figure 1.8 montre le résultat du calcul de la subdivision en cellules convexes sur le plan de l'appartement simplifié présenté dans la figure 1.4. Ce résultat met en évidence la détection des goulets dans l'encadrement des portes, entre la table et les murs... D'autre part, ce schéma montre un cas critique de la subdivision en cellules convexes. La table ronde (en haut à gauche du plan), est un objet convexe représenté par un grand nombre de segments. Comme le montre le plan, ce type d'objet génère un nombre élevé de cellules. Cependant, il faut garder en mémoire que d'après les propriétés de la triangulation de Delaunay, le nombre total de triangles, et donc, par extension, de cellules, est au pire linéaire en fonction du nombre de sommets représentant les objets ; cette complexité est donc bornée.

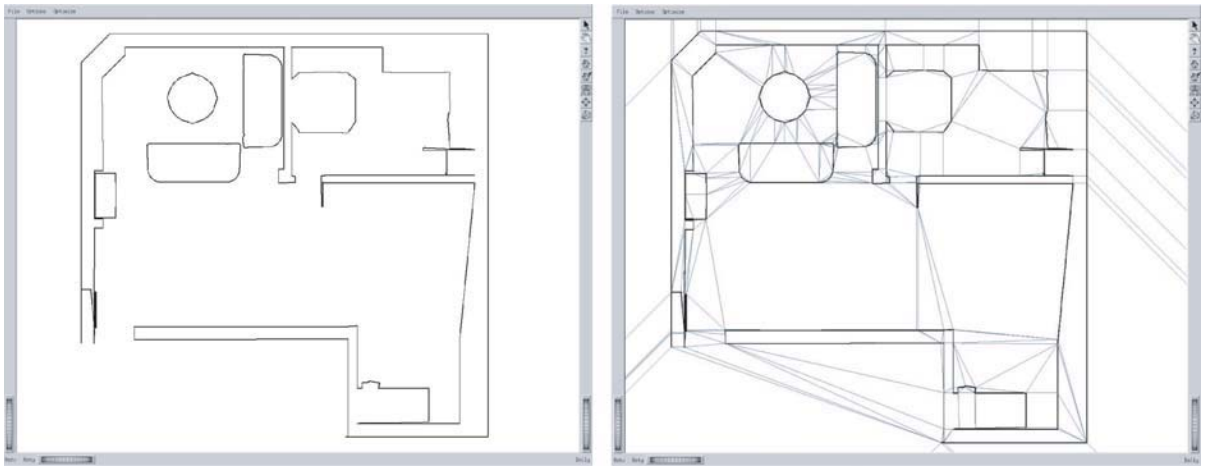


FIG. 1.8 – *Exemple de subdivision en cellules convexes sur le plan simplifié de l'appartement.*

La subdivision spatiale ainsi générée conserve localement les informations de goulets d'étranglement. Autrement dit, si un passage est étroit, relativement aux autres passages proches, il est conservé. D'autre part, l'ordre de traitement des segments donne une priorité sur la construction des cellules. Le fait de commencer par les segments les plus longs favorise la création de cellules maximisant la surface recouverte, cette propriété découle directement des propriétés de la triangulation de Delaunay contrainte qui est utilisée comme base de subdivision.

1.1.3 Propriétés de l'organisation en cellules convexes

La subdivision de l'espace en cellules convexes offre plusieurs bonnes propriétés pour son utilisation dans le cadre de la planification de chemin, de la navigation des entités et enfin des calculs de visibilité.

Liberté de déplacement. La propriété de convexité des cellules permet une navigation aisée de l'entité. En effet, tous les points appartenant à la même cellule convexe peuvent être reliés par un chemin en ligne droite ne contenant aucun obstacle. Cette propriété permet notamment de ne s'occuper que de la distance aux murs, lorsqu'une entité navigue. D'autre part, la détection des goulets d'étranglement permet de savoir, immédiatement si une entité, en fonction de son envergure peut passer d'une cellule à une autre en utilisant un segment libre donné.

Positionnement de l'entité. La notion de positionnement de l'agent à l'intérieur de la subdivision consiste à rechercher la cellule le contenant, pour être à même d'effectuer des calculs sur les obstacles proches par exemple. Ce positionnement, malgré le mode de subdivision utilisé est peu coûteux.

Lors du premier positionnement, une cellule est tirée au hasard pour devenir la cellule courante. Les segments délimitant la cellule sont alors parcourus jusqu'à ce que :

1. un segment soit trouvé tel que le produit scalaire entre la normale à ce segment dirigée vers l'intérieur de la cellule et la position relative de l'agent par rapport au segment soit négatif. Dans ce cas, l'agent n'est pas à l'intérieur de la cellule courante. La cellule connexe partageant ce segment avec la cellule courante devient la nouvelle cellule courante et l'algorithme recommence une nouvelle itération.
2. aucun segment ne satisfasse le premier cas. L'agent se trouve alors à l'intérieur de la cellule courante.

En moyenne, la complexité de ce type d'algorithme en fonction du nombre n de cellules est en $O(\sqrt{n})$. Cependant, par la suite, il est possible d'exploiter la continuité spatiale et temporelle pour localiser l'agent en fonction de la dernière cellule dans laquelle il se trouvait. La complexité devient alors $O(1)$ si l'hypothèse de continuité est respectée.

Distance aux obstacles. Lorsque l'agent est localisé à l'intérieur d'une cellule, il devient aisé de calculer sa distance par rapport aux obstacles les plus proches. Pour ce faire, il suffit de calculer sa distance aux segments *contraints* constituant la cellule le contenant. L'accès aux obstacles les plus proches possède alors une complexité linéaire en fonction du nombre de segments contraints délimitant la cellule.

Lancer de rayon et visibilité. La subdivision spatiale en cellules convexes offre une structure de données permettant de faire des calculs de lancer de rayon en deux dimensions très rapidement. Définissons un rayon par l'intermédiaire des informations suivantes:

- la position de sa source p,
- son vecteur directeur d,
- sa distance maximale de prise en compte α.

Notons $S_l(C)$ et $S_c(C)$ deux fonctions renvoyant respectivement l'ensemble des segments libres et l'ensemble des segments contraints délimitant une cellule convexe C. Notons $n(C,s)$ une fonction renvoyant le vecteur normal au segment s délimitant la cellule C et dirigé vers l'intérieur de la cellule. Enfin notons $R(\beta)$ la matrice 2×2 de rotation d'un angle β d'un vecteur à deux dimensions.

Le test d'intersection entre le rayon et un segment s, dont nous noterons A et B les positions des points extrémité, possède alors l'expression suivante :

$$(d.n(C,s) \leq 0) \wedge \left(\left(\left(\frac{A+B}{2} - p \right) \times R(\frac{\pi}{2}).(A-p) \right) \times \left(\left(\frac{A+B}{2} - p \right) \times R(\frac{\pi}{2}).(B-p) \right) \leq 0 \right)$$

En posant $M = (\frac{A+B}{2} - p) \times R(\frac{\pi}{2})$ l'expression devient alors:

$$(d.n(C,s) \leq 0) \wedge ((M.(A-p)) \times (M.(B-p))) \leq 0$$

Dans un premier temps, la cellule C contenant la source du rayon est déterminée. Ensuite, un test d'intersection est effectué pour chaque segment de $S_l(C)$. Si aucune intersection n'est trouvée avec les segments de $S_l(C)$, alors le rayon entre en intersection avec un des segments de $S_c(C)$. Dans ce cas, il suffit de déterminer ce segment en appliquant le même calcul. Dans le cas contraire où le rayon traverse l'un des segments de $S_l(C)$, il suffit de trouver la cellule adjacente à la cellule courante partageant ce segment. Le rayon est alors relancé dans cette nouvelle cellule. La prise en compte de la portée α du rayon se fait en cours de calcul par un simple test entre la distance du point d'intersection trouvé avec le segment et la position de la source du rayon.

Ce lancer de rayon, de par son faible coût de calcul, peut être utilisé intensivement à l'intérieur des applications. Il permet, notamment, de rapidement savoir si deux entités sont visibles l'une de l'autre ou bien encore d'obtenir l'obstacle le plus proche d'une position donnée, dans une direction donnée.

Discussion

Le mode de subdivision spatiale qui vient d'être présenté offre donc de très bonnes propriétés :

1. il représente très précisément l'environnement.
2. il adapte automatiquement la taille des cellules à la densité des obstacles.
3. il permet de savoir immédiatement si une entité peut emprunter un passage ou non par une simple comparaison entre son envergure et la longueur du segment délimitant le passage entre deux cellules.
4. il permet de calculer très rapidement la distance entre une entité et l'obstacle le plus proche.
5. il offre des fonctionnalités de lancer de rayon rapide, permettant d'exploiter cette technique intensivement en temps réel.

Cependant, il est possible d'objecter que le calcul de positionnement d'une entité est plus lourd que dans une discrétisation approximative de type grille régulière par exemple. Mais, dans le cas de la possibilité d'exploitation de la continuité spatio-temporelle du déplacement, la complexité est à peu près équivalente, même si le mode de calcul est pour sa part plus lourd. De plus, l'occupation mémoire nécessitée par la subdivision est uniquement dépendante de la complexité géométrique de l'environnement, mais pas de la précision de représentation que l'on souhaite obtenir. Enfin, l'efficacité des algorithmes de détection de la proximité des obstacles et de lancer de rayon permet de contrebalancer le coût associé au positionnement de l'entité.

1.2 Planification hiérarchique de chemin

La subdivision spatiale qui vient d'être présentée dispose de bonnes propriétés en terme de navigation et de représentation de l'environnement. Cependant, la structure ne peut être utilisée comme telle pour la planification de chemin. Prenons l'exemple de la représentation de l'environnement sous la forme d'une grille uniforme. La planification de chemin à l'intérieur de ce type de structure peut se faire par l'intermédiaire d'un graphe dans lequel on recherche une suite de cellules, connexes et franchissables, permettant d'atteindre un but et minimisant le nombre de cellules traversées. La

« validité » en terme de distance de ce type d'algorithme réside dans l'homogénéité de la taille des cellules. Le mode de subdivision spatiale présenté ne possède pas cette propriété car il adapte la taille des cellules à la complexité géométrique de l'environnement. L'utilisation de cet algorithme risque donc de générer des chemins minimisant le nombre de cellules mais d'une longueur les rendant particulièrement improbables.

L'idée consiste donc à exploiter les informations topologiques extractibles de la subdivision spatiale pour générer des niveaux d'abstractions topologiques. Par l'intermédiaire d'un plongement géométrique, il sera alors possible de générer une carte de cheminement, ainsi que des niveaux d'abstractions corrélés avec la topologie, pour rapidement planifier un chemin dans l'environnement.

1.2.1 Topologie

Comme précisé dans les paragraphes précédents, la subdivision de l'espace se traduit sous la forme d'un ensemble de cellules convexes, constituées de frontières. Ces frontières peuvent être, soit contraintes, soit libres. La subdivision étant un pavage de l'espace en deux dimensions représentant la carte de l'environnement, chaque frontière (hormis les « frontières du monde ») est partagée par deux cellules. Chaque frontière libre représente donc un passage d'une cellule à une autre, traduisant ainsi la topologie de l'environnement.

Graphe topologique Le graphe topologique représente la connexion des cellules par l'intermédiaire d'une zone de passage, dans notre cas une frontière libre. Par le passage de cette frontière, une entité changera de cellule ; cependant, si cette frontière n'est pas suffisamment large, elle ne pourra pas la franchir. En exploitant la propriété de la subdivision spatiale qui est de contenir les informations sur les goulets d'étranglement, il est possible de filtrer les frontières libres en fonction de l'envergure des entités. Si une frontière libre est d'une taille supérieure à l'envergure de l'entité, nous sommes assuré que l'entité pourra passer, dans le cas contraire, cette frontière sera considérée comme infranchissable.

La notion d'accessibilité entre deux zones étant la caractéristique importante pour la planification de chemin, le graphe topologique va être dépendant de l'envergure de l'entité. Chaque nœud de ce graphe représente une cellule et chaque arc, une frontière libre, franchissable par l'agent, partagée par deux cellules. Les composantes connexes de ce graphe représentent alors deux types de zones : les zones de navigation et l'intérieur des obstacles.

A titre d'exemple, la figure 1.9 montre le plan d'un appartement (simplifié) peuplé par une entité représentée par un cercle. Le plan de l'appartement est subdivisé et chaque cellule comporte son numéro ; les segments plus clairs représentent les frontières libres partagées par les cellules. Sur la droite de la figure se trouve le graphe topologique associé. Dans la mesure où l'entité possède une envergure supérieure à la longueur du passage entre les cellules 14 et 16, le graphe topologique ne comporte pas de lien entre les nœuds représentant ces cellules.

Propriétés topologiques De manière immédiate, il devient possible de classifier les cellules en fonction du nombre n de frontières libres et franchissables (autrement dit, l'arité du nœud représentant la cellule dans le graphe topologique) pour traduire leurs caractéristiques en terme de navigation. Nous distinguons quatre types de cellules (les exemples fournis par la suite se réfèrent

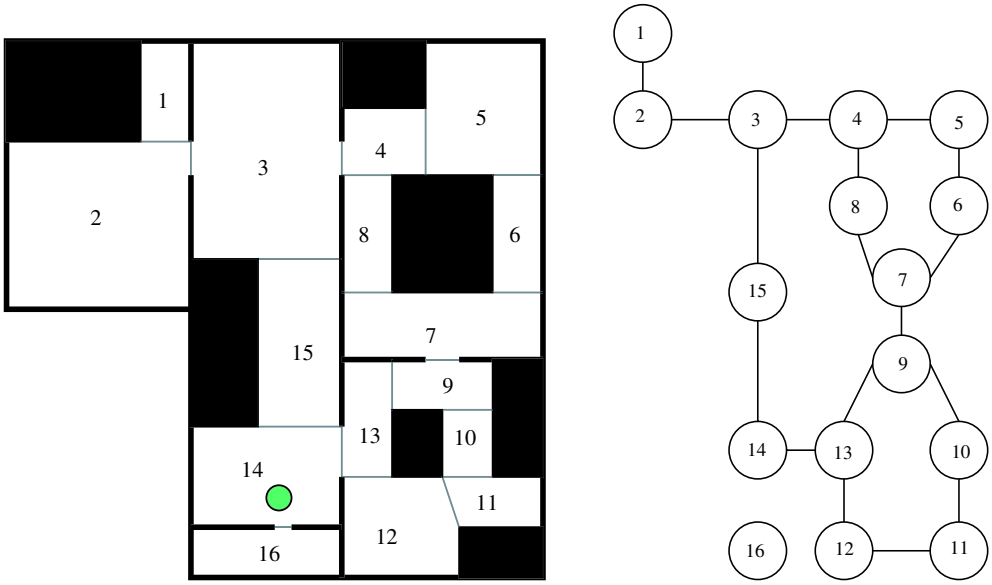


FIG. 1.9 – *Un plan d'appartement avec une entité et le graphe topologique associé.*

au graphe de la figure 1.9) :

- $n = 0$: il s'agit d'une cellule **close**. Il n'y a aucun moyen ni d'entrer ni de sortir de ce type de cellule. Dans le graphe, la cellule 16 est de type *close*.
- $n = 1$: il s'agit d'une cellule **cul-de-sac**, ce type de cellule ne devrait être utile que lorsqu'elle contient une cible. Dans le cas contraire, cette cellule ne devrait pas être traversée dans la mesure où il n'existe qu'une seule manière d'entrer et de sortir. Dans le graphe, la cellule 1 est de type *cul-de-sac*.
- $n = 2$: il s'agit d'une cellule **passage**, ce type de cellule peut être traversé en entrant d'un coté et en sortant de l'autre, par l'intermédiaire d'un chemin qui peut être considéré comme unique. Dans le graphe, les cellules {2, 5, 6, 10, 11, 12, 14, 15} sont de type *passage*.
- $n > 2$: il s'agit d'une cellule **carrefour**, un choix doit être effectué lors de l'arrivée dans ces cellules pour savoir par quelle frontière ressortir. Dans le graphe, les cellules {3, 7, 9, 13} sont de type *carrefour*.

Ces informations peuvent s'avérer plus qu'utiles en terme de navigation. Il devient possible de qualifier certaines parties du graphe topologique en fonction des types de cellules les constituant ainsi que de leurs interconnexions. Ainsi, par extension, un cul-de-sac est une suite de cellules de type *passage* connectée à une cellule de type *cul-de-sac* (les cellules {1, 2} forment un cul-de-sac). Un chemin est une suite de cellules de type *passage* (les cellules {10, 11, 12} forment un chemin).

Ce type d'information peut facilement être utilisé pour abstraire le graphe topologique de manière à générer des couches d'abstraction ayant, d'une part, une signification « cognitive » et permettant, d'autre part, d'accélérer les calculs de chemins dans la structure.

1.2.2 Abstraction du graphe topologique

Le principe de l'abstraction du graphe topologique consiste à travailler sur la typologie des cellules telle que décrite au paragraphe précédent. Précédemment, l'identification des cellules s'est principalement faite à partir du graphe topologique directement extrait de la subdivision spatiale. Ce graphe tient donc compte, indirectement, des contraintes géométriques de l'environnement. Lorsque l'on raisonne sur de la topologie, on ne se soucie plus de la géométrie mais des relations entre les zones. Nous allons donc étendre la notion de cellule par la notion de méta-cellule regroupant plusieurs cellules.

Une méta-cellule représente soit une cellule, soit une union de plusieurs méta-cellules connexes partageant deux à deux une frontière libre et franchissable. La frontière d'une méta-cellule est donc définie comme l'union des frontières des méta-cellules la constituant moins les frontières communes. Leurs propriétés topologiques sont donc identifiables à celles des cellules. En fonction de leur arité dans le graphe, elles peuvent être typées de la même manière que les cellules décrites dans le paragraphe précédent. Pour le cas de l'union de deux cellules, il est possible d'extraire la table de la figure 1.10, donnant le type d'une méta-cellule en fonction du type des deux méta-cellules la constituant.

	close	cul-de-sac	passage	carrefour
close	-	-	-	-
cul-de-sac	-	close	cul-de-sac	indéfini
passage	-	cul-de-sac	passage	carrefour
carrefour	-	indéfini	carrefour	indéfini

FIG. 1.10 – *Table de typage de l'union de méta-cellules connexes. La mention « indéfini » stipule qu'il faut effectuer un calcul sur le nombre de frontières libres de la méta-cellule pour être en mesure de la typer.*

L'algorithme d'abstraction topologique se base sur la typologie des cellules extraites ainsi que sur une heuristique portant sur le nombre de manières de traverser une cellule. Notons *arite* une fonction permettant de déterminer le nombre de frontières libres d'une cellule ou d'une méta-cellule. La fonction suivante *passages* fournit le nombre de manières de traverser une cellule en fonction du nombre de segments libres constituant sa frontière :

$$passages(C) = arite(C) \times (arite(C) - 1) \tag{1.1}$$

Il est donc possible de calculer combien de manières de traverser une méta-cellule sont ajoutées lors de l'union de deux méta-cellules. Ce calcul s'effectue par l'intermédiaire de la fonction *ajout* suivante :

$$ajout(C1,C2) = passages(C1 \cup C2) - passages(C1) - passages(C2) \tag{1.2}$$

Ces informations sont utiles lors du calcul des niveaux d'abstraction du graphe d'accessibilité. Le but de ce calcul est de générer une structure de données arborescente dans laquelle chaque nœud est une méta-cellule dont les fils sont les méta-cellules la constituant, sous la contrainte de limiter le plus possible le nombre de manières de traverser la cellule créée à chaque niveau d'abstraction.

Chacune des méta-cellules, suivant la classification du paragraphe précédent, peut être typée. Les niveaux d'abstraction vont donc être calculés en gardant une homogénéité de typage de manière à faire remonter dans l'arbre de l'information sémantique. Par exemple, une cellule de type *cul-de-sac* connectée à une suite de cellules de type *passage*, seront résumées à un plus haut niveau dans l'arbre par une méta-cellule de type *cul-de-sac*, constituée de toutes les cellules décrivant le chemin vers le cul-de-sac. D'autre part, comme nous le verrons par la suite, ces niveaux d'abstraction permettent de grandement optimiser les calculs de chemins. A la suite de l'algorithme, un exemple d'abstraction sur un graphe relativement simple est développé.

Algorithme d'abstraction. L'algorithme ici présenté, se déroule sur chacune des composantes connexes du graphe topologique. Son déroulement est caractérisé par quatre phases de calcul ; chacune d'elles pouvant être calculée plusieurs fois sur des niveaux d'abstraction différents.

***Phase 1** : Détection des passages et des cul-de-sac.*
Cette phase de l'algorithme consiste à détecter les suites de cellules interconnectées de type cul-de-sac et passage. Pour effectuer ce calcul, un sous-graphe du graphe topologique est extrait, en ne conservant que les méta-cellules de type *passage* ou *cul-de-sac*. Chaque composante connexe de ce sous-graphe représente par extension un cul-de-sac ou un chemin. Une fois ces suites détectées, un arbre d'abstraction est créé de façon à ce que chaque méta-cellule (n'étant pas une cellule feuille de l'arbre) ne soit constituée que de deux cellules et que cet arbre soit le plus équilibré possible. Dès lors, un nouveau graphe topologique est recréé, sur la base du graphe de départ, en remplaçant les suites de cellules abstraites par la méta-cellule les regroupant.

***Phase 2** : Union des culs-de-sac et des carrefours*
Après la première phase de l'algorithme, les suites de méta-cellules constituant des cul-de-sac sont regroupées sous la forme d'une seule méta-cellule elle même de type cul-de-sac et qui a la propriété d'être connectée à une cellule de type carrefour. De façon à simplifier le graphe, pour chaque carrefour connecté à une ou plusieurs cellules de type cul-de-sac, une unique cellule, constituée des cellules cul-de-sac et de la cellule carrefour, est générée. Ce traitement modifie la structure topologique car l'union des cellules cul-de-sac avec des cellules de type carrefour génère une cellule dont le type est soit cul-de-sac, passage ou carrefour. Le nouveau graphe topologique est alors généré en substituant les méta-cellules abstraites par leur abstraction. Il est donc possible que l'on puisse de nouveau trouver des chemins ou des cul-de-sac dans le graphe topologique. Donc, si au moins l'une des nouvelles cellules ainsi calculées est de type cul-de-sac ou passage, on retourne en phase 1 du traitement. Si le graphe n'est plus constitué que d'une seule cellule de type close, le traitement s'arrête. Dans le cas où aucune de ces conditions n'est satisfaite, on passe en phase 3 du traitement.

***Phase 3** : Union des chemins et des carrefours*
A ce stade de traitement, les cellules présentes dans le graphe d'accessibilité sont soit de type carrefour, soit de type passage. D'autre part, il n'existe plus de suites de cellules de type cul-de-sac ou passage à l'intérieur du graphe et s'il existe des cellules de type passage, elles sont forcément reliées à des cellules de type carrefour. Pour finir de simplifier la structure, chaque cellule de type passage est regroupée avec sa cellule de type carrefour connexe possédant la plus petite arité dans le graphe. Si plusieurs cellules de type passage se partagent le même carrefour, la méta-cellule possèdera alors plus de deux méta-cellules constituantes.

Phase 4 *: Optimisation de la structure des carrefours*
Désormais, le graphe désormais extrait des phases de traitement précédentes ne possède plus que des cellules de type *carrefour*. Pour être à même de simplifier ce graphe, une heuristique permettant de guider l'abstraction des cellules de ce type est proposée. Pour chaque couple de cellules connexes pouvant potentiellement être abstrait une heuristique fournissant un couple de valeurs est calculé. Notons C_1 et C_2 les deux cellules à abstraire, l'heuristique $h(C_1,C_2)$ est calculée de la manière suivante :

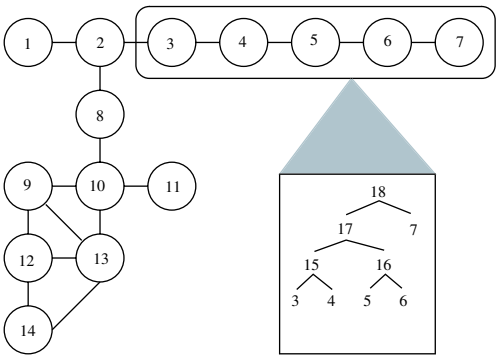
$$h(C_1,C_2) = (ajout(C_1 \cup C_2), passages(C_1 \cup C_2))$$

Pour la suite, nous utiliserons l'ordre lexicographique sur les valeurs de l'heuristique : posons $(a,b) = h(C_1,C_2)$ et $(c,d) = h(C_3,C_4)$, nous avons $h(C_1,C_2) < h(C_3,C_4) \Leftrightarrow a < c \vee (a = c \wedge b < d)$. Tous les couples de cellules connexes sont alors triés dans l'ordre croissant de la valeur de l'heuristique qui leur est associée. Deux cas sont alors à considérer :

- Il existe au moins un couple (C_1,C_2) tel que $(a,b) = h(C_1,C_2) \wedge (a < 0)$. Dans ce cas, chaque couple de cellules respectant cette contrainte est fusionné en une méta-cellule. Ce traitement s'effectue dans l'ordre croissant de l'heuristique (dans la mesure où chacune des cellules du couple n'a pas encore été traitée durant cette phase).
- Il n'existe pas de couple (C_1,C_2) tel que $(a,b) = h(C_1,C_2) \wedge (a < 0)$. Dans ce cas, notons $d = min(ajout(C_1 \cup C_2))$. Chaque couple de cellules (C_a,C_b) tel que $\exists x$, $(d,x) = h(C_a,C_b)$ est fusionné en une méta-cellule. Ce traitement s'effectue dans l'ordre croissant de l'heuristique (dans la mesure ou chaque cellule du couple n'a pas encore été traitée durant cette phase). Ce cas produit donc des méta-cellules augmentant le nombre de passages par rapport aux cellules sous-jacentes.

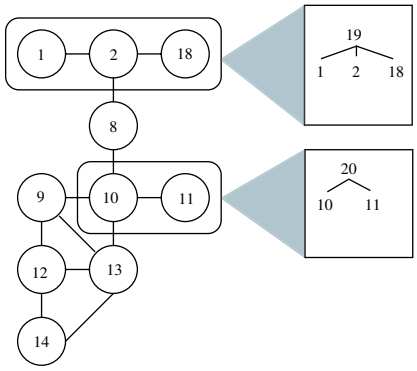
Finalement, si cette phase de calcul génère une cellule de type *close*, l'algorithme s'arrête. Sinon, si au moins une cellule de type *passage* ou *cul-de-sac* a été générée, on retourne en phase 1 du traitement. Enfin, si aucun de ces deux cas n'est satisfait, la phase 4 de calcul est répétée.

Déroulement de l'algorithme sur un exemple.

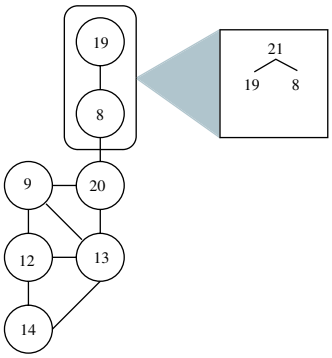


Passe 1 : Sur la gauche, un graphe topologique extrait d'un environnement est représenté. La phase 1 de l'algorithme a pour rôle d'extraire les suites de plusieurs cellules de type *passage* ou *cul-de-sac* pour les abstraire en une seule méta-cellule. La suite de cellules {3, 4, 5, 6, 7} est donc abstraite sous la forme d'un arbre binaire dont la racine (la méta-cellule englobant toutes les autres sous cellules) est la cellule 18, de type *cul-de-sac*. Le graphe dans lequel cette suite de cellules est remplacée par la cellule 18 est présenté ci-dessous et est utilisé pour la phase 2 de calcul.

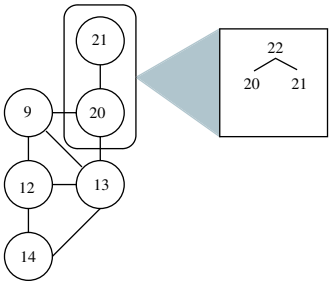
Passe 2 : La deuxième passe de calcul exécute la phase 2 de l'algorithme. Ici, le but est de supprimer tous les nœuds de type *cul-de-sac*, et de les réunir avec leurs voisins de type *carrefour*. Dans le cas du graphe extrait par la passe 1 de l'algorithme, les cellules 1, 18 et 11 sont de type *cul-de-sac*. Les cellules 1, 18 sont connexes à la cellule 2, ces trois cellules sont donc abstraites sous la forme de la méta-cellule 19. De la même façon, les cellules 11 et 10 sont abstraites au travers de la cellule 20.

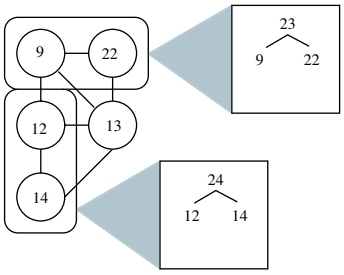


Passe 3 : Cette troisième passe de calcul exécute de nouveau la phase 1 de l'algorithme. En effet, au cours de la passe précédente, une cellule de type *cul-de-sac* a été générée. Elle permet de détecter la suite de cellules 19, 8 de types respectifs *cul-de-sac* et *passage*. Ces deux cellules sont donc abstraites par l'intermédiaire de la cellule 21 et remplacées par cette dernière pour générer le graphe de la passe 4.

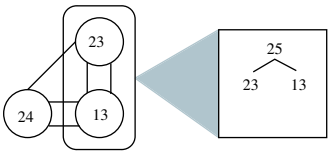


Passe 4 : Cette passe exécute de nouveau la phase 2 de l'algorithme. Une nouvelle cellule de type *cul-de-sac* ayant été générée, elle va être fusionnée à sa cellule connexe de type *carrefour*. Dans le cas présent, les méta-cellules 20 et 21 sont abstraites par l'intermédiaire de la méta-cellule 22. Le graphe alors généré ne possède plus de suites de cellules connexes de type *cul-de-sac* ou *passage*.





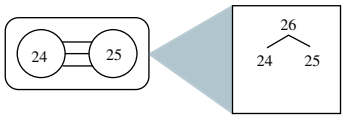
Passe 5 : Cette passe exécute la phase 3 de l'algorithme. Le principe est donc de réunir les cellules de type *passage* avec les cellules connexes de type *carrefour*. La cellule 23 est voisine du carrefour 9 d'arité 3 et du carrefour 13 d'arrité 4. Dans ce cas, elle est donc abstraite via la méta-cellule 23 avec le carrefour 9 car il possède la plus petite arité. De même, les cellules 14 et 12 sont abstraites via la cellule 24.



Passe 6 : Le graphe n'est plus constitué que de cellules de type *carrefour*. La phase 4 de l'algorithme est alors utilisée et chaque paire de cellules connexes se voit attribuée un couple de valeurs. Notons (C_1, C_2) un couple de cellules connexes, le couple de valeurs associé est $h(C_1, C_2) = (ajout(C1 \cup C2), passages(C1 \cup C2))$. Dans le cas de cet exemple, trois couples de cellules existent et se voient attribués ce couple de valeurs :

h(13, 23) = (12-12-6, 12) = (-6, 12)
h(13, 24) = (12-12-6, 12) = (-6, 12)
h(23, 24) = (12-6-6, 12) = (0, 12)

Il existe alors deux couples de cellules minimisant les couples de valeurs associées (13, 23) et (13, 24). Parmi ces deux couples, le couple (13, 23) est sélectionné de manière non déterministe pour être abstrait via la méta-cellule 25. Le couple (13, 24) ne peut donc plus être abstrait dans cette passe car la cellule 13 a déjà été utilisée. Il en est de même pour le couple (23, 24) car la cellule 23 a déjà été utilisée.



Passe 7 : Il s'agit de la dernière passe de l'algorithme. La phase 4 est encore exécutée puisqu'aucune cellule de type cul-de-sac ou passage n'a été générée. Cette phase réunit le couple de cellules (24, 25) via la méta-cellule 26. L'algorithme s'arrête alors car cette cellule est de type close et ne peut donc plus être abstraite.

Cet exemple montre le déroulement de l'algorithme d'abstraction et met en évidence son fonctionnement consistant à simplifier au maximum la structure topologique en créant différents niveaux d'abstraction corrélés avec la nature topologique des cellules. La figure 1.11 montre l'arbre d'abstraction créé durant cet exemple. L'environnement peut alors être considéré à différents niveaux (en fonction de la position spatiale de l'entité par exemple), permettant ainsi de détecter, comme dans le cas de la méta-cellule 21, les cul-de-sac étendus. Cette faculté est intrinsèque à l'algorithme car à chaque phase de calcul, le but est de générer des méta-cellules possédant la plus faible arité possible. L'arbre abstrait permet donc de considérer à haut-niveau une zone topologique de l'environnement comme une seule cellule tout en qualifiant ses propriétés topologiques vues de l'extérieur. En prenant l'exemple des jeux de stratégie, il devient possible de détecter les zones offrant les plus grandes possibilités de déplacement correspondant aux zones les plus exposées et les plus difficiles à contrôler. D'autre part, les zones correspondant à un plus haut niveau à des cul-de-sacs sont isolables en maîtrisant un unique passage. L'exploitation des niveaux d'abstraction offre un grand nombre d'informations utiles pour toutes les formes de stratégies basées sur l'exploitation de l'espace.

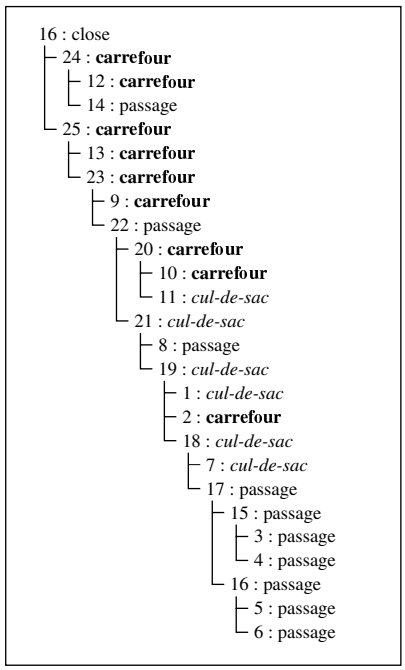


FIG. 1.11 – *L'arbre d'abstraction topologique généré à partir du graphe topologique de cet exemple. Les fils de chaque nœuds représente les méta-cellules constituant cette cellule. Pour chaque cellule, son type est spécifié.*

Comme nous allons le voir par la suite, l'algorithme de planification de chemin va pouvoir exploiter les propriétés de l'arbre d'abstraction topologique. Ce faisant, il sera donc possible de considérer des chemins constitués de plusieurs cellules sous la forme d'un seul passage regroupant ces cellules. D'autre part, si à un niveau d'abstraction suffisant, une zone est considérée comme un cul-de-sac, elle peut ne pas être prise en compte dans la planification de chemin si elle ne contient ni l'entité ni le but associé au déplacement.

1.2.3 Planification hiérarchique de chemin

Jusqu'à maintenant, l'intérêt s'est porté sur la structure topologique extraite du partitionnement de l'environnement. Le but était alors d'abstraire la topologie tout en conservant les informations sémantiques de carrefour, cul-de-sac ou passage. Cette structure, même si elle traduit le « squelette » topologique de l'environnement à plusieurs niveaux d'abstraction, ne peut être exploitée

directement pour planifier un chemin d'un point à un autre de l'environnement. En effet, elle ne s'occupe que d'information topologique au « détriment » de l'information géométrique de distance. De fait, elle ne permet pas d'obtenir directement un chemin que l'on pourrait qualifier de cohérent, c'est à dire tentant de minimiser, en partie, la distance parcourue du point de départ jusqu'au but. Notre solution consiste donc à générer un graphe de cheminement (ou *roadmap*), s'appuyant sur la structure de la subdivision spatiale et les informations qu'elle contient. Une fois ces informations calculées, elle pourront être abstraites, en suivant le schéma d'abstraction du graphe topologique pour mener à une optimisation des algorithmes de planification de chemin.

1.2.3.1 Graphe de cheminement et abstraction

Il s'agit d'augmenter la structure topologique avec des informations de nature géométrique permettant de traduire des informations de distance d'un point à un autre. Une fois ces informations géométriques extraites, il devient possible de créer un graphe de cheminement contenant des informations de distance qui seront nécessaires à l'algorithme de planification de chemin. Chaque nœud de ce graphe représente un point de passage généré à l'intérieur de l'environnement. Le lien entre deux points de passage, représenté par un arc dans le graphe de cheminement, indique l'existence d'un chemin en ligne droite, dépourvu d'obstacles, permettant de passer d'un point de passage à un autre.

Graphe de cheminement

Le calcul de la subdivision spatiale de l'environnement s'est appuyé sur une contrainte forte de convexité des cellules générées. Cette contrainte devient désormais utile. En effet, grâce à cette convexité, deux points appartenant à la même cellule peuvent être rejoints en ligne droite, sans rencontrer d'obstacles statiques.

Dans un premier temps, des points de passage sont générés sur les frontières libres de chacune des cellules directement extraites de la subdivision spatiale. Notons k le nombre de points de passage générés par frontière libre et n le nombre de frontières libres de la cellule C. Dans la mesure où la cellule est convexe, pour chaque point de passage généré sur une frontière, il existe, à l'intérieur de la cellule, des chemins, en ligne droite et sans obstacle, reliant chacun d'entre eux. Le nombre de chemins orientés permettant de traverser la cellule est donc :

$$kn(kn-k) = n(n-1)k^2 = k^2 passages(C) \tag{1.3}$$

A chaque cellule, on associe l'ensemble des chemins permettant de la traverser. Il s'agit donc de toutes les lignes droites reliant deux à deux les points de passage appartenant à des frontières libres différentes. Il est à noter que les points de passage étant générés sur des frontières partagées par deux cellules, tous les chemins appartenant à la même composante connexe sont interconnectés et couvrent « tout » l'espace de navigation. La figure 1.12 montre un exemple de graphe de cheminement généré sur le plan de l'appartement de la figure 1.9. Pour la génération, un seul point de passage a été considéré sur le milieu de chaque frontière libre et franchissable des cellules.

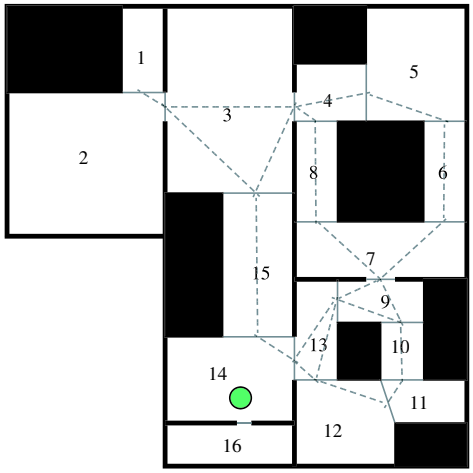


FIG. 1.12 – *Un graphe de cheminement généré sur le plan de l'appartement de la figure 1.9.*

Abstraction du graphe de cheminement

Dans la mesure où les méta-cellules sont constituées de méta-cellules partageant au moins une frontière libre, il existe, au même titre que pour les cellules, des chemins permettant d'interconnecter tous les points de passage appartenant aux frontières libres d'une méta cellule donnée. Le principe consiste donc à pré-calculer l'ensemble des chemins permettant de traverser les méta-cellules constituant l'abstraction du graphe topologique. Comme décrit dans la formule 1.3, le nombre de chemins générés à l'intérieur des méta-cellules est proportionnel au nombre de passages permettant de la traverser (si l'on considère que k est constant). Dans la mesure ou l'abstraction du graphe topologique se base sur une diminution du nombre de manières de traverser des méta-cellules, le nombre de chemins générés est donc limité grâce au lien entre le graphe de cheminement et la structure topologique.

Pour pré-calculer, à chaque niveau de la hiérarchie d'abstraction, le plus court chemin d'un point de passage à un autre à l'intérieur d'une méta cellule, nous nous basons sur les chemins déjà existants dans les méta cellules filles. Autrement dit, à chaque niveau d'abstraction, l'information du niveau d'abstraction inférieur est utilisée pour planifier le plus court chemin. Ces informations sont stockées à l'intérieur de la cellule en référençant les chemins internes aux cellules filles pour une économie de mémoire. Comme nous allons le voir, ce lien très fort entre l'abstraction topologique et l'abstraction du graphe de cheminement permet d'optimiser grandement la planification de chemin.

1.2.3.2 Planification hiérarchique

Le graphe de cheminement associé aux cellules directement extraites de la subdivision spatiale s'avère contenir un grand nombre de chemins. La complexité de calcul des chemins étant dépendante, d'une part, du nombre de sommets et, d'autre part, du nombre d'arcs du graphe utilisé, il est intéressant de limiter sa taille. Cependant, cette limitation ne doit pas influencer la qualité du résultat obtenu.

```
Ensemble<MetaCellule> abstraire(MetaCellule cell)
début
    MetaCellule tmpCell = cell
    Ensemble<MetaCellule> resultat
    tant que défini(père(tmpCell)) faire
        resultat = resultat ∪ { tmpCell }
        tmpCell = père(tmpCell)
    fin tant que
    retourne resultat
fin
```

FIG. 1.13 – *Algorithme d'abstraction d'une méta-cellule. Cet algorithme collecte toutes les méta-cellules dont la méta-cellule passée en paramètre est un constituant.*

Pour ce faire, la structure du graphe d'accessibilité abstrait ainsi que les informations de chemin stockées à l'intérieur des cellules sont utilisés. Le principe consiste à utiliser la structure topologique abstraite pour localiser à la fois l'entité faisant une requête de planification de chemin et le but qu'elle s'est fixée, en raisonnant de manière plus concrète au niveau du départ et de l'arrivée et en utilisant de l'information plutôt abstraite dans la phase intermédiaire liant le point de départ au point d'arrivée. L'algorithme se déroule donc en deux phases :

- en premier lieu, le graphe de planification « minimal » est calculé à partir de la structure topologique abstraite ;
- dans un second temps, un algorithme du type A* est utilisé pour rechercher le plus court chemin.

Vérification d'existence du chemin

Avant de pouvoir calculer un chemin de manière effective, il est important de pouvoir vérifier l'existence de ce dernier pour un coût limité. En effet, un algorithme comme A^* devra explorer l'intégralité du graphe de planification pour être en mesure de dire qu'un tel chemin n'existe pas. Ce processus s'avère excessivement coûteux. Dans notre approche, l'utilisation des informations fournies par les niveaux d'abstraction du graphe topologique permet d'obtenir cette réponse rapidement. La fonction *abstraire*, dont l'algorithme est décrit figure 1.13, calcule l'ensemble des méta-cellules appartenant au graphe topologique abstrait, dont la cellule passée en paramètre est un constituant. Dans la mesure où l'algorithme s'appuie sur les niveaux d'abstraction topologiques et que ces niveaux d'abstraction s'appuient sur la connexité des cellules, s'il existe un chemin, les cellules qu'il traverse appartiennent à la même composante connexe du graphe. Notons C_e la cellule contenant l'entité et C_b la cellule contenant le but. La condition d'existence d'un chemin devient donc :

$$abstraire(C_e) \cap abstraire(C_b) \neq \emptyset \tag{1.4}$$

La complexité du calcul associé à la fonction *abstraire* dépend de la hauteur de l'arbre d'abstraction topologique. Cependant, pour diminuer encore ce coût de calcul, il est possible de construire une

table de précalcul associant à chaque cellule la composante connexe à laquelle elle appartient. Dans ce cas, le résultat est immédiat. Cependant, comme nous allons le voir, la fonction *abstraire* est aussi utile pour l'algorithme de calcul du graphe topologique minimal.

Génération du graphe topologique minimal

Les chemins reliant les points de passage étant liés aux cellules qu'ils traversent, la minimisation du nombre de cellules minimise le nombre de chemins dans le graphe de planification. Le problème se rapporte donc au problème de trouver un ensemble minimal de méta-cellules permettant de créer le graphe de planification. D'autre part, pour être en mesure de planifier le chemin, le graphe topologique doit contenir les cellules C_e et C_b. Pour ce faire, l'algorithme de la fonction *cellulesMinimal* décrit fig 1.14 est utilisé. L'ensemble minimal de cellules est fourni par la formule suivante :

$$cellulesMinimal(\{C_e, C_b\}, abstraire(C_e) \cap abstraire(C_b)) \tag{1.5}$$

L'ensemble de méta-cellules ainsi calculé est le plus abstrait possible. Cet algorithme décompose toutes les méta-cellules contenant les cellules C_e et C_b, en arrêtant cette décomposition sur toutes les méta-cellules ne les contenant pas.

Planification

La planification s'effectue sur le graphe de planification extrait de l'ensemble de cellules minimal donné par la formule 1.5. Notons C_{min} l'ensemble de cellules extrait par cette formule. Le graphe est construit à partir de la liste des chemins contenus dans chacune des cellules de C_{min}. Un nœud du graphe représente donc un point de passage appartenant à l'une des frontières libres commune à deux cellules de C_{min} et un arc représente un chemin à l'intérieur d'une cellule reliant deux points de passage. L'arc est aussi informé par la longueur du chemin associé.

Ensuite, il faut connecter l'entité ainsi que son but au graphe de planification. Pour ce faire, un nœud correspondant à la position de l'entité est généré et est connecté à tous les points de passage appartenant aux frontières de la cellule contenant l'entité. Il en est de même pour le but. L'utilisation d'un algorithme A^* sur le graphe ainsi généré permet de trouver un chemin optimal pour un coût de calcul raisonnable dû à la limitation de la taille du graphe par l'utilisation de l'algorithme d'abstraction. L'une des conséquences de l'algorithme d'abstraction topologique est de supprimer toute forme de cul-de-sac lors de l'abstraction. L'algorithme A^* se trouve donc dans une configuration optimale pour sa recherche de chemin[2]. D'autre part, le nombre de nœuds présents dans le graphe est très restreint, cette propriété diminue donc considérablement le coût de gestion de la queue triée conservant les nœuds ouverts lors de l'exécution de l'algorithme.

Le chemin calculé se traduit finalement par la liste des cellules traversées ainsi que la liste des points de passage utilisés. Les sous-chemins induits par la liste de points de passage sont en très grande partie abstraits. Autrement dit, il s'agit de chemins traversant des méta-cellules et donc étant la résultante de la concaténation de chemins appartenant aux cellules extraites lors de la

2. Il existe un cas dans lequel des cul-de-sac peuvent exister : il s'agit du cas où le but et/ou l'entité se trouvent eux-même à l'intérieur d'un cul-de-sac.

```
Ensemble<MetaCellule>
cellulesMinimal(Ensemble<MetaCellule> init, MetaCellule C_top)
début
    Ensemble<MetaCellule> resultat, eviter, traitement
    resultat=init
    traitement = C_top
    tant que init ≠ ∅ faire
        MetaCellule tmp = extraire(init)
        eviter = eviter ∪ (abstraire(tmp) - {tmp})
    fin tant que
    tant que traitement ≠ ∅ faire
        MetaCellule tmp = extraire(traitement)
        si tmp ∈ eviter alors
            traitement = traitement ∪ fils(tmp)
        sinon
            resultat = resultat ∪ { tmp }
        fin si
    fin tant que
    retourne resultat
fin
```

FIG. 1.14 – *Algorithme de calcul du graphe minimal. Cet algorithme calcul le graphe minimal, en fonction des abstractions du graphe topologique, contenant explicitement l'ensemble des cellules passées en paramètre.*

subdivision spatiale. Pour être en mesure d'assurer une bonne navigation, exempte de problèmes de minima locaux, lorsqu'une entité arrive près d'un point de passage, il faut assurer la visibilité du prochain point de passage. La dernière phase consiste donc à concrétiser ces chemins abstraits. Pour ce faire, le chemin abstrait est décomposé en s'aidant du pré-calcul des chemins les plus courts à l'intérieur des cellules. Le chemin alors généré peut être exploité sous deux formes : la liste des points de passages visibles deux à deux ou la liste des segments franchissables ayant permis leur génération. La seconde forme, comme nous le verrons dans le prochain chapitre, offre une plus grande liberté de déplacement aux humanoïdes.

1.3 Cas de test : le centre ville de Rennes

Pour tester le gain apporté lors de la planification de chemin, nous avons généré la subdivision spatiale du centre ville de Rennes (fig. 1.15). La base géométrique est constituée de 212831 sommets et de 124815 triangles. Elle modélise 2600 bâtiments sur une surface carrée de 1,3 km de coté. Les environnements ouverts, tels que le centre ville de Rennes, constituent l'un des cas où les gains

possibles à obtenir avec l'algorithme d'abstraction sont les plus faibles. Cela s'explique par le peu de cul-de-sacs que la ville contient.

La subdivision spatiale de la base géométrique a généré 8165 cellules convexes décrites par 18444 segments (8005 segments contraints et 10439 segments libres). Grâce aux simplifications effectuées lors de la subdivision, le nombre de composantes connexes du graphe topologique est de 436 au lieu des 2600 bâtiments présents dans la base originale. Cette phase de l'algorithme a donc détecté les blocs de bâtiments et supprimé les contraintes internes à ces blocs car elles ne faisaient pas partie de la zone de navigation.

FIG. 1.15 – *Une vue d'ensemble du modèle géométrique du centre ville de Rennes et la subdivision spatiale générée par l'algorithme.*

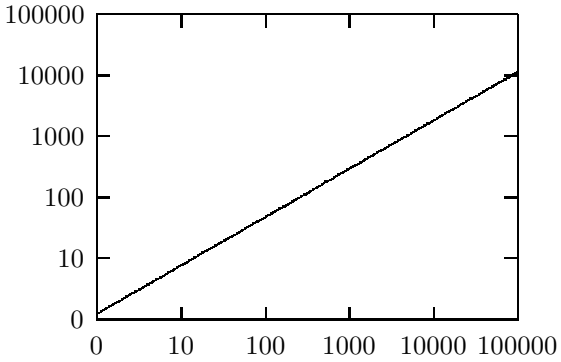


FIG. 1.16 – *Comparaison du nombre d'arcs explorés par l'algorithme A^* entre le graphe de planification complet, en abscisse, et abstrait, en ordonnée. Le tracé utilise une échelle logarithmique.*

Le gain obtenu grâce à l'utilisation de niveaux d'abstraction a été testé en utilisant cette carte. Pour chaque frontière libre de cellule, deux points de passage ont été générés pour les calculs de graphe de navigation. Le graphe complet est constitué de 85720 arcs représentant des chemins orientés. Le gain entre la planification de chemin utilisant l'abstraction et la planification de chemin

est montré figure 1.16. Ces statistiques ont été générées en utilisant un ensemble de 1319354 chemins. Puis, une régression linéaire a été calculée sur le logarithme du nombre d'arcs testés durant les deux explorations de l'algorithme A^* (graphe complet, graphe abstrait). Notons E_a le nombre d'arcs testés dans le graphe abstrait et E_c le nombre d'arcs testés dans le graphe complet. La régression linéaire nous a fourni la formule suivante :

$$log_{10}(E_a) = 0.792076 \times log_{10}(E_c) + 0.219315 \tag{1.6}$$

Cette formule montre que l'algorithme d'abstraction permet de réduire l'ordre de grandeur du nombre d'explorations durant la phase de planification. A titre d'exemple, la figure 1.17 montre une comparaison des temps de calculs de la planification de chemin entre l'algorithme A^* sur le graphe complet et l'algorithme A^* sur le graphe abstrait avec concrétisation du chemin (les tests ont été effectués sur un ordinateur équipé d'un processeur Athlon XP 1800+). Le temps maximal de planification de chemin ne dépasse alors pas 2.5ms sur le graphe abstrait, contre 40ms pour le graphe complet. Il devient donc possible de planifier des chemins dans de très grands environnements, en temps réel.

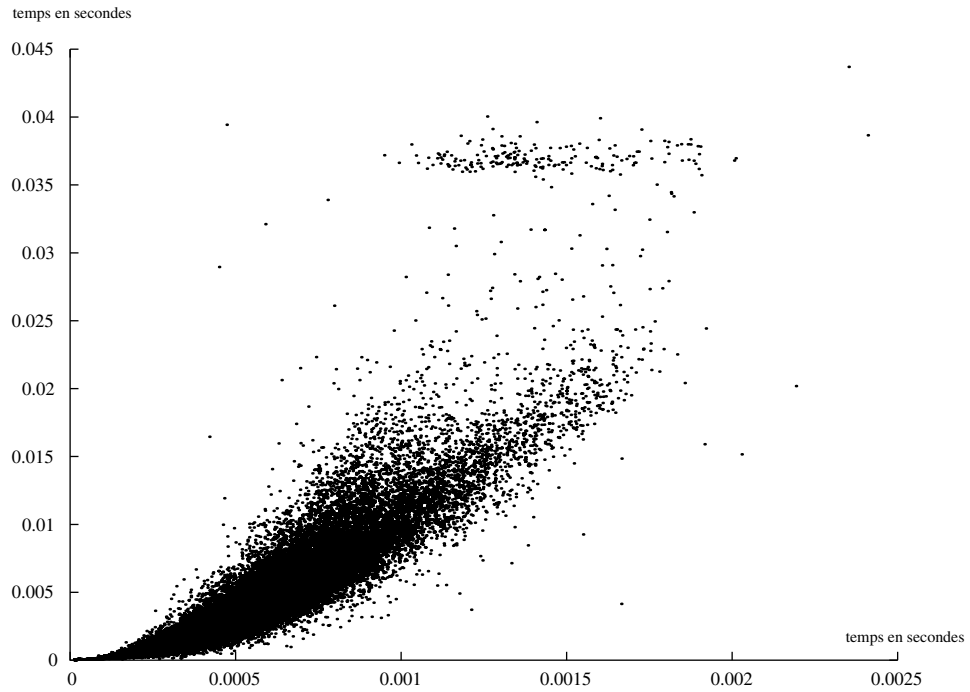


FIG. 1.17 – *Comparaison des temps de planification entre l'algorithme A^* sur le graphe complet (en ordonnée) et l'algorithme A^* sur le graphe abstrait avec concrétisation du chemin (en abscisses).*

Conclusion

L'objectif de ce chapitre était d'extraire, automatiquement, à partir du modèle $3d$ d'un environnement statique, des informations permettant son peuplement par des entités autonomes et de

permettre à ces entités de planifier un chemin à l'intérieur de ce même environnement. Cette approche s'est donc scindée en deux parties : une subdivision spatiale exacte, avec calcul des goulets d'étranglements et un algorithme de planification de chemin hiérarchique, exploitant les informations topologiques, fournies par la subdivision.

Nous sommes partis du postulat que les entités évoluent sur un sol plat. Cette contrainte, certes restrictive, permet tout de même de gérer un grand nombre d'environnements intérieurs et extérieurs. Le mode de subdivision spatiale proposé dispose de plusieurs bonnes propriétés.

- Il permet d'effectuer une discrétisation exacte de l'environnement. Sa complexité de représentation dépend donc directement de la complexité géométrique intrinsèque de l'environnement et non de la précision de la discrétisation de l'environnement comme dans les méthodes approximatives. La taille des cellules s'adapte automatiquement à la densité des obstacles, tout en essayant de maximiser l'espace dans lequel la navigation de l'entité n'est pas contrainte.
- La détection automatique des goulets d'étranglement permet de savoir, par une simple requête sur la largeur d'une frontière, si une entité peut passer d'une cellule à une autre. Cette propriété nous permet, lors de la planification de chemin, d'être assuré qu'une entité peut emprunter le chemin planifié sans être confrontée au problème des passages trop étroits.
- La propriété de convexité des cellules permet de considérablement optimiser les calculs de lancer de rayon. Cette technique peut alors être utilisée intensivement pour des tests de visibilité.
- L'accès aux obstacles proches de l'entité est très rapide et se fait par une simple consultation des frontières contraintes d'une cellule et éventuellement des cellules voisines.

Outre ces bonnes propriétés algorithmiques, il offre une structure de données permettant de raisonner sur la topologie de l'environnement. Ce raisonnement topologique permet de créer des niveaux d'abstraction topologiques permettant de qualifier des zones complètes de l'environnement. Cette propriété est à la base de l'optimisation de l'algorithme de planification de chemin et permet de changer l'ordre de grandeur de la complexité de ce traitement. Il est alors possible de gérer de très grands environnements, complexes, pour un surcoût de calcul très faible. Il devient notamment possible de planifier un chemin à l'intérieur de villes modélisées avec l'intérieur des bâtiments, sachant que s'ils constituent un cul-de-sac, ils ne seront explorés que si cela s'avère nécessaire (présence du but ou de l'entité à l'intérieur de l'un d'entre eux). Il est à noter que durant ce chapitre, aucune différence n'a été faite entre environnement intérieur et environnement extérieur car l'algorithme ne fait lui-même aucune différence et s'adapte à la géométrie qui lui est fournie sans aucune hypothèse de *forme*.

Son principal défaut reste l'hypothèse de départ : la navigation sur sol plat. Les entités ne sont donc pas à même d'emprunter des escaliers ou de naviguer à l'intérieur d'un immeuble avec des étages. Ce type de navigation doit être traité au cas par cas, mais n'est pas automatisé. Cependant l'extension en $3d$ est possible, elle réside dans l'utilisation de la même structure de subdivision, mais plutôt que d'être basée sur des triangles en deux dimensions, elle est basée sur des prismes (de base triangulaire au sol) dont la hauteur dépend des contraintes géométriques de sol et de plafond. Une fois la subdivision spatiale $3d$ calculée, tous les algorithmes qui ont été décrit sur les propriétés topologiques ainsi que l'optimisation de la planification de chemin restent valables.

Chapitre 2

Navigation réactive

Dans le chapitre précédent, nous sommes partis d'un modèle géométrique $3d$ d'un environnement pour créer une subdivision spatiale permettant à une entité de se représenter les obstacles statiques avec lesquels elle peut entrer en interaction. Une entité est donc à même de planifier son chemin dans l'environnement et de détecter rapidement les obstacles dont elle est proche. Si ces informations sont suffisantes pour gérer la navigation d'une seule entité, elles ne permettent pas de gérer la navigation des centaines d'entités qui peuvent, par exemple, peupler une ville. En effet, les interactions que cela implique, ne sont plus seulement de la forme d'une entité avec son environnement statique mais aussi de la forme d'une entité avec les autres entités. Un nouveau problème est alors soulevé, quel modèle peut-on utiliser pour traduire un comportement de navigation réaliste et permettant de gérer des centaines de piétons en temps réel ? Cette question contient plusieurs sous-problèmes :

1. une entité ayant préalablement planifié un chemin, doit être à même de le suivre.
2. une entité, lorsqu'elle navigue doit avoir conscience des entités voisines pour être à même de prédire d'éventuelles collisions et être en mesure de les éviter.
3. lorsqu'une collision est prédite, l'entité doit réagir de manière à l'éviter.

Dans ce chapitre, nous allons traiter ces problèmes en proposant une architecture de navigation réactive configurable. Cette architecture nous permet de gérer la navigation de plusieurs centaines de piétons tout en permettant de prendre en compte diverses règles issues de l'analyse du comportement piétonnier. Ces même règles pourront être utilisées à la fois en environnement extérieur et en environnement intérieur, dans la mesure où le mode de représentation de l'espace ne différencie pas ces deux types d'environnement.

2.1 Architecture de navigation

D'après diverses études sociologiques et cognitives [LW92, RQ98, Gof71], la navigation de l'être humain s'avère être caractérisée par un compromis entre plusieurs règles de comportement. Ces règles se déclinent en trois catégories :

- **Optimisation de la trajectoire.** Lorsqu'il navigue, l'être humain a tendance à optimiser sa trajectoire en minimisant l'angle entre sa direction et sa cible, tout en empruntant un chemin minimisant la dépense énergétique. D'autre part, la détermination de la cible courante (i.e. de la direction vers laquelle se diriger), se fait par l'utilisation de la perception. Autrement dit, l'homme a tendance à se diriger plus ou moins en ligne droite vers la zone visible appartenant à son chemin, si le chemin ainsi créé est libre d'obstacles. Cependant, dans la réalité, la ligne

droite est une tendance mais la trajectoire s'avère exhiber une légère courbure qui pourrait être due à l'inertie du piéton [BJ03].

- **Respect de l'espace personnel.** Chaque piéton, lorsqu'il navigue dans un environnement peuplé cherche à maintenir une certaine distance par rapport aux obstacles statiques et aux autres piétons peuplant l'environnement. Cette distance caractérise l'espace personnel, une zone libre autour du piéton dans laquelle seules les personnes appartenant au cercle des connaissances sont admises. Le non-respect de cette zone peut traduire deux types de situation. Soit il s'agit d'un groupe de piétons évoluant ensemble. Soit la densité de population est trop élevée pour pouvoir respecter cette contrainte. Cette contrainte s'avère donc être une contrainte lâche dont le respect dépend grandement de la densité de population. En environnement très dense, elle n'est plus applicable.
- **Évitement de collision.** Chaque piéton naviguant dans une zone prend en compte les autres piétons de son voisinage proche et adopte une réaction en fonction d'une éventuelle collision détectée. Cette réaction dépend d'un certain nombre de règles sociales [Tho99] ainsi que de la configuration de l'interaction : collision de face, collision arrière...

Les règles de respect de l'espace personnel ainsi que d'évitement de collision ont besoin d'informations sur le voisinage de l'entité. Ce voisinage se traduit par un ensemble d'entités, topologiquement proches de l'entité concernée par la navigation, avec lesquelles une interaction éventuelle peut se produire. La règle de respect de l'espace personnel travaille sur une aire d'influence à rayon limité autour de l'entité alors que la règle d'évitement de collision travaille en priorité sur un temps de prédiction de collision, dépendant grandement de la densité de population. Ces propriétés soulèvent le problème crucial de la détection des voisins d'une entité et surtout de sa complexité.

En se basant sur cet ensemble de règles à respecter et sur l'ensemble des informations nécessaires à leur respect, une architecture de navigation a été développée (voir fig. 2.1). Cette architecture se base sur deux parties distinctes en terme de calcul :

- le **graphe de voisinage**, qui permet d'accéder rapidement au voisinage d'une entité. Ce graphe est calculé une seule fois pour l'ensemble des entités présentes dans l'environnement. Cette propriété permet de limiter la complexité du calcul, sans nuire à l'autonomie de l'entité qui pourra effectuer des requêtes de perception au travers de ce graphe.
- le **contrôleur de navigation réactive**, associé à chaque entité, qui permet de faire un suivi de trajectoire tout en respectant les règles explicitées ci-dessus.

Le contrôleur de navigation réactive se scinde en quatre modules, chacun en charge du respect de l'une des règles de navigation. Dans un premier temps, le module de suivi de trajectoire génère une vitesse permettant à l'entité de se diriger vers son but en suivant le chemin généré par la planification et en optimisant la trajectoire à la volée. Cette vitesse peut être considérée comme une vitesse théorique dans la mesure où elle ne prend pas en compte les éventuelles interactions avec les autres entités qui sont déléguées aux deux modules suivants. La vitesse théorique est donc passée au module de respect de l'espace personnel. Ce module calcule la distance entre l'entité et les entités voisines et modifie la vitesse fournie en entrée pour dévier la trajectoire en vue du respect de l'espace personnel. Le troisième module modifie cette vitesse, en fonction des règles sociales fournies par l'utilisateur, en vue de générer une trajectoire évitant les collisions avec les autres entités. Finalement, le quatrième module, le module de sécurité, assure le respect d'un certain temps minimum de réaction par rapport

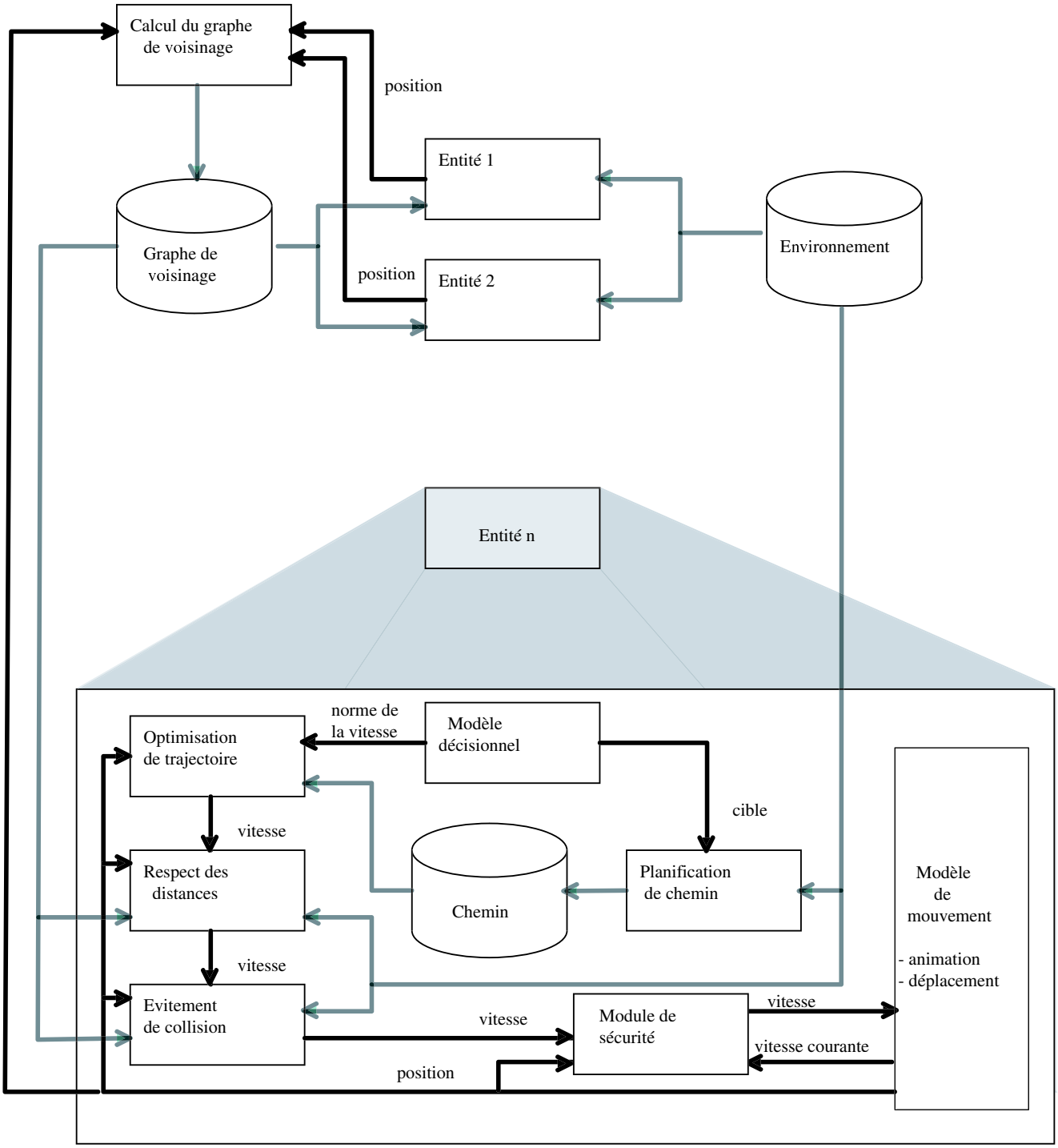


FIG. 2.1 – *Architecture utilisée pour la navigation des entités à l'intérieur d'un environnement peuplé.*

à la prochaine collision prévue en fonction de la vitesse courante, et non de la vitesse voulue, de l'entité. Cette notion est liée à des problèmes rencontrés lors du contrôle du déplacement d'un humanoïde virtuel. Le mouvement de marche de l'humanoïde [Men03] est un mélange de plusieurs méthodes de calcul : capture de mouvement et cinématique inverse pour la gestion des pieds d'appuis. Les moments propices à un ralentissement ou un changement de direction dépendent donc de ces calculs qui eux-mêmes dépendent du mouvement capturé, de la taille des jambes de l'humanoïde, de sa vitesse et de son mode de déplacement (course, marche)... La loi de commande associée à ce type de modèle est beaucoup trop complexe et variable pour pouvoir être modélisée correctement. Le rôle de ce module est donc d'assurer une certaine sécurité lors du déplacement en essayant, systématiquement, de prendre en compte un temps de réaction minimal lors du calcul d'une vitesse, tentant ainsi d'amortir les problèmes liés au contrôle de la marche[1].

L'ordre d'enchaînement des calculs des modules s'avère être très important. Il correspond à des niveaux de priorité liés aux contraintes. Il traduit notamment le fait que les évitements de collisions sont plus important que le respect de l'espace personnel eux mêmes plus importants que le respect de la trajectoire optimale.

Dans la suite de ce chapitre, nous allons nous attacher à décrire les différents algorithmes utilisés pour la gestion de la navigation réactive. Dans un premier temps, le mode de calcul ainsi que les propriétés du graphe de voisinage seront exposés. Puis, nous décrirons l'architecture des quatre modules liés au contrôleur de navigation réactive. Enfin, nous montrerons que malgré la richesse du modèle de navigation, il est possible d'animer des foules de plusieurs centaines de piétons en temps réel sur les ordinateurs actuels.

2.2 Graphe de voisinage

La navigation des entités s'appuie sur leur perception de l'environnement. Les informations perçues peuvent être classées dans deux catégories : les informations statiques et les informations dynamiques. Les informations statiques sont liées à la structure de l'environnement et représentent principalement sa géométrie et la manière dont celle-ci contraint la perception et la navigation. Dans la section 1.1, le modèle de subdivision spatiale a été présenté ainsi que ses bonnes propriétés en terme de navigation (cellules convexes et structure topologique) ainsi qu'en terme de filtrage pour la perception (lancer de rayon). Il permet donc de représenter efficacement les informations statiques. L'information dynamique pour sa part qualifie la position relative des objets en mouvement et pouvant entrer en interaction lors de la navigation. Cette information ne peut être pré-calculée dans la mesure où l'évolution spatiale des humanoïdes ne peut être prévue avant la simulation. Le rôle du graphe de voisinage est donc d'effectuer un calcul global, prenant en compte toutes les entités dynamiques, et permettant par la suite d'optimiser les requêtes de perception, autrement dit, de voisinage.

1. Nous ne disposons actuellement pas d'un modèle permettant de gérer les empruntes au sol. Ce type d'approche permettrait de résoudre ce problème beaucoup plus simplement.

2.2.1 Construction du graphe

Dans la mesure où le graphe de voisinage est une structure de données traduisant les positions relatives des objets dynamiques, il doit pouvoir être mis à jour rapidement en fonction du déplacement des entités. Sa complexité en terme de calcul doit donc être maîtrisée. D'autre part, si l'on ne fait aucune hypothèse sur le mode de perception des entités et principalement sur la distance de perception, la complexité d'accès aux informations de voisinage doit dépendre du nombre d'entités voisines mais pas de leur distance relative. Enfin, deux entités peuvent être considérées comme voisines si elles ne sont séparées par aucun obstacle.

En partant de ces constatations, nous définissons le graphe de voisinage par une triangulation de Delaunay filtrée par visibilité. La figure 2.2 montre un exemple de construction de ce graphe. Dans un premier temps, les positions spatiales des entités sont collectées et constituent l'ensemble des points à trianguler. Supposons que n entités peuplent l'environnement, la complexité de construction de la triangulation de Delaunay en deux dimensions est de l'ordre de $O(n \ln n)$, ce qui la rend exploitable pour un grand nombre d'entités. Dans un second temps, pour chaque segment de la triangulation, un lancer de rayon est effectué pour s'assurer de la visibilité entre les entités. Si le rayon entre en intersection avec une contrainte, le segment correspondant est supprimé de la triangulation. D'après les propriétés de la triangulation de Delaunay, le nombre de lancers de rayon est de $3n + 3$ [2], donc de complexité linéaire en fonction du nombre d'entités. Finalement, ce traitement génère un graphe où chaque sommet représente une entité et chaque arc, une relation de visibilité et de proximité (au sens de la triangulation de Delaunay) entre entités.

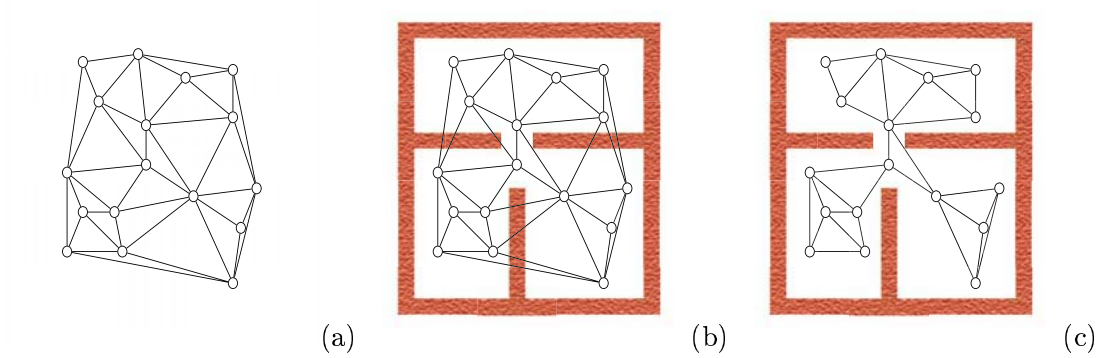


FIG. 2.2 – *Les différentes étapes associées à la construction du graphe de voisinage entre entités. La figure (a) montre la triangulation de Delaunay calculée en fonction de la position des entités représentées par un cercle. La figure (b) superpose la géométrie de l'environnement à la triangulation entre entités. Finalement, la figure (c) montre le filtrage par visibilité de la triangulation pour obtenir le graphe de voisinage.*

2. Nous considérons que la triangulation de Delaunay est construite en utilisant un triangle englobant, ajoutant 3 points qui ne représentent pas des entités.

2.2.2 Voisinage direct et propriété algorithmique

Outre ses bonnes propriétés en terme de complexité de calcul, la triangulation de Delaunay possède aussi de bonnes propriétés topologiques. En effet, elle assure que chaque point est relié à son plus proche voisin. Le graphe de voisinage hérite donc de cette propriété en ajoutant la notion de filtrage par visibilité.

Définissons le voisinage direct $V_d(E)$ d'une entité E comme étant l'ensemble des entités reliées à E par un arc dans le graphe de voisinage ; cet ensemble contient donc le plus proche voisin visible d'une entité. Cette propriété montre que pour la détection de collision par exemple, seules les entités appartenant à $V_d(E)$ sont à tester. D'autre part, d'après le mode de construction, ce voisinage direct s'adapte à la densité de la population, autrement dit, en environnement dense, il contient un ensemble d'entités proches de l'entité E, et en environnement peu dense, un ensemble d'entités éloignées. Par la suite, les algorithmes de respect de l'espace personnel ainsi que d'évitement de collision se baseront sur ces propriétés et ne travailleront que sur $V_d(E)$. De fait, ils adapteront automatiquement les distances de prévision à la densité de le population sans surcoût de calcul. Cependant, il est possible dans certaines configurations, que cette utilisation du voisinage direct implique un biais dans l'évitement de collision. En effet, des entités constituant des obstacles peuvent être détectées un peu tardivement (en fonction de leur distance relative et de la densité de la population), générant alors une réaction un peu brusque.

Plus généralement, l'utilisation du voisinage direct possède une conséquence algorithmique intéressante. Soit ε l'ensemble des entités présentes pour le calcul du graphe de voisinage. La triangulation possède $3 \times card(\varepsilon) + 3$ segments, ce qui constitue un majorant du nombre d'arcs présents dans le graphe de voisinage. L'existence d'un arc entre $e_1 \in \varepsilon$ et $e_2 \in \varepsilon$ dans le graphe de voisinage équivaut à $e_1 \in V_d(e_2)$ et $e_2 \in V_d(e_1)$ par construction. On peut donc en déduire la propriété suivante :

$$\sum_{e \in \varepsilon} card(V_d(e)) \leqslant 2 \times (3 \times card(\varepsilon) + 3) \tag{2.1}$$

Cette propriété permet de déduire que tout algorithme, travaillant au niveau d'une entité e, et effectuant des calculs sur la base de $V_d(e)$ en complexité linéaire, aura, globalement, au niveau de la simulation, un comportement qui au pire sera linéaire en fonction du nombre d'entités.

2.3 Suivi de chemin

La trajectoire empruntée lors de la navigation est optimisée grâce à la perception visuelle [RQ98]. Autrement, dit, l'homme lorsqu'il navigue a tendance à privilégier une trajectoire approximativement en ligne droite vers le prochain but qu'il perçoit. Le suivi d'un chemin consiste donc à constamment calculer une nouvelle direction à appliquer à l'entité permettant de générer une trajectoire cohérente par rapport au but fixé. Cependant, cette trajectoire, dans la mesure où l'entité n'est pas seule à naviguer dans son environnement risque d'être modifiée par les interactions avec les autres entités. Elle ne peut donc être totalement pré-calculée.

Dans notre modèle, le chemin à emprunter pour arriver à une cible fixée est planifié par l'algorithme de la section 1.2. Les chemins ainsi calculés peuvent être consultés sous deux formes, soit sous

la forme d'une liste de points contenant les points de passage utilisés lors de la planification, soit sous la forme d'une liste de segments, correspondant aux segments ayant servi à la génération des points de passage utilisés pour la planification. Cette deuxième forme de représentation du chemin est celle que nous utiliserons pour la définition de la trajectoire empruntée par le piéton. De cette manière, la génération de la trajectoire ne sera pas dépendante du nombre de points de passage présents dans l'environnement.

2.3.1 Optimisation par visibilité

Chaque segment présent dans le chemin généré par la phase de planification correspond à une sorte de portail à franchir pour arriver au but fixé. D'après les propriétés du mode de partitionnement, chacun des points définissant ce segment appartient à une contrainte de l'environnement. En conséquence, ce portail définit une zone de visibilité vers les portails suivants, la visibilité entre portails étant assurée par la propriété de convexité des cellules. L'algorithme va donc s'appuyer sur cette propriété pour calculer une optimisation de chemin à la volée, dépendant d'un cône de vision dont l'ouverture est conditionnée par le portail. Notons O la position de l'entité, (A,B) le prochain segment (ou portail) appartenant au chemin généré par la phase de planification. Le cône de vision $\mathcal{C}_O(A,B)$ est défini comme la zone délimitée par les deux demi droites OA et OB et possédant le plus petit angle d'ouverture (voir fig. 2.3). Tant que l'entité reste dans la cellule dont le portail est une frontière, la propriété de convexité des cellules assure que le triangle (O,A,B) est une zone de navigation libre de tout obstacle statique.

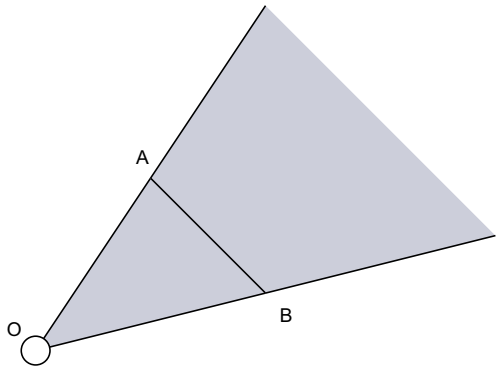


FIG. 2.3 – *Cône de vision associé à l'entité O au travers du portail (A,B).*

L'optimisation du chemin consiste à définir une suite de cônes de vision associés à chaque portail constituant le chemin. Leurs intersections sont alors calculées, selon l'ordre d'apparition dans le chemin, jusqu'à la limite de visibilité (qui se traduit par une intersection vide). La figure 2.4 présente un exemple où le chemin est décrit par la suite de portails $\{(A,B), (C,D), (E,F), ...\}$. Sur la figure 2.4(a), un premier cône $\mathcal{C}_O(A,B)$ est défini au travers du portail (A,B). Ce cône est ensuite intersecté avec le cône $\mathcal{C}_O(C,D)$; le résultat de cette intersection est représenté par la partie la plus grisée de la figure 2.4(b). Le même processus est répété avec le cône $\mathcal{C}_O(E,F)$. Finalement, la figure 2.4(c) montre le résultat de l'intersection de ces trois cônes, toujours représenté par la partie la plus grisée. L'algorithme s'arrête alors, car le prochain portail n'est pas visible.

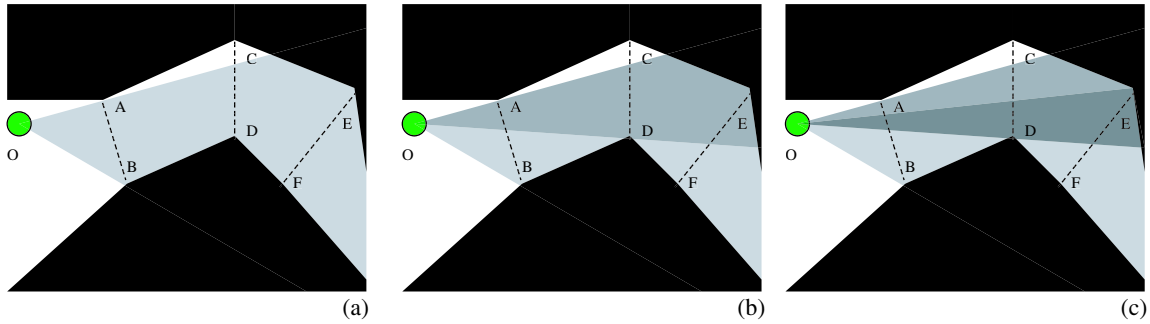


FIG. 2.4 – *Calcul d'intersection des cônes de visibilité jusqu'à la limite de visibilité. Les trois niveaux de gris représentent les trois intersections successives.*

Le processus s'arrête donc à la limite de visibilité des portails. Pour être en mesure d'optimiser un peu plus la trajectoire, en prenant en compte le prochain virage, un segment reliant le milieu du dernier portail au milieu du suivant (ou au point d'arrivée si l'on est en fin de chemin) est généré. Le cône de vision correspondant est intersecté avec le cône de vision courant. Si cette intersection est non vide, le cône nouvellement calculé devient le cône final, sinon le cône précédent est conservé. Ce cône définit une zone de navigation libre d'obstacle statique dans laquelle l'entité peut évoluer pour se diriger vers son but ; cette propriété est obtenue grâce à la convexité des cellules, dont les portails sont des frontières franchissables.

2.3.2 Interpolation entre cibles intermédiaires

Une fois le cône de vision calculé, il s'agit de calculer la nouvelle direction à appliquer à l'entité. Pour ce faire, une cible située sur la bissectrice du cône est calculée, à une certaine distance de l'entité (distance dépendant de la vitesse de l'entité ainsi que de la fréquence de calcul de cet algorithme). L'application directe de la direction de déplacement génère une trajectoire anguleuse. Ce qui s'avère peu réaliste [BJ03] en comparaison de la navigation piétonnière où une variation angulaire importante traduit le plus souvent un évitement de collision ou la décision de changer de chemin [RQ98].

Pour générer une trajectoire plus réaliste, nous proposons d'interpoler les cibles en utilisant une notion d'inertie. Pour ce faire, une information est stockée sur la dernière cible générée $ancienneCible$ et un calcul basé sur une inertie $inertie \in [0;1]$ et la nouvelle cible $nouvelleCible$ est effectué pour calculer la cible courante.

$$cibleCourante = inertie \times ancienneCible + (1.0 - inertie) \times nouvelleCible \qquad (2.2)$$

L'utilisation de cette inertie permet d'assouplir les virages provoqués par le changement de cible intermédiaire. La vitesse V_{out} fournie en sortie du module pour l'entité positionnée en P devant évoluer à une vitesse v est alors :

$$V_{out} = \frac{cibleCourante - P}{||cibleCourante - P||} \times v$$

Un exemple de trajectoire ainsi générée est présentée sur la figure 2.5 ; elle montre la trajectoire suivie par un piéton naviguant seul dans un quartier de Rennes.

FIG. 2.5 – *Un exemple de trajectoire suivie par un piéton dans le modèle 3d de la ville de Rennes.*

2.3.3 Prise en compte du déplacement

Le chemin calculé par l'algorithme de planification se présente sous la forme d'une suite de portails (ou segments) à franchir. Lorsque l'entité se déplace dans l'environnement en suivant les cibles intermédiaires générées, elle passe ces portails les uns après les autres. Cependant, la vitesse définie en sortie de l'algorithme n'est pas forcément la vitesse appliquée à l'entité du fait des éventuelles interactions avec les autres entités peuplant l'environnement. La convergence de la trajectoire réelle vers le but n'est donc pas toujours immédiatement assurée.

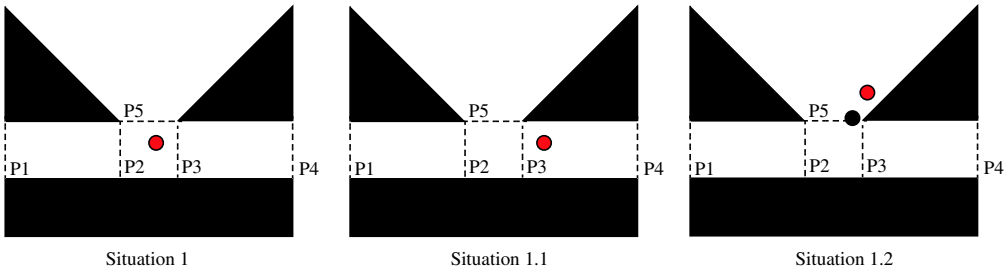


FIG. 2.6 – *Exemples de configurations pour un piéton suivant le chemin $P3$, $P4$.*

L'exemple de la figure 2.6, montre plusieurs configurations spatiales d'un piéton voulant suivre un chemin passant par les portails $P3$ et $P4$. Au départ, le piéton se trouve dans la situation 1. L'évolution normale est montrée dans la situation 1.1 où le piéton a franchi le portail $P3$. Le portail $P3$ doit donc être supprimé de la tête de liste des portails à franchir. La situation 1.2 montre une configuration dans laquelle le piéton a été « poussé », par un évitement de collision, dans une situation où il ne voit plus le portail $P3$. Ce cas est problématique, et nécessite de franchir, de nouveau, le portail $P5$ pour revenir à la configuration d'origine.

L'évolution de la liste des portails constituant le chemin s'effectue en prenant ces deux cas

en compte. Lorsqu'un portail est franchi, il est testé. S'il s'agit du portail de tête de chemin (le portail $P3$ dans l'exemple), ce portail est supprimé du chemin (dans l'exemple, il ne reste donc plus que le portail $P4$ à franchir). Dans le cas contraire, l'humanoïde risque de se trouver dans une situation similaire à la situation 1.2 de l'exemple. Le portail franchi est alors ajouté en tête du chemin, stipulant que l'humanoïde doit le franchir dans l'autre sens pour revenir à une situation normale. L'utilisation de cette technique, que l'on peut qualifier de chaînage arrière, autorise toutes les modifications de vitesse issues des évitements de collision. Elle assure le suivi du chemin, sans demande de nouvelle planification, économisant ainsi du temps de calcul.

2.4 Prise en compte des interactions

Les cibles intermédiaires générées par le modèle de suivi de trajectoire donnent une direction à suivre pour être en mesure d'atteindre un but fixé. La trajectoire ainsi générée, ne dépend que de la position de l'entité. Cependant, dès lors que l'entité n'est pas seule à naviguer dans l'environnement, il faut prendre en compte les éventuelles interactions. Ces interactions, comme nous l'avons décrit dans la section 2.1, sont de deux types: d'une part, le respect de l'espace personnel et, d'autre part, l'évitement de collision avec les piétons et les murs. Avant de présenter les algorithmes liés au traitement de ces contraintes, nous présentons le mode de représentation de l'entité.

2.4.1 Représentation de l'entité

Les calculs de collision entre des humanoïdes peuvent s'avérer très complexes et coûteux s'il faut prendre en compte toute leur géométrie. Pour simplifier ces calculs, nous allons nous appuyer sur la notion de cylindre englobant de l'humanoïde (voir fig. 2.7) et que nous réduisons à un cercle car la représentation en deux dimensions est l'hypothèse de départ du mode de subdivision spatiale.

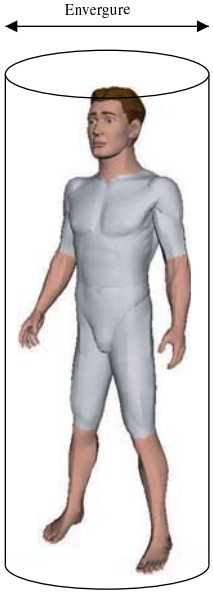


FIG. 2.7 – *Le cylindre englobant d'un humanoïde.*

Les informations associées à une entité et perceptibles par les autres pour le problème de la navigation sont les suivantes :

- **La position spatiale** : un vecteur en deux dimensions exprimé dans le repère du monde.
- **L'envergure de l'entité** : un scalaire représentant le diamètre du cercle représentant l'entité.
- **La vitesse courante** : un vecteur en deux dimensions indiquant la direction de déplacement et dont la norme fournit la vitesse.
- **Rayon de l'espace personnel** : il s'agit d'un scalaire représentant le rayon de l'espace personnel respecté par l'entité.

Généralement, un piéton est représenté par un ovale d'à peu près $50cm \times 60cm$ [Fru71]. Notre approximation par un cercle est due à une simplification des calculs. Lors des tests de collision, nous pouvons alors utiliser la distance entre deux humanoïdes, sans avoir à prendre en compte leur direction. D'autre part, en se référant aux proportions que nous venons de fournir, le cercle reste une approximation très raisonnable.

2.4.2 Respect de l'espace personnel

La contrainte de respect de l'espace personnel correspond à l'une des règles sociales à appliquer lors de la navigation. Si l'on reprend sa définition, il s'agit de respecter une certaine distance par rapport aux obstacle statiques et aux autres entités naviguant dans l'environnement ; cette distance définissant une zone de non intrusion. Potentiellement, seules les personnes faisant partie du cercle de connaissance sont admises à naviguer à l'intérieur. Le non respect de cette distance est lié à trois situations :

- la densité de population est trop importante pour pouvoir respecter cette contrainte,
- il s'agit d'une navigation en groupe, dans ce cas, les entités appartenant au groupe n'ont pas à respecter cette distance,
- l'entité navigue plus vite que ses entités voisines, temporairement, le temps de doubler, il est possible de ne plus respecter cette contrainte.

Comme on peut le remarquer, cette contrainte est souple. Elle définit principalement un comportement en milieu stationnaire comme, par exemple, une foule où les piétons naviguent à peu près dans la même direction. Finalement, le respect de cette contrainte peut être interprété comme une légère modification de la direction de déplacement de l'entité mais sans modification de vitesse. Le modèle proposé consiste donc à effectuer des calculs sur le voisinage direct de l'entité (voir section 2.2.2), en calculant une modification de direction dépendante de la proximité des voisins lors de la navigation. Le modèle, proche d'un modèle à particules, calcule des forces de répulsion à appliquer sur le vecteur vitesse de l'entité pour modifier sa direction en vue du respect de l'espace personnel.

Notons d_p la distance à respecter entre entités voisines. Définissons dans un premier temps, une fonction de $f : [0; d_p] \rightarrow [0; 1]$ telle que $f(0) = 1$ et $f(d_p) = 0$:

$$f(x) = \frac{-x + d_p}{d_p} \tag{2.3}$$

Cette fonction calcule un facteur entre 0 et 1 permettant de pondérer la prise en compte d'une force de répulsion en fonction de la distance. Elle est utilisée pour le calcul de la force de répulsion normée à appliquer sur le vecteur vitesse de l'entité. Soit p_1 et p_2 les positions spatiales de deux

entités, d'envergure respective $2 \times r_1$ et $2 \times r_2$, telles que $||p_1 - p_2|| < d_p + r_1 + r_2$, autrement dit telles que les deux entités entrent en interaction pour le respect de l'espace personnel. La fonction $R_E : \mathbb{R}^2 \times \mathbb{R}^2 \rightarrow \mathbb{R}^2$ suivante calcule la force de répulsion associée à l'entité dont p_1 est la position:

$$R_E(p_1,p_2) = \frac{f(||p_2 - p_1|| - r_1 - r_2)}{||p_2 - p_1|| - r_1 - r_2}(p_1 - p_2) \tag{2.4}$$

Définissons $V_I(p_1)$, l'ensemble des positions des entités du voisinage direct $V_d(p_1)$ de l'entité en position p_1 et entrant en interaction pour le respect de l'espace personnel. Cet ensemble est défini comme suit :

$$V_I(p_1) = \{x \mid x \in V_d(p_1) \wedge ||x - p_1|| < d_p + r_1 + r_2\} \tag{2.5}$$

D'autre part, définissons $V_M(p_1)$, l'ensemble des projections de la position p_1 de l'entité sur les frontières contraintes de la cellule dans laquelle elle se trouve. L'ensemble $V_{Im}(p_1)$ suivant prend en compte l'ensemble des distances aux murs provoquant une répulsion :

$$V_{Im}(p_1) = \{x \mid x \in V_M(p_1) \wedge ||x - p_1|| < d_p + r_1 + r_2\}$$

Pour une raison de simplification des notations, nous utilisons le même rayon d'espace personnel pour le respect de la distance avec les autres piétons et les murs ; il est cependant possible de les différencier, mais dans le cadre de nos simulations, le besoin ne s'en est pas fait sentir. L'ensemble des forces de répulsion $R(p_1)$ à prendre en compte pour la modification du vecteur vitesse de l'entité en p_1 est donc défini comme suit :

$$R(p_1) = \sum_{p_2 \in V_I(p_1) \cup V_{Im}(p_1)} R_E(p_1,p_2) \tag{2.6}$$

Soit v la vitesse actuelle de l'entité en p_1, soit α un coefficient permettant de pondérer la prise en compte du vecteur de répulsion calculé, la nouvelle vitesse à appliquer à l'entité est calculée comme suit :

$$\frac{||v||}{||v + \alpha R(p_1)||}(v + \alpha R(p_1)) \tag{2.7}$$

Cette formule n'est valable que dans le cas $||v + \alpha R(p_1)|| \neq 0$, dans le cas contraire, la vitesse n'est pas modifiée. Comme le montre cette formule, la prise en compte du voisinage ne modifie que la direction du vecteur vitesse, mais pas sa norme. Il est important de noter que cet algorithme n'a pas pour but de produire un évitement de collision, dans cette mesure la conservation de la norme de la vitesse est nécessaire. Il ne fait que provoquer une modification de la direction en vue de respecter une certaine distance avec les autres entités durant la navigation. La vitesse produite, nécessite donc d'être à nouveau filtrée pour assurer une navigation propre et sans collision.

2.4.3 Navigation et évitement de collision

Les deux modèles précédents ont traité le problème de la génération de trajectoire pour le suivi de chemin ainsi que la prise en compte dans la vitesse générée pour l'entité du respect de l'espace personnel. L'utilisation de ces algorithmes cependant, ne prend pas encore en compte la contrainte la plus forte de la navigation qu'est l'évitement de collision. L'évitement de collision est une partie relativement codifiée du comportement de navigation. De façon générale, deux grands types de comportements sont utilisés : les comportements agissant sur la vitesse (ralentir, accélérer) et les comportements agissant sur la direction (éviter à droite ou à gauche).

L'observation du comportement piétonnier montre par ailleurs une certaine dépendance entre les réactions à l'évitement de collision et le code de la route en vigueur dans le pays [LW92]. Si l'on étudie le cas de deux flots de piétons, arrivant face à face, en France la tendance sera à la création de deux flots naviguant chacun en "serrant" la droite de la route. En Angleterre, le phénomène inverse sera constaté, les deux flots auront tendance à serrer la gauche de la route. Pour être en mesure de traduire facilement ce type de règles, et pourquoi pas simuler un Français se promenant dans Londres, une architecture générale est proposée. Le principe de fonctionnement est le suivant. D'une part, un certain nombre de modules sont définis pour réagir aux collisions (éviter à gauche, éviter à droite, ralentir, accélérer). Ces modules sont abonnés à des types de collision, l'organisation des modules ainsi que leur disposition permettra ainsi de définir le comportement de navigation. Ensuite, en fonction d'une vitesse appliquée à l'entité, un algorithme fait une prédiction de collision et type cette collision. Une fois cette collision typée, les modules permettant de réagir à ce type de collision calculent une nouvelle vitesse et notent la qualité de leur solution. Enfin, un dernier algorithme itère sur les solutions calculées de manière à trouver une combinaison de modules maximisant la qualité de la réaction et ne provoquant plus de collision.

2.4.3.1 Prédiction de collision

Le modèle de prédiction de collision se base sur une extrapolation linéaire de la position des entités, en fonction de leur position et de leur vitesse à un instant donné. Ce modèle est simple, mais il est rapide à calculer et ne nécessite pas de stockage des dernières positions des entités voisines, contrairement à une extrapolation de degré 2 ou plus.

Une entité est représentée par un cercle centré sur sa position, d'un diamètre égal à son envergure. En fonction de la vitesse courante et de la position de deux entités, il est possible de déterminer un polynôme de collision traduisant l'évolution temporelle de la distance les séparant. Notons p_1,p_2 les positions, v_1,v_2 les vitesses et r_1,r_2 les rayons des deux entités et t le temps. Les positions respectives des entités en fonction du temps se traduisent par les formules suivantes :

$$\begin{array}{rcl} P_1(t) & = & p_1 + v_1 t \\ P_2(t) & = & p_2 + v_2 t \end{array} \tag{2.8}$$

Les deux entités peuvent entrer en collision si, à un instant t donné, leur distance relative est inférieure ou égale à la somme du rayon de leurs envergures. Cette situation est caractérisée par le vérité de l'inéquation $||P_1(t) - P_2(t)|| \leq r_1 + r_2$. Le moment de la collision peut donc être calculé par l'intermédiaire du polynôme suivant :

$$\begin{array}{rcl} C(t) & = & (P_1(t) - P_2(t))^2 - (r_1 + r_2)^2 \\ & = & (v_1 - v_2)^2 t^2 + 2 \times (p_1 - p_2)(v_1 - v_2)t + (p_1 - p_2)^2 - (r_1 + r_2)^2 \end{array} \tag{2.9}$$

Deux situations sont à prendre en compte:

- $C(0) \leq 0$: dans ce cas, une collision est en cours. L'algorithme lève donc une exception redonnant la main à l'utilisateur. Cette solution permet notamment de générer un comportement particulier tel que la présentation d'excuses par exemple.
- $C(0) > 0 \wedge C'(0) < 0$: dans ce cas, une collision va se produire dans le futur. Notons t_c la solution de l'équation $C(t) = 0$ minimisant les deux solutions. Les entités entrent donc en collision au temps t_c.

Dans le cas d'une collision future, il est possible de typer cette collision en fonction de la position relative des deux entités au moment t_c de leur collision. Notons $c_1 = p_1 + v_1 t_c$ et $c_2 = p_2 + v_2 t_c$ les positions des entités au moment de leur collision. Quatre types de configurations (voir fig. 2.8) sont identifiés:

1. **Collision dans le dos**: la seconde entité va donc percuter l'entité considérée par l'arrière. Cette situation est caractérisée par la vérité de l'assertion suivante: $(c_2 - c_1).v_1 < 0$.
2. **Collision de face**: les deux entités arrivent face à face. Cette situation est caractérisée par l'assertion suivante: $(c_2 - c_1).v_1 > 0 \wedge (c_2 - c_1).v_2 < 0$.
3. **Collision arrière**: l'entité considérée va percuter l'autre entité par l'arrière. Cette situation est caractérisée par l'assertion suivante: $(c_2 - c_1).v_1 > 0 \wedge (c_2 - c_1).v_2 > 0$.
4. **Collision statique**: l'entité considérée va entrer en collision avec une entité statique. Cette situation est caractérisée par l'assertion suivante: $||v_2|| = 0$.

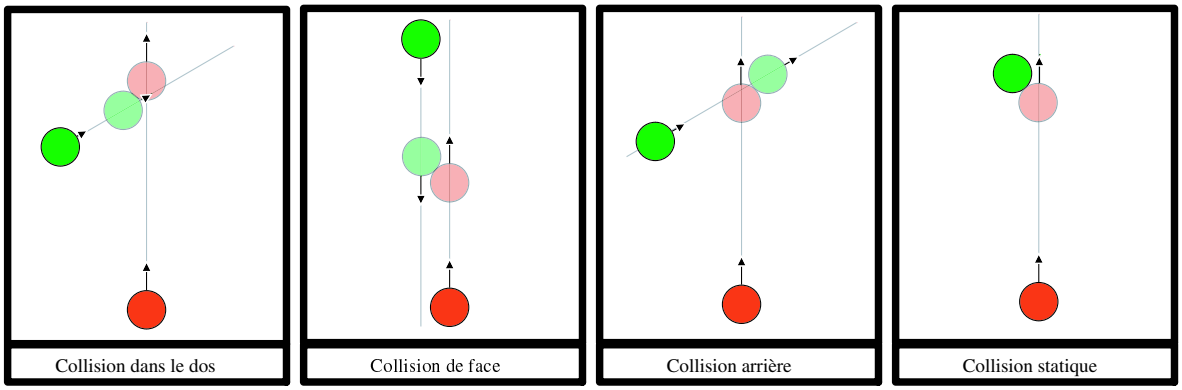


FIG. 2.8 – *Les quatre types de collisions qualifiés par l'algorithme de prédiction. L'entité dont on cherche à caractériser la collision est représentée par un cercle en bas de figure.*

Le besoin de différenciation de ces types de collisions est issu des recherches sur le comportement de navigation humain qui montrent qu'en fonction des cas de collisions, les typologies de réaction ne sont pas exactement les mêmes (minimisation de l'angle d'évitement[Gof71], application du code de la route [LW92]).

2.4.3.2 Modules de réaction aux collisions

Les réactions aux collisions peuvent être classées dans deux grandes catégories de comportement : d'une part, les comportements agissant sur la vitesse de l'entité (s'arrêter, ralentir, accélérer), d'autre part, les comportements agissant sur la direction (éviter à gauche, éviter à droite). L'ensemble des ces comportements forme une base permettant d'exprimer tout type de réaction aux collisions. Cependant, en fonction du contexte, notamment du type de collision, leur mélange varie.

Pour prendre en compte ces constatations, la notion de module de réaction aux collisions est proposée. Ce module décrit une règle d'évitement de collision tout en renvoyant une information

sur la qualité de sa solution. Notons E_1 et E_2 deux entités, E_1 est l'entité dont on veut modifier la vitesse pour éviter la collision avec E_2. Un module est donc une boite noire prenant en entrée :

- les positions actuelles de E_1 et E_2,
- la vitesse perçue de E_2,
- la vitesse proposée pour E_1 et provoquant une collision,
- le type de la collision prédite.

Il est à noter que l'utilisation de la vitesse perçue plutôt que de la vitesse voulue de la seconde entité rend l'algorithme plus robuste au cas des entités possédant une inertie, ce qui est le cas de l'être humain. Les sorties de ces modules sont les suivantes :

- la nouvelle vitesse à appliquer à l'entité E_1,
- la qualité de la réaction calculée, il s'agit d'un scalaire dans l'intervalle $[0; 1[$. L'intervalle exclut la valeur 1 pour des raisons de convergence de la multiplication des qualités, qui traduit la qualité de leur combinaison.

En plus de ces informations de sortie, un module de collision peut être en mesure de lever une exception. Cette exception lui permet de stipuler que dans la configuration courante, il n'est pas en mesure de fournir une réponse correspondant à son rôle. Ce peut être le cas pour un module de ralentissement, par exemple, s'il n'est pas possible de ralentir suffisamment pour être en mesure d'éviter une collision potentielle.

Enfin, un dernier module, un peu particulier, est fourni par le système. Ce module, que nous appellerons module d'acceptation, permet de valider automatiquement une vitesse provoquant une collision. Ce module peut être utilisé pour spécifier, par exemple, que dans le cas d'une collision dans le dos, l'entité considère qu'il est du ressort de l'autre entité d'éviter cette collision.

Par la suite, l'utilisateur est à même de fournir un ensemble de modules, chacun proposant une solution permettant de résoudre un problème de collision particulier. La notion de qualité associée à la sortie d'un module devient alors utile pour la recherche d'une solution optimale permettant de combiner plusieurs réactions en fonction des différentes collisions qui peuvent être détectées.

2.4.3.3 Navigation par recherche de solution

Lorsque l'on dispose d'un certain nombre de modules de réaction aux collisions, il est possible de définir le comportement de navigation d'une entité. La définition de ce comportement se fait par abonnement de modules à des types de collision. La table de la figure 2.9 montre un exemple de configuration. Une fois cette configuration effectuée, un algorithme recherche une combinaison optimale, en terme de qualité et de prédiction temporelle, générant une nouvelle vitesse assurant l'absence de collision.

Chaque module étant spécialisé pour l'évitement d'une seule entité, la vitesse qui est fournie en sortie peut elle-même provoquer une collision avec une autre entité voisine. Autrement dit, dans un grand nombre de cas, il va être nécessaire de combiner plusieurs modules de réaction pour trouver une vitesse optimale. Cependant il n'est pas *toujours* possible de trouver une vitesse évitant *toutes* les collisions. Pour s'en convaincre, il suffit de se représenter un piéton entouré par d'autres piétons convergeant tous vers lui, sans laisser d'espace entre eux. Ce cas ne possède pas de solution. Pour assurer la convergence de l'algorithme, à chaque calcul d'une nouvelle collision, nous allons utiliser le temps de prédiction associé à la collision précédente comme borne temporelle. Autrement dit, si

réaction \ collision	dans le dos	de face	arrière	statique
évitement droite		X	X	X
évitement gauche			X	X
ralentissement		X	X	
accélération		X	X	
acceptation	X			

FIG. 2.9 – *Un exemple schématique de configuration du module d'évitement de collision pour un piéton naviguant dans un pays privilégiant l'évitement à droite.*

une nouvelle collision est détectée dans $10s$ alors que la première collision a été détectée dans $5s$, la seconde collision ne sera pas prise en compte en remettant à plus tard le calcul de réaction. La solution optimale calculée par l'algorithme sera donc celle maximisant la qualité des réactions qui ont contribué à son calcul, mais sera aussi celle qui maximise le temps durant lequel aucune collision n'est prise en compte ou détectée. L'algorithme se déroule donc comme suit :

1. L'algorithme nécessite l'utilisation d'une structure de données contenant des triplets de la forme (q,v,t), constamment triée dans l'ordre décroissant de $q \times t$. Ce triplet représente une réaction à une collision dans laquelle q représente la qualité de la réaction, v la vitesse proposée et t la date de la collision que cette solution permet d'éviter. Le fait de trier cette structure de données suivant $q \times t$ favorise les propositions maximisant la qualité et la distance temporelle de la collision. Cette liste, que nous appellerons liste de traitement, est initialisée avec le triplet $(1, v_{in}, t_{max})$ dans lequel 1 représente la qualité maximale, v_{in} la vitesse voulue en entrée du module et enfin t_{max} le temps maximum de prédiction de collision.
2. Si la liste de traitement est vide, aucune solution n'a été trouvée. Dans ce cas, la vitesse appliquée devient la vitesse nulle, laissant ainsi la possibilité aux autres entités de réagir plus facilement, et l'algorithme se termine. Sinon, le triplet (q, v, t) est extrait de la tête de la liste de traitement triée car il constitue la solution la plus prometteuse.
3. Un calcul de prédiction de collision est effectué en simulant l'application de la vitesse v à l'entité. Deux cas peuvent se présenter en fonction de l'existence ou de la prise en compte d'une collision :
   (a) une collision est détectée au temps t_c mais $t_c > t$ ou bien aucune collision n'est détectée. Dans le premier cas, la collision a lieu après la première collision détectée, elle pourra donc être prise en compte plus tard et n'a pas besoin d'être traitée immédiatement. Dans les deux cas, aucune collision n'est prise en compte, la proposition choisie est donc celle qui maximise à la fois la qualité et le temps prévu sans collision. Cette solution est alors élue par l'algorithme, qui se termine en renvoyant la vitesse v.
   (b) Une collision est détectée et $t_c < t$. La collision est alors typée (collision de face, dans le dos...) et les modules abonnés à ce type de collision sont consultés. Si l'un des modules est un module d'acceptation, la solution est élue et l'algorithme se termine en renvoyant la vitesse v. Dans le cas contraire, pour chaque module, le couple de solution (q_m, v_m) est calculé. Pour chacun de ces couples, le triplet $(q \times q_m, v_m, t_c)$ est ajouté dans la liste de traitement, à un rang correspondant à $q \times q_m \times t_c$. Si un module génère une exception, sa solution n'est pas calculée et n'est donc pas ajoutée à la liste de traitement.

Cet algorithme permet de travailler sur toutes les formes de navigation réactive, sans hypothèse de réaction aux collisions. Le comportement de navigation exhibé par le piéton est alors dépendant de la configuration des modules et de la définition de la qualité de leur solution. Cette propriété permet de ne pas fournir le même comportement à tous les piétons et autorise, en temps réel, des modifications de configuration. Enfin, il est construit sur le schéma d'un algorithme A^* qui lui confère donc ses bonnes propriétés.

Les calculs de prédiction de collisions s'effectuent en exploitant le voisinage direct d'une entité (voir section 2.2.2), ce qui permet de grandement limiter le nombre de voisins à prendre en compte. Dans la mesure où ce voisinage contient l'entité la plus proche, son utilisation assure que toutes les collisions pourront être détectées et prédites. Cependant, il est possible que, dans certain cas, l'exploitation de cette propriété génère une prédiction un peu tardive qui implique une réaction rapide. Il s'agit de la conséquence du compromis que nous avons effectué entre la qualité des évitements de collisions et la rapidité de calcul. D'autre part, la distance des entités appartenant au voisinage direct est dépendante de la densité de la population. En environnement dense, la prédiction de collision se fera donc à courte distance alors que dans une foule dispersée, elle se fera à grande distance. Cette propriété permet de dire, que l'algorithme de détection de collision adapte automatiquement ses réactions à la densité de la foule dans laquelle l'entité évolue, sans aucun surcoût de calcul.

2.4.3.4 Module de sécurité.

Le module de sécurité permet de filtrer la vitesse v_{in} calculée lors des traitements précédents, en fonction de la prochaine collision détectée avec les murs ou avec les entités en prenant v_r, la vitesse réelle de l'entité (au pas de temps précédent), comme référence. L'utilisation de la vitesse courante, plutôt que de la vitesse voulue, permet de prendre en compte des phénomènes de retard (inertie, changement de pied d'appui...) sur l'application des commandes. Le module est configuré avec un temps de réaction, que nous noterons t_{react}, qui correspond à une distance temporelle à respecter avec la prochaine collision détectée. Le rôle du module est alors de diminuer la norme de la vitesse voulue (v_{in}) en fonction du moment de la prochaine collision détectée en appliquant la vitesse v_r à l'entité. Les collisions prises en compte, sont de deux types : les collisions avec les murs et les collisions avec les entités.

Collision du piéton avec les murs. Supposons que nous voulions tester la collision entre une entité positionnée en p, évoluant à une vitesse v, d'une envergure de $2 \times r$ et un segment (A,B) représentant le bord d'un obstacle. Le moment de la collision avec ce segment, possédant une normale n telle que $n.(p-A) \geq 0$, peut être calculé en exprimant la distance entre l'entité et la droite (A,B) en fonction du temps :

$$d(t) = (p + vt - A).n = (p - A).n + v.n \times t$$

Le moment de la collision éventuelle est alors fourni par l'équation $d(t) = r$, qui a pour solution $t_c = \frac{r-(p-A).n}{v.n}$. Posons $p_c = p + v \times t_c$ la position de l'entité lors de la collision. Il y a une collision entre l'entité et le segment si $(||p_c - A|| < r) \vee (||p_c - B|| < r) \vee ((p_c - A).(p_c - B) < 0)$. Pour détecter la prochaine collision avec un mur, nous utilisons l'algorithme de lancer de rayon décrit dans la section 1.1.3 en remplaçant le test d'intersection avec un mur par le test qui vient d'être décrit.

Conservation du temps de réaction. Notons t_m le moment de la prochaine collision détectée avec un mur en appliquant la vitesse courante v_r de l'entité pour la prédiction. De même, notons t_e le temps de la prochaine collision avec une autre entité de l'environnement, toujours en appliquant la vitesse réelle v_r de l'entité. Notons $t_c = min(t_m,t_e)$, la date de la collision la plus proche dans le temps. Dans le cas où $t_c \geq t_{react}$, la vitesse v_{in} n'est pas modifiée et le module la renvoie comme solution. Dans le cas où $t_c < t_{react}$ la vitesse fournie en sortie du module est :

$$v_{in} \times \frac{t_c}{t_{react}}$$

Autrement dit, la norme de la vitesse v_{in} fournie en entrée est réduite de manière à tenter de respecter la distance temporelle t_{react} avec la prochaine collision, en ne prenant en compte que la vitesse courante. Ce calcul considère donc que la vitesse voulue ne sera pas appliquée immédiatement et permet de compenser le phénomène de retard sur les commandes. Le temps de réaction t_{react} suffisant pour assurer l'absence de collision doit être configuré manuellement, cependant, plus ce temps est élevé, plus l'algorithme est robuste.

Les différents modules qui viennent d'être décrits constituent une chaîne de traitement essayant de faire un compromis entre la vitesse idéale permettant de suivre une trajectoire planifiée et les interactions provoquées par la présence d'autres piétons dans l'environnement. L'ajout du module de sécurité permet de réguler la vitesse, pour éviter les collisions qui peuvent se produire si un retard est présent dans la loi de commande, qui nous est inconnue. Il est cependant à noter, que dans le cadre de l'animation comportementale, contrairement au cas de la robotique, des situations de collisions sont acceptables (d'ailleurs il en existe dans la réalité) dans la mesure où elles ne sont pas trop fréquentes. D'autre part, dans le cas de grandes foules avec une forte densité de personnes, comme le souligne Helbing [HFV00], les personnes entrent en contact corporel, ce qui peut être assimilé à une collision à très faible vitesse. La partie configurable de l'algorithme de recherche de vitesse optimale, permet de traduire des comportements de navigation complexes et réalistes, si les modules sont bien configurés, tout en limitant le coût de calcul. Cependant, comme un grand nombre de modèles de navigation réactive, il est soumis au problème que l'on peut qualifier des minima locaux. Un piéton peut se trouver bloqué sans trouver de solution de sortie, particulièrement lorsqu'il rencontre un groupe de piétons statiques très proches d'un mur.

2.5 Modèle de piéton virtuel

En vue de traduire les règles de navigation inspirées du comportement de l'être humain, nous définissons quatre modules simples ayant pour rôle de fournir une vitesse évitant une collision en : évitant à droite, évitant à gauche, ralentissant ou accélérant. Les fonctions de qualité qui vont leur être associées vont alors pouvoir traduire une qualité en fonction d'une différence angulaire ou bien d'une différence de vitesse. Nous allons définir ces modules puis présenter quelques configurations permettant de traduire différents comportements de navigation.

2.5.1 Définition des modules

Deux types de modules sont utilisés pour traduire un comportement de navigation. D'une part les modules agissant sur la vitesse de l'entité, d'autre part les modules agissant sur la direction.

Notations : Pour la suite de l'explication, nous allons considérer deux entités 1 et 2. L'entité 1 est l'entité dont la vitesse souhaitée provoque une collision avec l'entité 2. Les calculs vont donc tenter de trouver une nouvelle vitesse à appliquer à l'entité 1 de manière à éviter la collision avec l'entité 2. Durant l'explication, les formules utiliseront les notations suivantes :

- p_1, p_2 sont respectivement les positions de l'entité 1 et de l'entité 2.
- i_1, i_2 sont les rayons des cercles représentant les deux entités.
- v_2 est la vitesse perçue de l'entité 2.
- v_{in} est la vitesse en entrée du module, autrement dit, la vitesse que l'entité 1 voudrait appliquer mais qui provoque une collision.
- v_{out} est la vitesse en sortie du module, autrement dit, une vitesse que l'entité devrait appliquer pour éviter la collision détectée.
- $R(\theta)$ est une matrice de rotation 2×2 d'un angle θ.

2.5.1.1 Position relative et direction d'évitement

Pour être en mesure de trouver des solutions d'évitement de collision, nous allons travailler avec les positions et vitesses relatives des entités. Les positions des entités en fonction du temps sont données par les formules suivantes :

$$\begin{aligned} P_1(t) &= p_1 + v_{in}t \\ P_2(t) &= p_2 + v_2 t \end{aligned}$$

Exprimons l'évolution de la position de l'entité 1 relativement à l'entité 2, notée $P_r(t)$:

$$P_r(t) = P_1(t) - P_2(t) = p_1 - p_2 + (v_{in} - v_2)t$$

Posons $P = p_1 - p_2$ et $V = v_{in} - v_2$, P et V correspondent respectivement à la position et à la vitesse relatives de l'entité 1 par rapport à l'entité 2. Par ce changement de repère, la position de l'entité 2 est (0,0) et celle de l'entité 1 est $P_r(t) = P + Vt$. Puisque les entités risquent d'entrer en collision, la distance entre le point (0,0) et la droite décrite par $P_r(t)$ est inférieure à $i_1 + i_2$, la somme des rayons des cercles représentant les entités. Éviter la collision consiste donc à trouver une nouvelle vitesse relative définissant une trajectoire passant à une distance de (0,0) supérieure à $i_1 + i_2$ (voir fig. 2.10). Cette résolution s'appuie donc sur le calcul des tangentes au cercle de centre (0,0) et de rayon $I = i_1 + i_2 + \varepsilon$ passant par P, où ε représente une distance supplémentaire de sécurité lors de l'évitement. En posant $X = \frac{P}{||P||}$ et $Y = X \times R(\frac{\pi}{2})$, les points T_g et T_d, soutenant les tangentes au cercle passant par P, sont définis comme suit :

$$\begin{aligned} T_d &= \frac{I^2}{||P||}X + (I^2 - \frac{I^4}{P^2})^{\frac{1}{2}}Y \\ T_g &= \frac{I^2}{||P||}X - (I^2 - \frac{I^4}{P^2})^{\frac{1}{2}}Y \end{aligned}$$

Les directions $D_d = T_d - P$ et $D_g = T_g - P$ définissent respectivement une direction d'évitement à droite et à gauche. Donc toute vitesse relative colinéaire à $D = D_d$ ou $D = D_g$ permet d'éviter la collision en passant à une distance ε de l'entité 2. Cette contrainte peut s'écrire sous deux formes en considérant que $D \in \{D_d, D_g\}$:

ct$_1$: $\alpha D = v_{out} - v_2$ avec $\alpha > 0$. Cette contrainte exprime que la vitesse relative obtenue doit être colinéaire à D et qu'elle doit conserver la même direction que D.

ct$_2$: $D \times R(\frac{\pi}{2}).(v_{out} - v_2) = 0 \wedge D.(v_{out} - v_2) \geq 0$. Cette contrainte exprime la colinéarité sous la forme d'un produit scalaire de valeur nulle et la seconde partie assure la conservation de la direction de la vitesse relative.

Créer un module d'évitement de collision consiste donc à trouver une vitesse v_{out} à appliquer à l'entité 1, respectant la contrainte de colinéarité et de direction de la vitesse relative, exprimées par la contrainte ct$_1$ ou ct$_2$.

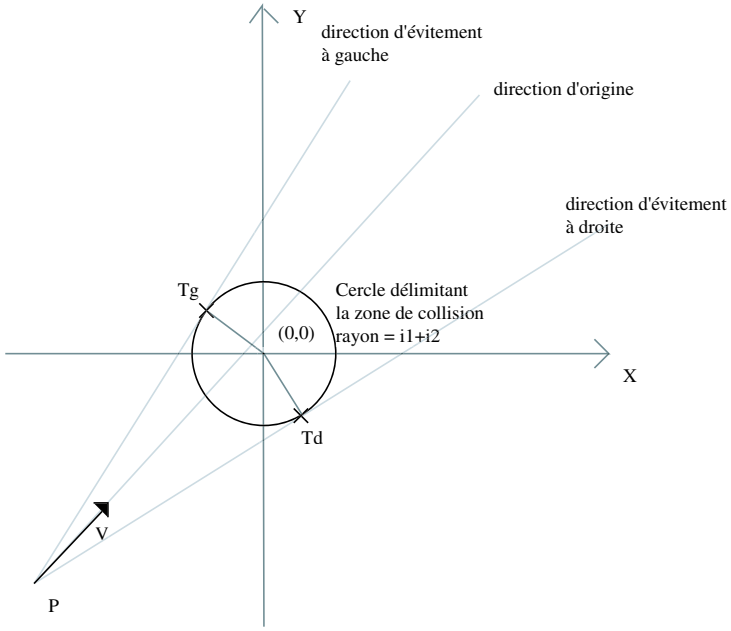


FIG. 2.10 – *Méthode de calcul de l'évitement à gauche et à droite en position et vitesse relative.*

2.5.1.2 Modules d'évitement à gauche et à droite

Les modules d'évitement de collision par la gauche et la droite n'agissent que sur la direction de la vitesse de l'entité mais conservent sa norme. Le calcul de la vitesse v_{out} fournie en sortie de ces modules s'appuie sur l'expression de la contrainte ct$_2$ qui vient d'être présentée. En prenant $D = D_d$ pour un évitement à droite et $D = D_g$ pour un évitement à gauche, v_{out} est solution de l'équation suivante :

$$v_{out} = \alpha D + v_2$$

Le problème se ramène donc à la recherche de α tel que $||v_{in}|| = ||v_{out}||$, ce qui s'écrit sous la forme suivante :

$$||v_{in}||^2 = ||\alpha D + v_2||^2$$

En développant cette équation, α peut être calculé par l'intermédiaire des solutions du polynôme suivant :

$$\alpha^2 D^2 + 2\alpha D.v_2 + v_2^2 - v_{in}^2 = 0$$

Il faut donc prendre la solution positive de ce polynôme, pour conserver la direction de déplacement, permettant de calculer la nouvelle vitesse $v_{out} = \alpha v_e + v_2$. Si une telle solution n'existe pas, le module lève une exception stipulant ainsi qu'aucune solution n'a été trouvée.

Définition de la qualité. La fonction de qualité utilisée a pour but de corréler la qualité associée à cette réaction au cosinus de l'angle entre les deux vecteurs vitesse v_{in} et v_{out}. De cette manière, les modules minimisant la déviation pour l'évitement pourront être favorisés. Elle est définie comme suit :

$$c(v_{in},v_{out},\beta) = ((1 + \frac{v_{in}.v_{out}}{||v_{in}|| \times ||v_{out}||}) * 0.5)^{\beta} \times 2 - 1.0$$

L'exposant β ajouté permet de modifier la forme de la courbe associée à la fonction. Plus cet exposant est grand plus la qualité décroît en fonction de l'angle entre v_{in} et v_{out}. La figure 2.11 montre différentes formes de courbes associées à la modification du paramètre β. La bonne définition de cette fonction, va permettre de favoriser certains types de réactions en fonction du type de la collision à laquelle le module sera abonné. L'utilisation de cette fonction assure un résultat dans $[0;1[$ car une vitesse colinéraire à la vitesse en entrée ne permet pas d'éviter une collision et car l'angle entre les deux vecteurs v_{in} et v_{out} est dans l'intervalle $[-\frac{\pi}{2};\frac{\pi}{2}]$ par définition du module.

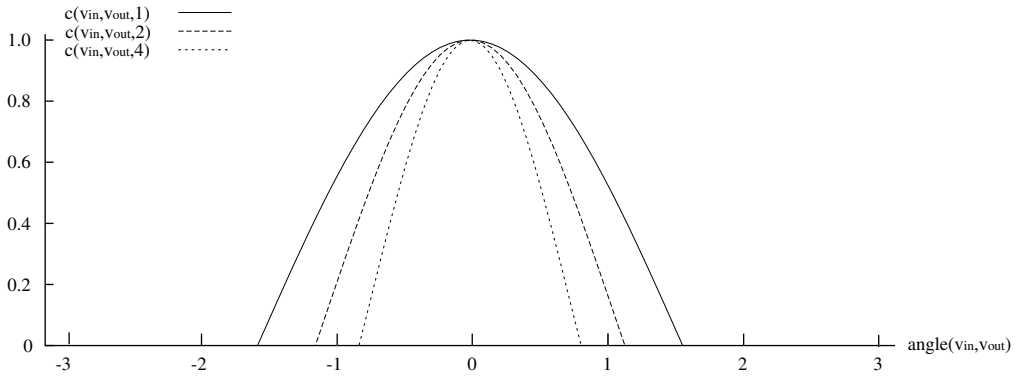


FIG. 2.11 – *Différentes forme de la fonction de qualité* $c(v_{in},v_{out},\beta)$.

2.5.1.3 Modules de modification de la vitesse.

Ces deux modules ont pour rôle de modifier la norme de la vitesse v_{in} fournie en entrée de manière à générer une vitesse v_{out} conservant la même direction et ayant une norme inférieure dans le cas d'une décélération et supérieure dans le cas d'une accélération. Il est à noter que ces modules ne prennent pas en compte la loi d'accélération ou de décélération associée à l'entité simulée. Ils ont pour rôle de calculer une vitesse instantanée idéale à appliquer. Modifier la norme de la vitesse en vue d'éviter la collision se résume à trouver un facteur α tel que :

$$v_{out} = \alpha v_{in}$$

Dans le cas de $\alpha > 1$, nous aurons $||v_{out}|| > ||v_{in}||$, l'entité devra donc accélérer. Dans le cas $0 \leq \alpha < 1$, nous aurons $||v_{out}|| > ||v_{in}||$, l'entité devra donc décélérer. Pour trouver la valeur de α, l'expression de la contrainte ct_2 est utilisée. Il faut donc trouver α tel que pour $D \in \{D_g, D_d\}$:

$$D \times R(\frac{\pi}{2}).(\alpha v_{in} - v_2) = 0 \tag{2.10}$$

La solution est alors fournie par l'équation suivante :

$$\alpha = \frac{D \times R(\frac{\pi}{2}).v_2}{D \times R(\frac{\pi}{2}).v_{in}} \tag{2.11}$$

Cette équation n'est valide que si $D \times R(\frac{\pi}{2}).v_{in} \neq 0$. Si les modules agissant sur la vitesse sont utilisés, nous nous trouvons dans un cas de collision prévue entre les deux entités. Comme nous allons le voir, dans ce cas précis, cette situation ne peut pas se produire. Supposons que $D \times R(\frac{\pi}{2}).v_{in} = 0$, l'équation 2.10 peut se réécrire sous la forme suivante : $D \times R(\frac{\pi}{2}).v_2 = 0$. Nous en déduisons que v_{in} et v_2 sont tous deux colinéaires à D et donc que v_{in} et v_2 sont eux aussi colinéaires. Dans ce cas, $v_{in} - v_2$ est lui même colinéaire à D. Dans la mesure où D correspond à une direction permettant d'éviter la collision, cette situation ne peut pas provoquer de collision et donc le module ne peut être appelé. En conclusion, si le module est utilisé en condition normale, autrement dit dans un cas de collision prévue, l'inéquation $D \times R(\frac{\pi}{2}).v_{in} \neq 0$ est toujours vérifiée. L'équation 2.11 est donc toujours valide dans le cas d'une collision.

En fonction de la valeur de D (D_d ou D_g), l'équation 2.11 permet d'obtenir deux valeurs de α. Notons S l'ensemble des valeurs de α permettant d'éviter les collisions :

$$S = \{\frac{D_d \times R(\frac{\pi}{2}).v_2}{D_d \times R(\frac{\pi}{2}).v_{in}}, \frac{D_g \times R(\frac{\pi}{2}).v_2}{D_g \times R(\frac{\pi}{2}).v_{in}}\}$$

En fonction des valeurs contenues par cet ensemble, il est possible de définir les modules de décélération et d'accélération.

Module de décélération. Dans le cas du module de décélération, il s'agit de trouver une solution pour α telle que $0 \leqslant \alpha < 1$. Notons S_d l'ensemble des solutions de l'équation respectant cette contrainte :

$$S_d = \{x \mid x \in S \wedge 0 \leqslant x < 1\}$$

Si $card(S_d) = 0$, aucune solution n'est trouvée, le module n'est donc pas en mesure de fournir une solution adaptée à son rôle. Dans ce cas, le module lève une exception pour informer le module de recherche de solution de cette situation. Dans le cas contraire, la solution est $\alpha = max_{x \in S_d} x$.

Module d'accélération. Dans le cas du module d'accélération, il s'agit de trouver une solution pour α telle que $\alpha > 1$. Notons S_a l'ensemble des solutions de l'équation respectant cette contrainte :

$$S_a = \{x \mid x \in S \wedge x > 1\}$$

De la même manière que pour le module de décélération, si $card(S_d) = 0$, le module lève une exception pour informer le module de recherche de solution qu'aucune solution n'existe. Dans le cas contraire, la solution est $\alpha = min_{x \in S_d} x$.

Une autre information à prendre en compte lorsque que l'on génère une vitesse supérieure à la vitesse voulue du piéton est sa vitesse maximale admissible. Si la vitesse calculée, dépasse cette vitesse maximale, le module lève une exception pour spécifier qu'aucune solution n'a été trouvée.

Définition de la qualité La qualité de la réaction associée aux modules agissant sur la vitesse est définie en fonction du rapport entre la norme de la vitesse en entrée du module et la norme de la vitesse en sortie du module. Ce qui est défini par l'intermédiaire de la fonction suivante :

$$n(v_{in},v_{out},\beta) = (\frac{min(||v_{in}||,||v_{out}||)}{max(||v_{in}||,||v_{out}||)})^{\beta}$$

De la même manière que la fonction définie pour les comportements d'évitement à droite et à gauche, le paramètre β de cette fonction permet de changer sa forme.

Les modules qui viennent d'être décrits sont donc constitués d'un couple (mode de réaction, qualité de la réaction). D'autre part, ils peuvent tous être configurés par l'intermédiaire du paramètre β associé à leurs fonctions de qualité pour permettre de favoriser certains types de réaction en fonction de la configuration. La figure 2.12 montre l'influence du facteur β sur la forme de la courbe et donc la qualité associée aux réactions.

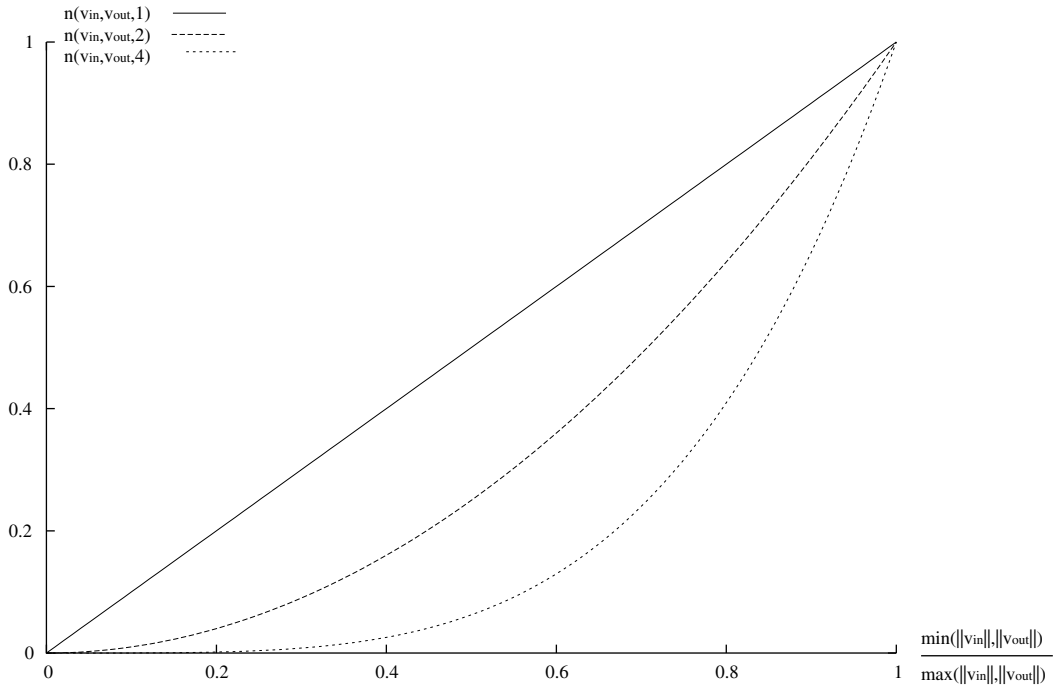


FIG. 2.12 – *Différentes forme de la fonction de qualité* $n(v_{in},v_{out},\beta)$.

2.5.2 Configuration du modèle

Différents modèles permettent de traduire les modes de navigation de l'être humain. Ils appliquent différentes formes de règles telles que les règles du code de la route [LW92], la règle des changements minimaux [Gof71] ou bien encore un compromis entre les deux [Tho99]. Notre modèle de navigation, en se basant sur une bonne configuration des modules permet de traduire ces divers modes de navigation. Par défaut, dans les différents modèles présentés, la collision de dos n'est pas

prise en compte. Nous considérons en effet que dans ce cas, l'entité qui va percuter le piéton devra elle-même éviter cette collision. Pour le cas des collisions avec des entités statiques, nous considérons que la règle de l'angle minimum est toujours appliquée. Enfin, quel que soit le type de collision, les modules d'accélération et de décélération peuvent être utilisés. Cependant, la décélération est favorisée par sa fonction de qualité ayant un exposant β valant 1 par rapport au module d'accélération qui possède un exposant β valant 5.

La configuration de la figure 2.13 permet de traduire les règles du code de la route (dans les pays roulant à droite, dans un pays roulant à gauche, il faut inverser les colonnes correspondant à la collision de face et à la collision arrière). Lorsqu'une collision de face est détectée, le choix est d'éviter cette collision sur la droite. Dans le cas d'un collision arrière, l'entité va doubler sur la gauche.

réaction \ collision	dans le dos	de face	arrière	statique
évitement droite		$c(v_{in},v_{out},1)$		$c(v_{in},v_{out},1)$
évitement gauche			$c(v_{in},v_{out},1)$	$c(v_{in},v_{out},1)$
ralentissement		$n(v_{in},v_{out},1)$	$n(v_{in},v_{out},1)$	
accélération		$n(v_{in},v_{out},5)$	$n(v_{in},v_{out},5)$	
acceptation	X			

FIG. 2.13 – *Modèle de piéton appliquant le code de la route.*

Le second modèle, présenté sur la figure 2.14, traduit la règle des changements minimaux. Cette règle consiste à privilégier le plus petit changement de direction. Pour ce faire, les modules d'évitement à droite et à gauche se voient attribuer la même fonction de qualité. La direction d'évitement choisie sera donc celle qui minimisera la différence angulaire (par définition des fonctions de qualité).

réaction \ collision	dans le dos	de face	arrière	statique
évitement droite		$c(v_{in},v_{out},1)$	$c(v_{in},v_{out},1)$	$c(v_{in},v_{out},1)$
évitement gauche		$c(v_{in},v_{out},1)$	$c(v_{in},v_{out},1)$	$c(v_{in},v_{out},1)$
ralentissement		$n(v_{in},v_{out},1)$	$n(v_{in},v_{out},1)$	
accélération		$n(v_{in},v_{out},5)$	$n(v_{in},v_{out},5)$	
acceptation	X			

FIG. 2.14 – *Modèle de piéton appliquant la règle des changements minimaux.*

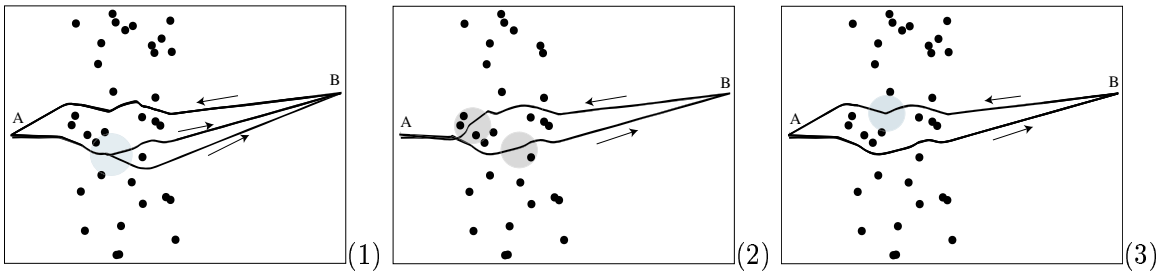
Enfin, le modèle le plus évolué, décrit sur la figure 2.15, est un compromis entre le code de la route et la règle de l'angle minimum. Nous considérons que lors d'une collision arrière, le piéton privilégie la règle de l'angle minimum. Dans le cas d'une collision de face, la règle d'évitement à droite est favorisée par sa fonction de qualité possédant un exposant β valant 1 comparativement à la règle d'évitement à gauche ayant un exposant β valant 4.

La figure 2.16 montre les formes de trajectoires qu'il est possible d'obtenir en utilisant les trois modèles qui viennent d'être présentés. Dans la mesure où il n'est pas facile de représenter sur un schéma l'interaction de plusieurs entités en mouvement, nous avons appliqué les règles des modèles

réaction \ collision	dans le dos	de face	arrière	statique
évitement droite		$c(v_{in},v_{out},1)$	$c(v_{in},v_{out},1)$	$c(v_{in},v_{out},1)$
évitement gauche		$c(v_{in},v_{out},4)$	$c(v_{in},v_{out},1)$	$c(v_{in},v_{out},1)$
ralentissement		$n(v_{in},v_{out},1)$	$n(v_{in},v_{out},1)$	
accélération		$n(v_{in},v_{out},5)$	$n(v_{in},v_{out},5)$	
acceptation	X			

FIG. 2.15 – *Modèle de piéton évolué, privilégiant l'évitement à droite mais appliquant à courte distance la règle de l'angle minimum.*

présentés sur les collisions avec des entités statiques (représentées par un point noir sur le schéma). Dans ce test, l'entité effectue des aller-retours entre deux points A et B, en évitant les autres entités. Sur la figure 2.16(1), le cercle en transparence caractérise une différence de trajectoire entre les différents allers-retours du modèle d'évitement à droite. Cette différence est due à la précision des calculs effectués, qui ne placent pas l'entité exactement au même endroit durant les différents déplacements. En fonction de la position limite au centre du cercle, la collision peut être détectée ou non. La figure 2.16(2) montre la trajectoire généré par le modèle utilisant la loi des changements minimaux. Les zones grisées mettent en évidence les différences de trajectoires adoptées par rapport au modèle d'évitement à droite. La trajectoire ainsi générée, tente, en fonction de la détection du voisinage direct, de toujours minimiser l'angle de déviation. Sur la figure 2.16(3), le cercle en transparence caractérise le biais (aussi présent sur les deux autres trajectoires) introduit par l'utilisation de ce voisinage. Les entités bouchant le passage ne sont détectées qu'au moment du point d'inflexion de la trajectoire provoquant ainsi ce changement « brusque ». Ce type de situation peut être évité en étendant la notion de voisinage et donc en n'utilisant plus uniquement le voisinage direct mais un voisinage plus grand (accessible rapidement par l'intermédiaire du graphe de voisinage). Cependant, cela nuit à la rapidité des algorithmes qui doivent alors prendre en compte un nombre de voisins beaucoup plus grand lors de la détection de collision. La trajectoire suivie par le modèle évolué montre le compromis qui est fait entre le respect du code de la route et la règle des changements minimaux avec une tendance à passer à gauche seulement dans le cas d'un petit angle de déviation.



(1) (2) (3)

FIG. 2.16 – *Comparaison des trois modèles de navigation. De gauche à droite : le modèle issu des règle du code de la route, le modèle des changements minimaux et le modèle évolué.*

La figure 2.17 montre un exemple de congestion à l'intérieur d'un goulet d'étranglement. La photographie de droite met en avant l'influence de la règle d'évitement à droite, qui est plus prioritaire

FIG. 2.17 – *Exemple de goulets d'étranglement.*

que la règle d'évitement à gauche dans le modèle évolué. Cette notion de priorité sur l'évitement à droite, permet, d'une part, de tenir compte d'une tendance connue du comportement piétonnier et, d'autre part, d'éviter les blocages lorsque deux flots se rencontrent dans un passage étroit. La tendance est alors à une auto organisation des flots avec un croisement à droite.

Durant la description des algorithmes de navigation réactive et de subdivision spatiale, aucune différence n'a été faite entre les environnements intérieurs et extérieurs. Les algorithmes présentés s'adaptent automatiquement à la taille de l'environnement et aux contraintes qu'il impose. La figure 2.18 montre un exemple de navigation de deux humanoïdes à l'intérieur d'un appartement, en utilisant les mêmes règles de navigation qu'en environnement extérieur.

Notre modèle de navigation s'avère donc générique et permet de traduire rapidement et simplement des comportements de navigation évolués, traités par un modèle purement réactif et donc rapide. Par une simple modification des abonnements des modules, les comportements de navigation peuvent changer et traduire des comportements différents en fonction des entités par exemple.

2.5.3 Performances

Malgré son apparente richesse, ce modèle s'avère rapide et permet de gérer des centaines d'entités en temps réel. La figure 2.19 montre l'évolution du coût de calcul d'un pas de temps de simulation en fonction du nombre de piétons. Le modèle est à même de gérer environ deux milles piétons simulés à dix hertz, en temps réel, sur un ordinateur équipé d'un athlon XP 1800+, tous calculs compris (construction du graphe de voisinage et simulation des piétons). Il est à noter que l'évolution du temps de calcul en fonction du nombre de piétons semble linéaire alors que la complexité de construction du graphe de voisinage est pour sa part logarithmique. Le temps de gestion du comportement de navigation doit donc amortir le temps de calcul du graphe de voisinage, le rendant négligeable. L'utilisation du voisinage direct lors des calculs de collision permet de stabiliser le temps de calcul associé à la gestion de la navigation d'une entité. Il le rend peu sensible au nombre de piétons peuplant l'environnement dans la mesure où, en prenant une approche globale, le nombre de voisins des entités est linéaire en fonction du nombre global d'entités.

FIG. 2.18 – *Exemple de navigation en environnement intérieur.*

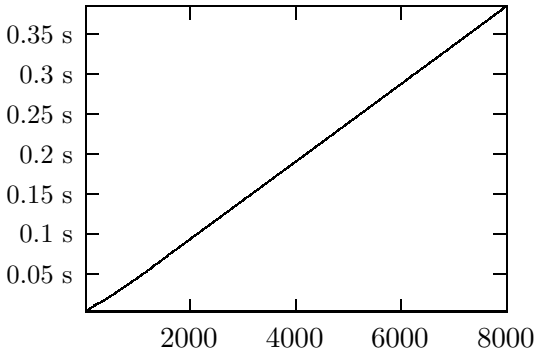


FIG. 2.19 – *Évolution du temps de calcul d'un pas de temps de simulation en fonction du nombre de piétons.*

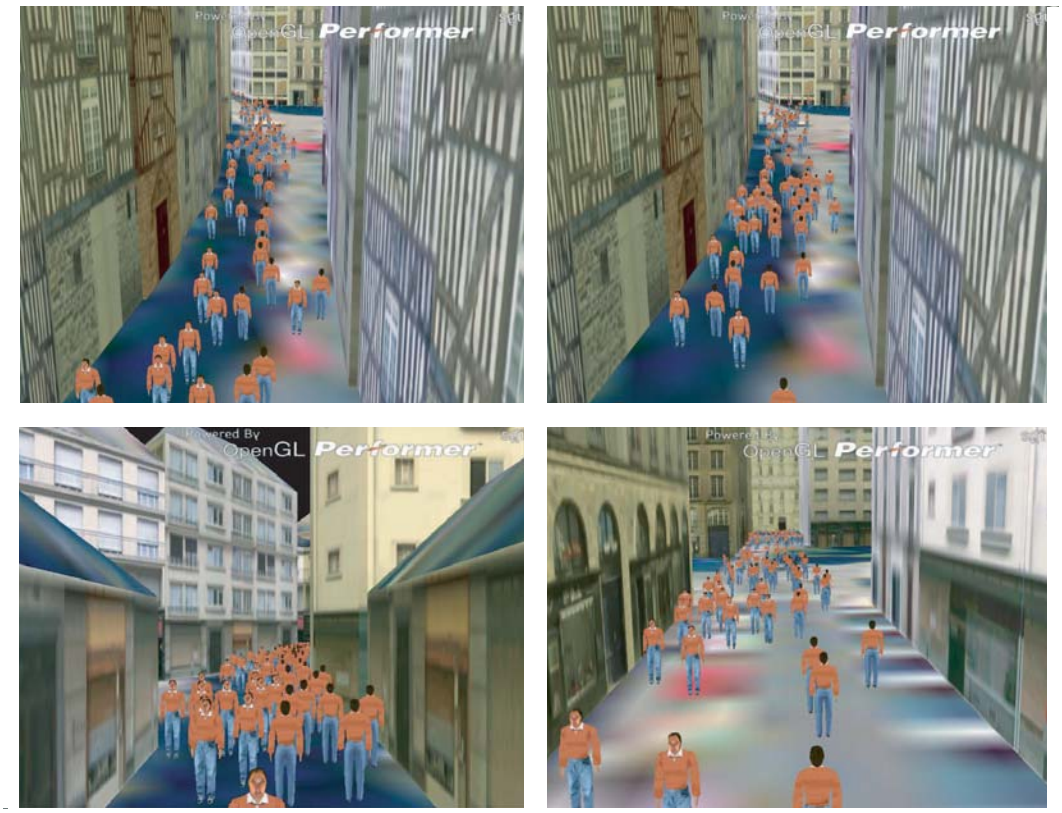


FIG. 2.20 – *Différentes rues peuplées du centre ville de Rennes.*

Ce modèle de navigation a été couplé avec la planification de chemin en vue de simuler le comportement piétonnier dans le centre ville de Rennes que nous avons fourni en exemple du chapitre précédent (voir fig. 2.20). Avec l'ajout de l'affichage de la ville et l'utilisation d'imposteurs $3d$ pour le rendu nous sommes à même de simuler aux alentours de 600 piétons dans une simulation fonctionnant à dix images secondes (sur un ordinateur équipé d'un processeur athlon 1800+ et d'une carte graphique basée sur un GPU de type RADEON 8500). Cependant, en utilisant quelques propriétés sur les fréquences de calcul (une possibilité offerte par la plate-forme OpenMASK), il est possible d'augmenter la vitesse de la simulation sans trop nuire à sa qualité. Le principe consiste à simuler les piétons à une fréquence de 25 hertz, en effectuant un calcul d'évitement de collision à 5 hertz (en utilisant un déphasage pour répartir la charge de calcul). La fréquence de calcul du graphe de voisinage est elle aussi fixée à 5 hertz et la fréquence de rendu à 25 Hertz. Nous sommes alors à même de simuler aux alentours de 2400 piétons avec une fréquence réelle de rafraîchissement de douze images secondes.

Conclusion

Nous venons de présenter un modèle de navigation réactive, s'inspirant des études sur le comportement piétonnier. Le suivi de chemin est effectué en se basant sur l'optimisation visuelle imitant

ainsi le comportement humain. La trajectoire alors empruntée, n'est pas directement dépendante de l'algorithme de planification de chemin (et donc du nombre et de la position des points de passages) mais du point de vue que l'entité possède sur son environnement. D'autre part, les possibilités de configuration qui sont offertes permettent de traduire une grande variété de modes de navigation issus de l'analyse du comportement humain. Le grand nombre d'entités qu'il permet de simuler permet d'effectuer des simulations microscopiques de grandes foules tout en permettant de reproduire des comportements de navigation réalistes. Ce type de résultat autorise son utilisation dans le cadre du test d'aménagements urbains ou bien dans le cadre de la simulation de comportements de panique en prenant en compte des comportements réflexes, plus organisés et réalistes, que ceux obtenus avec les modèles à base de particules. Comme ces derniers, le modèle proposé est cependant soumis au problème des minima locaux. Il existe des configurations, lors d'interactions avec des piétons statiques ou dans le cas d'une confrontation dans un couloir d'une largeur inférieure à l'envergure de deux entités, dans lesquelles les piétons peuvent se trouver bloqués lors de la navigation, et ne pas trouver de solution satisfaisante.

Une autre de ses caractéristiques importantes se situe dans l'utilisation du graphe de voisinage. Ce graphe explique grandement les bonnes propriétés du système. Ce calcul effectué de manière globale, d'une part, ne nuit pas à l'autonomie, et, d'autre part, fournit une structure topologique très riche grâce à la triangulation de Delaunay dont il est extrait. Il définit, au pire, un nombre linéaire de paires d'entités voisines tout en prenant en compte la notion de visibilité entre entités. En cela, il permet de stabiliser la complexité et donc le coût de calcul des algorithmes ayant besoin d'informations de voisinage. La contrepartie du gain obtenu par l'exploitation du voisinage direct se trouve dans la prédiction de collision qui peut, dans certains cas, être un peu tardive comparativement à celle de l'être humain. Cependant, au prix d'une diminution des performances, il est possible de considérer un voisinage plus grand, détecté en s'aidant de la structure topologique du graphe de voisinage.

En terme d'extensions, l'exploitation de ce graphe offre de grandes opportunités. Dans ce chapitre, le modèle de navigation qui a été présenté est centré sur l'autonomie de l'entité et s'avère être un modèle purement réactif qui peut être contrôlé par des comportements de plus haut niveau. Grâce aux informations de voisinage, il est possible de décrire rapidement des comportements de groupe tels que ceux proposés par Reynolds [Rey99] ou Raupp Musse [MGD99], tout en gardant une autonomie du piéton. D'autre part, la structure même de la triangulation de Delaunay ainsi que les informations sur les plus proches voisins permet d'accélérer les algorithmes de classification. L'utilisation de tels algorithmes permettrait alors de détecter automatiquement des groupes d'entités évoluant à peu près dans la même direction et donc d'éviter des groupes plutôt que des piétons. Il serait aussi possible de changer le comportement du piéton en fonction de son appartenance éventuelle à un groupe, en lui permettant d'alterner entre un comportement de navigation lié au groupe et un comportement de navigation autonome. Cela permettrait d'augmenter encore le réalisme tout en accélérant les calculs car un comportement de type nuée s'avère moins coûteux que le comportement par défaut que nous venons d'exposer.

Chapitre 3

Comportements réactifs

Les comportements réactifs sont des comportements ne nécessitant pas l'utilisation explicite d'une représentation des connaissances. Ils se réfèrent, suivant la classification de Newell [New90] au niveau tâche unitaire. Ils décrivent une suite d'opérations, pré-enregistrée, qui peut être vue de l'extérieur comme une brique de comportement stable et possédant sa propre cohérence interne.

Ces comportements peuvent être adoptés dans deux cadres : en réaction à l'environnement ou en action sur ce dernier. Les comportements adoptés en réaction à l'environnement sont des comportements réflexes, qui corrèlent directement la perception à l'action, sans intervention d'un processus cognitif. Lorsque ces comportements sont adoptés en action sur l'environnement, ils proviennent d'un processus cognitif. Ils constituent une brique élémentaire d'action, évoluée, qui contribue à la réalisation d'un but. Au niveau cognitif, ces comportements sont manipulés de manière abstraite, sans connaissance précise sur leur modalité de réalisation, seules leurs conséquences sont prises en compte.

En s'appuyant sur les analyses du comportement humain, les architectures proposées cherchent à offrir des mécanismes permettant de gérer la concurrence et le parallélisme [Don01, BW95]. Cependant, ces approches restent assez limitées dans leur utilisation. Elles nécessitent un certain nombre de précautions lors de la description des comportements pour assurer une cohérence, primordiale, lors de la réalisation. Le mélange de plusieurs comportements provenant de différentes sources (réflexes ou cognitives) nécessite des moyens de description particuliers ainsi qu'une architecture d'exécution permettant de les manipuler indépendamment les uns des autres. D'autre part, les comportements lors de leur réalisation interagissent. Pour donner un exemple, il suffit de s'imaginer assis à une terrasse de café, à lire un journal, à boire son café et à fumer une cigarette. La reproduction de ce comportement suppose un système, à même de prendre en compte l'importance relative des sous comportements et de les synchroniser correctement pour ne pas boire en fumant par exemple. Dans leur réalisation, les comportements s'adaptent les uns aux autres, de manière cohérente, pour se répartir les ressources corporelles (mains, yeux, bouche). Cette notion d'adaptation n'est pas présente dans les systèmes proposés, sauf par une description exhaustive du comportement particulier de lire, boire et fumer. Cependant, cette description va à l'encontre de la manipulation abstraite des comportements, nécessaire à l'interaction entre le processus cognitif, manipulant des abstractions, et le processus réactif, réalisant les comportements demandés et agissant de manière réflexe.

Dans ce chapitre, nous allons aborder cette problématique au travers d'un système permettant de décrire les comportements indépendamment les uns des autres, en vue d'obtenir une abstraction suffisante pour gérer automatiquement leur concurrence et leur adaptation. Ce système se décline

sous trois formes que nous allons présenter successivement :

- un mécanisme d'ordonnancement gérant automatiquement la synchronisation des comportements,
- un langage, intégrant des notions de modélisation objet pour la description et la manipulation abstraite des comportements,
- une architecture d'exécution ouverte permettant l'exploitation des caractéristiques du modèle.

Enfin, nous montrerons que ce modèle est à même de reproduire, automatiquement, l'exemple de la personne qui lit, boit et fume à partir d'une description indépendante des trois comportements.

3.1 Contexte

Dans cette section, nous allons d'abord effectuer un bref exposé des caractéristiques des comportements réactifs qui ont servi de point de départ à la conception de notre système. Puis, nous exposerons les hypothèses de départ de notre travail se basant sur une représentation particulière des comportements, tout comme le mode de gestion de leur réalisation en parallèle.

3.1.1 Les caractéristiques des comportements réactifs

Sur la base des travaux issus de l'analyse de l'architecture du comportement humain [New90] et des divers travaux qui ont étés menés en modélisation du comportement, nous avons extrait cinq caractéristiques importantes des comportements réactifs :

* La **concurrence**. Le comportement émergent d'un être vivant correspond à la résultante d'une concurrence entre plusieurs comportements. Chacun de ces comportements peut agir comme une unité spécialisée, permettant de résoudre un problème donné. Le comportement global est alors défini par un ensemble de choix successifs entre les divers sous-comportements. Cette notion est bien adaptée à l'informatique, où les architectures modulaires sont souvent préférées pour leur souplesse d'emploi. Il faut cependant être à même de gérer cette concurrence de manière cohérente.
* La **cohérence globale**. Le comportement émergent d'un être vivant, montre non seulement une certaine logique (même si parfois elle est difficile à comprendre), mais aussi une grande cohérence. Il existe un certain nombre de mécanismes permettant d'arrêter un comportement en cours d'exécution si celui-ci n'est plus adapté au contexte ou si un comportement est considéré comme plus important. Cependant, ces mécanismes d'arrêt de comportement doivent être manipulés avec précaution pour conserver la cohérence globale. Il n'est pas possible de stopper ou suspendre un comportement à tout moment ; ces moments dépendent de sa nature et de son déroulement.
* La **synchronisation**. Chaque comportement en cours d'exécution est limité par un certain nombre de ressources corporelles. Par exemple, il n'est pas possible de regarder deux choses exactement en même temps si ces deux choses ne sont pas dans le champ de vision. Il en est de même lors de la manipulation des objets, qui fait appel à la fois à la vision et à l'utilisation des mains pour la préhension. Cette notion de ressource corporelle est un facteur très limitant et fait partie de la problématique liée à la cohérence globale du comportement émergent.
* L'**adaptation**. L'être humain ne faisant que rarement une seule chose à la fois, les comportements sont en concurrence sur les ressources corporelles. Cette concurrence se traduit sous la

forme d'une modification de la réalisation du comportement. Il s'adapte, dans la mesure du possible, à la disponibilité des ressources.

* La **dépendance au contexte**. Un comportement, lorsqu'il est actif, est lié à un contexte. Que ce contexte soit cognitif ou lié à l'environnement, ou les deux, il détermine l'adéquation du comportement à la situation. Le monde étant actif, et pouvant évoluer indépendamment de l'être considéré, cette adéquation peut changer à tout moment.

Ces caractéristiques traduisent la problématique de la description des comportements réactifs. Deux questions peuvent alors être posées :

- comment définir des outils permettant à un concepteur de comportements pour les entités virtuelles humanoïdes d'exploiter tous ces concepts ?
- comment automatiser la gestion de la concurrence tout en permettant de respecter la cohérence globale ?

3.1.2 Une approche à base d'automates parallèles

Le formalisme des automates est très utilisé pour la description des comportements [BW95, CKP95, NT97, KAB+98, Don01]. Il permet de les décrire au travers d'une suite d'opérations élémentaires, symbolisées par les états. Les transitions décrivent l'enchaînement conditionnel des opérations. Elles permettent de traduire la cohérence interne du comportement et de prendre en compte l'influence des modifications de l'environnement sur son déroulement. L'introduction de la notion de parallélisme d'automates permet de retranscrire l'une des caractéristiques fondamentales du comportement humain, qui réside dans la réalisation de plusieurs tâches en simultanée. Cependant, cette notion pose le problème de la synchronisation et de la cohérence. Deux grandes méthodes ont été proposées :

- l'utilisation de sémaphores simples et de processus légers pour les PaT-Nets [BW95],
- une architecture hiérarchique de contrôle, basée sur un parallélisme simulé, pour HPTS [Don01] par exemple.

Ces approches n'abordent pas le problème de la synchronisation des comportements et de leurs possibilités d'adaptation dans son ensemble. Elles fournissent des moyens permettant de traiter ce problème au cas par cas et nécessitent, lors de la description de nouveaux comportements, de connaître les comportements déjà décrits, pour assurer la cohérence de l'exécution. Cependant, le formalisme associé aux automates offre plus qu'une possibilité de décrire un enchaînement logique d'opérations considérées comme atomiques ; il offre une structure, sous la forme d'un graphe, permettant de travailler à un plus haut niveau d'abstraction sur la globalité du comportement et des enchaînements d'opérations.

Notre système a pour but de répondre aux cinq points exposés dans la section précédente. Il vise à offrir des mécanismes permettant d'automatiser la concurrence et l'adaptation des comportements. Pour ce faire, il se base sur deux hypothèses de départ. D'une part, nous utilisons une représentation des comportements sous la forme d'automates car, comme nous le précisions précédemment, leur formalisme dispose d'une structure intrinsèque, utile pour les manipuler de manière abstraite. D'autre part, il présuppose une gestion particulière de leur déroulement en parallèle. Ce parallélisme doit être simulé ; l'exécution s'effectue alors par pas de temps et chaque automate, en un pas de temps, ne peut franchir qu'une seule transition. Ces deux hypothèses sont à la base du mécanisme d'ordonnancement que nous allons présenter.

3.2 Synchronisation des comportements

La reproduction de comportements réalistes nécessite d'être capable de gérer la concurrence des comportements. Cette concurrence se traduit le plus souvent sous la forme de contraintes d'utilisation de ressources corporelles telles que les mains, les yeux, les jambes... Une bonne gestion de cette concurrence se traduit par un ordonnancement des comportements en conservant d'une part leur cohérence, et, d'autre part, en respectant leur importance respective dépendante de plusieurs paramètres psychologiques et/ou physiologiques [LD02]. Pour être à même de gérer finement la concurrence des comportements, un certain nombre d'informations ont été ajoutées à la description des automates telles que la gestion des ressources ainsi que la définition de priorités et de degrés de préférence. Durant l'exécution, un algorithme d'ordonnancement combine toutes ces informations pour gérer automatiquement, en fonction de l'importance respective des comportements et de la disponibilité des ressources, leur concurrence et leur adaptation en fonction des conflits sur les ressources.

Notations. Pour la suite de la description, nous allons travailler sur des automates, leur structure ainsi qu'un certain nombre d'informations qui leurs sont associées. Les automates vont pouvoir utiliser un ensemble de ressources noté R et chaque automate va être défini par le sextuplet $A = (E_A,T_A,P_A,H_A,i_A,prio_A)$ dans lequel :

- E_A est l'ensemble des états de l'automate.
- $T_A \subset E_A \times E_A \times (() \rightarrow bool) \times (() \rightarrow [-1;1])$ est l'ensemble des transitions. Chacune d'entre elles possède une condition $(() \rightarrow bool)$ qui est une fonction sans paramètre renvoyant un booléen, ainsi qu'un degré de préférence traduit par une fonction renvoyant un résultat dans l'intervalle $[-1;1]$.
- $P_A : E_A \rightarrow 2^R$ est une fonction renvoyant l'ensemble des ressources utilisées par un état.
- $H_A : E_A \rightarrow 2^R$ est une fonction renvoyant l'ensemble des ressources héritées par un état.
- $i_A \in E_A$, avec $P_A(i_A) = \emptyset$, est l'état initial de l'automate.
- $prio_A : () \rightarrow \mathbb{R}$ est une fonction sans paramètre renvoyant la priorité de l'automate.

Il est à noter que chaque élément appartenant à la définition d'un automate est indexé par l'automate auquel il appartient. Ces notations vont être utilisées durant toute la suite de l'explication. D'autre part, dans la suite de la description nous allons considérer que n automates se déroulent en parallèle, nous les noterons A_1,...,A_n.

3.2.1 Les concepts

Les notions introduites pour permettre la gestion automatisée de la concurrence des comportements se déclinent sous deux formes :

- une partie exclusion mutuelle est gérée par un système de sémaphores associés à la notion de ressources ;
- une partie dynamique permet de spécifier l'importance d'un comportement à chaque pas de temps, ainsi que différentes possibilités d'adaptation, en fonction du contexte et de la disponibilité des ressources.

Nous allons définir ces concepts, qui servent de base au mécanisme d'ordonnancement.

3.2.1.1 Ressources

Une ressource correspond à la notion de sémaphore ; à chaque instant de l'exécution, une ressource ne peut être utilisée que par un et un seul automate permettant ainsi d'éviter les conflits éventuels entre plusieurs comportements. Pour traduire précisément la granularité d'utilisation des ressources, ainsi que pour créer une structure sur laquelle travailler, un automate peut spécifier l'ensemble des ressources utilisées dans chacun de ses états. Cette description possède une conséquence intéressante pour la description : les ressources sont automatiquement prises en entrant dans un état, relâchées lors de la sortie de l'état et conservées lors d'une transition entre deux états utilisant une même ressource. Leur granularité d'allocation est alors uniquement dépendante de la granularité de description de l'automate.

Compatibilité de ressources. La première contrainte à respecter durant l'exécution des automates se partageant des ressources est qu'une ressource ne soit utilisée que par un seul automate à chaque instant. Prenons e_1,...,e_n, n états appartenant respectivement à n automates A_1,...,A_n. Une combinaison d'états pour tous les automates est valide (i.e. ne pose pas de conflit de ressources) si la contrainte suivante est vraie:

$$\forall (i,j) \in \{1..n\}^2,\ i \neq j,\ P_{A_i}(e_i) \cap P_{A_j}(e_j) = \emptyset$$

Notons $P_k = \cup_{1 \leq i < k} P_{A_i}(e_i)$ l'ensemble contenant toutes les ressources utilisées par les états e_1 à e_k. Cette contrainte peut être étendue à n automates et est vérifiée si la formule suivante est vraie :

$$\begin{cases} c1_n(e1,...,e_n) & = & (P_{A_n} \cap P_n = \emptyset) \wedge c1_{n-1}(e_1,...,e_{n_1}) \\ c1_1(e_1) & = & vrai \end{cases} \tag{3.1}$$

La vérification de cette contrainte assure qu'aucun automate, n'utilise la même ressource au même instant. Cependant, elle n'est pas suffisante pour assurer le bon déroulement de l'exécution. En effet, les ressources étant des sémaphores, il faut être à même, pour assurer une totale liberté lors de la description des automates de détecter les cas éventuels d'inter-blocage.

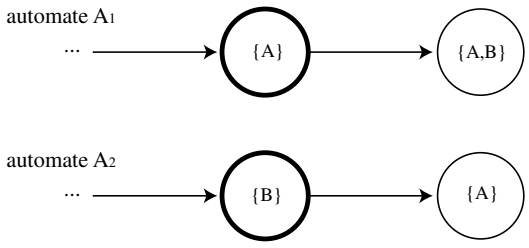


FIG. 3.1 – *Un exemple d'inter-blocage entre deux automates.*

Détection des inter-blocages. Le cas le plus simple pour donner un exemple d'inter-blocage est l'exemple du dîner. Deux personnes sont assises autour d'une table pour manger mais elles ne possèdent qu'un seul couteau et qu'une seule fourchette. La première personne possède le couteau, la seconde la fourchette. La première personne réclame la fourchette pour pouvoir manger, alors que la seconde personne réclame le couteau. Si aucune des deux personnes n'accepte de faire de

concession, la situation ne pourra pas évoluer et restera bloquée, il s'agit d'un inter-blocage. Dans le cas des automates, le principe est le même. La figure 3.1 montre un exemple d'inter-blocage entre deux automates, les états courants sont symbolisés par les cercles en gras. L'automate A_1 ne peut transiter sans prendre la ressource B et ne relâche pas la ressource A ; l'automate A_2 ne peut relâcher la ressource B sans prendre la ressource A. La situation est bloquée et les automates A_1 et A_2 ne peuvent pas poursuivre leur exécution. La question alors à se poser est : comment réagir. Soit l'application est bloquée (ce qui n'est pas souhaitable), soit un mécanisme détectant cet inter-blocage peut être mis en œuvre et il peut tuer l'un des automates en étant la cause. Dans ce cas, la cohérence des comportements n'est pas assurée. Pour assurer la cohérence, en déchargeant le concepteur de la gestion de ce problème, il faut fournir un mécanisme permettant d'éviter cette situation et permettant de conditionner les transitions d'un automate avec une contrainte de non inter-blocage.

Une transition entre deux états d'un automate utilisant des ressources crée une dépendance de ressources. Les ressources utilisées dans le premier état ne peuvent être relâchées que si celles de l'état suivant peuvent être prises. Pour prendre en compte ces dépendances, la notion de ressources héritées est utilisée. Ces ressources héritées sont associées à chaque état d'un automate utilisant des ressources (par l'intermédiaire de la fonction $H_A : E_A \rightarrow 2^R$) et représentent l'ensemble des ressources qui peuvent être utilisées dans le futur lorsque l'on se trouve dans cet état. Elles sont calculées par l'intermédiaire d'un graphe de dépendance $G_A = (E_{G_A}, T_{G_A})$ associé à l'automate A tel que :

- $E_{G_A} = E_A$ est l'ensemble des nœuds de ce graphe,
- $T_{G_A} = \{(e_1,e_2) \mid \exists c \in () \rightarrow bool, \exists p \in \mathbb{R}, (e_1,e_2,c,p) \in T_A \wedge P_A(e_1) \neq \emptyset \wedge P_A(e_2) \neq \emptyset\}$ est l'ensemble des arcs du graphe, représentant les transitions créant des dépendances de ressources.

Un exemple de graphe de dépendance associé à un automate est présenté sur la figure 3.2 ; seules les transitions connectant deux états utilisant des ressources sont conservées.

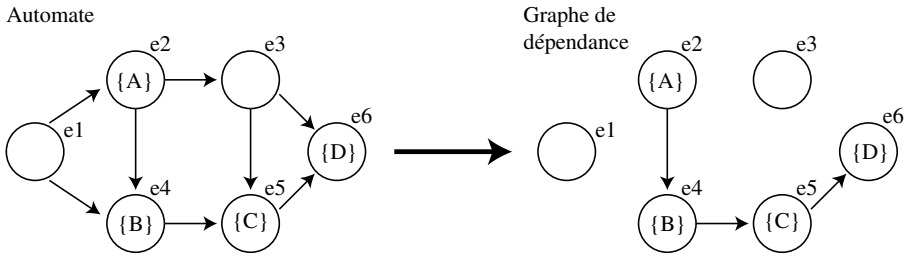


FIG. 3.2 – *Calcul du graphe de dépendance.*

La définition de $H_A : E_A \rightarrow 2^R$, l'application associant à chaque état de l'automate l'ensemble des ressources héritées, est la suivante :

$$\forall e \in E_A, H_A(e) = \bigcup_{(s \in E_{G_A}) \wedge (e \xrightarrow[G_A]{+} s)} P_A(s)$$

Ce calcul s'effectue donc par fermeture transitive du graphe, à partir de l'état considéré. En reprenant l'exemple de la figure 3.2, nous avons $H_A(e2) = P_A(e4) \cup P_A(e5) \cup P_A(e6) = \{B, C, D\}$. Ces

ressources héritées ne peuvent donc être calculées que lorsque la structure de l'automate est fixée (dynamiquement dans le cas d'automates construits dynamiquement, de manière statique sinon). En utilisant cette information, il est possible d'écrire une contrainte traduisant que deux états de deux automates ne peuvent pas provoquer d'inter-blocage s'il sont exécutés en parallèle. Soit A_1 et A_2 deux automates et $e_1 \in E_{A_1}$ et $e_2 \in E_{A_2}$ deux états ; e_1 et e_2 peuvent être exécutés en parallèle sans provoquer d'inter-blocage si :

$$(P_{A_1}(e_1) \cap H_{A_2}(e_2) = \emptyset) \vee (H_{A_1}(e_1) \cap P_{A_2}(e_2) = \emptyset)$$

Cette contrainte assure qu'au moins un des automates peut, depuis l'état considéré transiter vers son prochain état sans conflit de ressources, et par ce biais empêche tout inter-blocage. Cependant, elle détecte un sur-ensemble des inter-blocages car elle ne tient pas compte de l'ordre de prise des ressources dans le futur. Pour des raisons de performances, qui apparaîtront lors de la description de l'algorithme d'ordonnancement, elle doit pouvoir être calculée très rapidement, même au détriment de la précision de détection des inter-blocages. Cette contrainte peut être étendue à n états. Notons e_1,...,e_n, n états appartenant respectivement à n automates A_1,...,A_n. Notons $P_k = \cup_{1 \leq i < k} P_{A_i}(e_i)$ et $H_k = \cup_{1 \leq i < k} H_{A_i}(e_i)$; les états e_1,...,e_n peuvent être exécutés en parallèle sans provoquer d'inter-blocage si la contrainte suivante est vraie :

$$\begin{cases} c2_n(e1,...,e_n) & = & ((P_{A_n}(e_n) \cap H_n = \emptyset) \vee (H_{A_n}(e_n) \cap P_n = \emptyset)) \wedge c2_{n-1}(e_1,...,e_{n-1}) \\ c2_1(e_1) & = & vrai \end{cases} \quad (3.2)$$

Il s'agit d'une généralisation de la contrainte précédente, elle détecte donc aussi un sur-ensemble des inter-blocages et dépend, de plus, de l'ordre de présentation des états. Cependant, la complexité de vérification est en $O(n)$ pour n états en terme d'opérations sur des ensembles, ce qui rend possible un calcul intensif de prévention d'inter-blocage.

L'utilisation de ces contraintes de compatibilité de ressource (équation 3.1) et de prévention d'inter-blocage (équation 3.2) permet de trouver des configurations d'états pour tous les automates en cours d'exécution assurant la cohérence globale. Elles constituent le pas à franchir pour manipuler des automates sans aucune connaissance sur leur déroulement mais en étant assuré de la cohérence. Lors de la création d'un nouveau comportement, seul l'ensemble des ressources partagées doivent être prises en compte, mais la connaissance des autres comportements et de leur déroulement, n'est désormais plus nécessaire.

Ensemble des combinaisons valides. Plaçons nous dans le cas de n automates A_1,...,A_n fonctionnant en parallèle. A chaque pas de temps, les automates peuvent transiter dans les états extrémité des transitions possédant une condition vraie. Notons c_{A_k} l'état courant de l'automate A_k, et $\mathcal{E}_{A_k}$ l'ensemble des états valides pour le pas de temps courant, cet ensemble est calculé comme suit :

$$\forall k \in [1..n], \mathcal{E}_{A_k} = \{e \mid (c_{A_k},e,c,p) \in T_{A_k} \wedge c()\} \cup \{c_{A_k}\} \quad (3.3)$$

Il est constitué de tous les états extrémité des transitions sortant de l'état courant et ayant une condition vraie ainsi que de l'état courant lui-même. Notons $\mathcal{P} = \prod_{k=1}^{n} \mathcal{E}_{A_k}$ l'ensemble de toutes les configurations d'états possibles pour tous les automates fonctionnant en parallèle. De par la concurrence des automates sur l'ensemble des ressources, toutes ces combinaisons ne sont pas valides.

Certaines enfreignent la règle de compatibilité de ressources (équation 3.1) et d'autres peuvent produire des inter-blocages (équation 3.2). En utilisant ces deux contraintes, il est possible d'identifier un sous ensemble $\mathcal{P}_{comp}$ de $\mathcal{P}$ contenant l'ensemble des combinaisons valides :

$$\mathcal{P}_{comp} = \{(e_1,...,e_n) \in \mathcal{P} \wedge (e_1,...,e_n) \mid c2_n(e_1,...,e_n) \wedge c1_n(e_1,...,e_n)\} \tag{3.4}$$

Durant l'exécution des automates, et en fonction de leur état courant, cet ensemble, contient à chaque pas de temps l'ensemble des combinaisons viables, respectant les contraintes de conflit de ressources et de non inter-blocage. Cependant, parmi toutes ces combinaisons, il faut en choisir une comme devenant la nouvelle configuration des automates. Pour ce faire, les notions de degré de préférence et de priorité sont introduites.

3.2.1.2 Dynamique du comportement

Un comportement est dépendant du contexte ayant provoqué son exécution. Que ce contexte soit cognitif (exécution d'un plan), lié à l'environnement (réflexe) ou physiologique (soif, faim...), il définit la nature du comportement mais aussi et surtout, la raison de son exécution. Dans la mesure où l'environnement dans lequel une entité évolue est dynamique et non prédictible, l'adéquation du comportement au contexte ayant provoqué son exécution peut varier au cours du temps. D'autre part, la réalisation du comportement est dépendante d'un deuxième contexte qui est l'ensemble des autres comportements actuellement en cours de réalisation. Dans la mesure, où plusieurs comportement concurrents doivent se partager un ensemble de ressources restreint, ils peuvent entrer en conflit sur ces dernières. Dans ce cas, des compromis sont faits de manière à relâcher des ressources lorsque le besoin s'en fait sentir pour les concéder à un comportement plus « prioritaire », mais dans un soucis de cohérence. Pour automatiser ce mécanisme tout en laissant le choix à l'utilisateur de stipuler quand et comment relâcher des ressources de manière cohérente, deux notions sont introduites dans la description des automates : les degrés de préférence et la fonction de priorité.

Degré de préférence. Un degré de préférence est défini par l'intermédiaire d'une fonction, associée à une transition de l'automate, renvoyant une valeur réelle dans l'intervalle [-1 ; 1]. A un instant donné, la valeur de cette fonction correspond à la propension qu'a l'automate à utiliser cette transition lorsque la condition associée est vraie. En fonction de sa valeur, le degré de préférence (p) a différentes significations :

- $p > 0$: cette transition favorise la réalisation du comportement. Par défaut, à un instant donné, la transition possédant le plus grand degré de préférence et une condition vraie devrait être choisie.
- $p < 0$: cette transition ne favorise pas la réalisation du comportement. Ce type de transition est utilisé pour décrire une façon cohérente de stopper ou d'adapter le déroulement du comportement en relâchant des ressources.
- $p = 0$: cette transition n'influe pas sur le comportement. Elle peut être franchie ou non sans conséquence sur sa réalisation.

Ce degré de préférence permet de décrire des comportements sous la forme d'automates, en décrivant différentes possibilités d'adaptation lors de leur déroulement. Par défaut, les transitions possédant le plus grand degré de préférence sont choisies, mais dans un cas de conflit de ressource, ce coefficient permet de décrire le coût de la possibilité d'adaptation offerte par la transition à laquelle il est associé.

Priorité. La fonction de priorité est associée à un automate. Elle fournit une valeur numérique permettant de stipuler l'importance du comportement ou plus précisément l'adéquation du comportement au contexte ayant provoqué son exécution. En fonction du signe de son résultat (*prio*), elle possède deux significations et un cas particulier :

- $prio > 0$: Le comportement doit être réalisé et est adapté au contexte. Plus sa valeur est grande, plus le comportement est adapté et doit être réalisé.
- $prio < 0$: Le comportement est inhibé, sa valeur est alors interprétée comme le taux d'inadéquation entre le contexte et le comportement. Par défaut, les comportements possédant une priorité négative doivent être stoppés, mais de manière cohérente.
- $prio = 0$: Il s'agit d'un cas où le comportement n'est ni activé, ni inhibé. Le comportement ne devrait alors ni stopper ni continuer son déroulement. Cette valeur peut être utilisée de façon transitoire lors de l'évolution dynamique de la fonction de priorité, mais dans le cas général, elle doit être évitée de par son manque de signification réelle.

Cette fonction peut être utilisée pour contrôler (à haut niveau) un comportement durant sa phase d'exécution. Comme elle est définie par l'utilisateur, elle peut être corrélée à l'état interne de l'humanoïde (paramètres psychologiques, physiologiques ou ses intentions) ou à un stimulus extérieur. Elle fournit une manière simple d'influer sur le déroulement d'un comportement, sans pour autant connaître sa description dans les moindres détails.

3.2.2 Ordonnancement

Les notions de ressources, de degré de préférence ainsi que de priorité permettent de créer un algorithme d'ordonnancement favorisant l'exécution des comportements les plus prioritaires en forçant, de manière cohérente, l'adaptation de l'exécution des autres comportements. Cet algorithme effectue un calcul d'ordonnancement à chaque pas de temps permettant ainsi d'obtenir une certaine réactivité pour la prise en compte des modifications des priorités associées aux automates.

3.2.2.1 Description générale de l'algorithme

Parmi toutes les combinaisons d'états valides de l'ensemble $\mathcal{P}_{comp}$ (équation 3.4), il faut choisir la meilleure en respect des degrés de préférence associés aux transitions ainsi que de la priorité courante de l'automate. Pour ce faire, reformulons l'équation 3.3 de manière à associer à chaque état valide d'un automate le degré de préférence associé à la transition permettant de l'atteindre :

$$\forall k \in [1..n], \mathcal{E}'_{A_k} = \{(e,p()) \mid (c_{A_k},e,c,p) \in T_{A_k} \wedge c()\} \cup \{(c_{A_k},0)\} \tag{3.5}$$

Par défaut, un degré de préférence de 0 est associé à l'état courant de l'automate, stipulant ainsi que le fait de rester dans l'état courant ne pénalise ni ne favorise la réalisation du comportement. Une fonction $\mathcal{W}_{A_k} : \mathcal{E}_{A_k} \rightarrow \mathcal{R}$ associant un poids à chaque proposition valide d'un automate peut alors être construite en fonction de la valeur courante de la priorité $prio_{A_k}$ de l'automate A_k :

$$\forall e \in \mathcal{E}_{A_k}, \mathcal{W}_{A_k}(e) = Max_{(e,p)\in\mathcal{E}'_{A_k}} p \times prio_{A_k}$$

Considérons un poids $\mathcal{W}_{A_k}(e)$ associé à un état valide e de l'automate A_k. Ce poids a différentes significations :

- $\mathcal{W}_{A_k}(e) > 0$: l'automate a tendance à vouloir transiter dans l'état e:
  - $prio_{A_k} > 0 \wedge p > 0$: la transition dans cet état favorise la réalisation du comportement.

- $prio_{A_k} < 0 \wedge p < 0$: le comportement est inhibé. Les transitions favorisant sa terminaison de manière cohérente sont alors favorisées.

° $\mathcal{W}_{A_k}(e) < 0$: l'automate n'est pas enclin à transiter dans l'état e:

- $prio_{A_k} > 0 \wedge p < 0$: la transition dans cet état ne favorise pas la réalisation du comportement. Cependant, cette forme est utilisée pour proposer des possibilités d'adaptation cohérentes du déroulement du comportement en relâchant certaines ressources.
- $prio_{A_k} < 0 \wedge p > 0$: une telle proposition peut être utilisée pour proposer une adaptation du déroulement de l'exécution lorsque l'automate est inhibé.

° $\mathcal{W}_{A_k}(e) = 0$: le fait de transiter dans cet état ne favorise ni pénalise la réalisation du comportement, c'est notamment le cas pour l'état courant de l'automate.

Pour être en mesure de sélectionner la meilleure combinaison dans l'ensemble $\mathcal{P}_{comp}$, un poids est associé à chacune d'entre elles. Ce poids est fourni par la fonction $\mathcal{W} : \mathcal{P}_{comp} \rightarrow \mathcal{R}$ associant à chaque combinaison de $\mathcal{P}_{comp}$ la somme des poids des états la constituant :

$$\forall (e_1,...,e_n) \in \mathcal{P}_{comp}, \mathcal{W}(e_1,...,e_n) = \sum_{i=1}^{n} \mathcal{W}_{A_k}(e_i)$$

Le résultat fourni par cette fonction peut être interprété comme le taux de satisfaction global des automates si la combinaison obtenant ce poids est sélectionnée. Dans cette mesure, les combinaisons d'états maximisant ce poids sont celles qui globalement maximisent le degré de satisfaction de la plus grande partie des automates. Les comportements ayant la plus grande priorité vont être favorisés dans leur réalisation alors que ceux qui sont inhibés ou qui ont la plus faible priorité vont relâcher leurs ressources si c'est possible et de façon cohérente. L'adaptation du déroulement des différents comportements devient alors automatique. Notons $\mathcal{M} = Max_{(e_1,...,e_n) \in \mathcal{P}_{comp}} \mathcal{W}(e_1,...,e_n)$ le meilleur taux de satisfaction trouvé dans l'ensemble $\mathcal{P}_{comp}$. L'ensemble des meilleurs combinaisons est alors défini par :

$$\mathcal{P}_{best} = \{(e_1,...,e_n) \mid (e_1,...,e_n) \in \mathcal{P}_{comp} \wedge \mathcal{W}(e_1,...,e_n) = \mathcal{M}\} \tag{3.6}$$

Après la création de l'ensemble $\mathcal{P}_{best}$, le système d'ordonnancement peut élire l'une de ces combinaisons comme le nouvel état global du système et donc provoquer les transitions des automates dans les états ainsi sélectionnés. Cette méthode d'ordonnancement assure la cohérence des comportements dans la mesure où elle n'exploite que des propositions de transitions faites par les automates et qui sont donc supposées cohérentes. D'autre part, elle essaie de satisfaire au mieux l'ensemble des comportements en fonction de leur importance respective et de leurs besoins en ressources.

3.2.2.2 Optimisation de l'algorithme

En supposant que l'ordonnancement est calculé à chaque pas de temps de l'exécution, il est utile de limiter la complexité du calcul. En utilisant la méthode qui vient d'être décrite la complexité pourrait paraître être de $\prod_{k=1}^{n} card(\mathcal{E}_{A_k})$ en terme de combinaisons calculées et de vérification de contraintes de ressources. Cependant, les contraintes exprimées dans les équations 3.1 et 3.2 n'utilisent que les ensembles de ressources prises et héritées dans un état de l'automate. D'autre part, lors du calcul des poids associés à chaque état, seul le meilleur poids obtenu en combinant les degrés de préférence et la priorité est important. L'exploitation de ces propriétés permet de réduire la complexité de l'algorithme.

Dans un premier temps, exprimons sous forme incrémentale la construction de l'ensemble $\mathcal{P}_{comp}$ tout en vérifiant les contraintes liées aux ressources :

$$\begin{cases} \mathcal{P}_{comp}^{(n)} & = & \{(e_1,...,e_n) & | & e_n \in \mathcal{E}_{A_n} \wedge (e_1,...,e_{n-1}) \in \mathcal{P}_{comp}^{(n-1)} \wedge \\ & & & & ((P_{A_n}(e_n) \cap H_n = \emptyset) \vee (P_n \cap P_{A_n}(e_n) = \emptyset)) \wedge (P_{A_n}(e_n) \cap P_n = \emptyset)\} \\ \mathcal{P}_{comp}^{(1)} & = & \mathcal{E}_{A_1} & & \end{cases}$$

Le calcul incrémental de l'ensemble $\mathcal{P}_{comp}$ permet, à chaque niveau, de supprimer les combinaisons violant les contraintes de ressources dès qu'elles sont trouvées. D'autre part, la vérification des contraintes au niveau $k > 1$ utilise seulement:

- les ressources prises et héritées associées à un état de $\mathcal{E}_{A_k}$,
- l'union des ressources prises et l'union des ressources héritées associées aux états constituant une combinaison de $\mathcal{P}_{comp}^{(k-1)}$.

Notons $\mathcal{R}^{(k)}$ l'ensemble des couples composés de l'union des ressources prises et de l'union des ressources héritées associés à chaque combinaison d'état de $\mathcal{P}_{comp}^{(k)}$. Considérons un couple (P,H) de $\mathcal{R}^{(k)}$. Ce couple est associé à un ensemble de combinaisons de $\mathcal{P}_{comp}^{(k)}$, notons $S_{(P,H)}^{(k)}$ cet ensemble de combinaisons. Toutes les combinaisons de $S_{(P,H)}^{(k)}$ sont équivalentes du point de vue de la vérification des contraintes de ressources au niveau $k+1$. Leur différence réside uniquement dans le poids qui leur est associé. Et seules les combinaisons maximisant ce poids pourront participer à la création de l'ensemble $\mathcal{P}_{best}$. Il est donc possible de créer une fonction de filtrage $\mathcal{F}^{(k)}$ permettant de filtrer les combinaisons de $\mathcal{P}_{comp}^{(k)}$. Cette fonction génère un sous-ensemble de $\mathcal{P}_{comp}^{(k)}$ ne conservant pour chaque couple $(P,H) \in R^{(k)}$ qu'une des combinaisons de $S_{(P,H)}^{(k)}$ maximisant le poids associé (la première trouvée pour assurer la stabilité de l'algorithme). Dès lors, la propriété suivante peut être déduite : $card(\mathcal{F}^{(k)}(\mathcal{P}_{comp}^{(k)})) = card(R^{(k)})$. L'application de cette fonction de filtrage ne modifie pas le résultat de l'algorithme. Le choix de l'une des combinaisons maximisant le poids est fait durant la phase de calcul, au lieu d'être faite à la fin de cette dernière. D'autre part, les couples $(P,H) \in 2^R \times 2^R$, donc la propriété suivante peut être déduite :

$$card(\mathcal{F}^{(k)}(\mathcal{P}_{comp}^{(k)})) \leq min(\prod_{i=1}^{k} \mathcal{E}_{A_i}, 2^{2 \times card(R)})$$

La fonction $\mathcal{F}^{(k)}$ permet de limiter le nombre de combinaisons calculées et conservées pendant le déroulement de l'algorithme et limite donc la complexité du calcul. D'autre part, cette fonction peut aussi être appliquée à chaque ensemble $\mathcal{E}_{A_k}$ pour supprimer les doublons éventuels avant le calcul. Finalement, l'algorithme s'exprime de la manière suivante :

$$\begin{cases} \mathcal{P}_{comp}^{(n)} & = & \{(e_1,...,e_n) & | & e_n \in \mathcal{F}^{(1)}(\mathcal{E}_{A_n}) \wedge (e_1,...,e_{n-1}) \in \mathcal{F}^{(n-1)}(\mathcal{P}_{comp}^{(n-1)}) \wedge \\ & & & & ((P_{A_n}(e_n) \cap H_n = \emptyset) \vee (P_n \cap P_{A_n}(e_n) = \emptyset)) \wedge (P_{A_n}(e_n) \cap P_n = \emptyset)\} \\ \mathcal{P}_{comp}^{(1)} & = & \mathcal{F}^{(1)}(\mathcal{E}_{A_1}) & & \end{cases}$$

Après le calcul de l'ensemble $\mathcal{P}_{comp}$, l'ensemble des propositions obtenant le meilleur poids est extrait et la meilleure combinaison est élue comme le nouvel état global de l'ensemble des automates.

Cet algorithme permet de gérer aux alentours d'une dizaine de ressources en temps réel et sans problème de mémoire, en gérant le comportement de plusieurs humanoïdes. Outre la complexité du calcul, deux phases critiques d'implémentation sont à distinguer. D'une part, les calculs ensemblistes peuvent prendre beaucoup de temps s'ils sont implémentés par la méthode classique d'arbre binaire. La recommandation ici est d'utiliser une représentation des ensembles par champs de bits, rendue possible car le nombre de ressources exploitées par le système est connu avant l'ordonnancement. Les calculs d'intersection se bornent alors à des opérations de type "ET bit à bit", réduisant ainsi considérablement le coût de calcul. D'autre part, la fonction de filtrage, pour être performante peut être implémentée par le biais d'une table de hachage exploitant le représentation spécifique des ensembles. Ces considérations peuvent sembler techniques mais conditionnent grandement la performance de l'algorithme, il est donc nécessaire de les prendre en compte.

3.2.2.3 Exemple de déroulement de l'algorithme

Pour montrer le fonctionnement de cet algorithme, prenons un exemple simple basé sur la concurrence de trois automates (a_1,a_2,$a3$) sur un ensemble de trois ressources (E,H_r,H_l) représentés sur la figure 3.3. La valeur associée aux transitions correspond à la valeur du degré de préférence, les ensembles contenus dans les états correspondent aux ressources prises dans cet état.

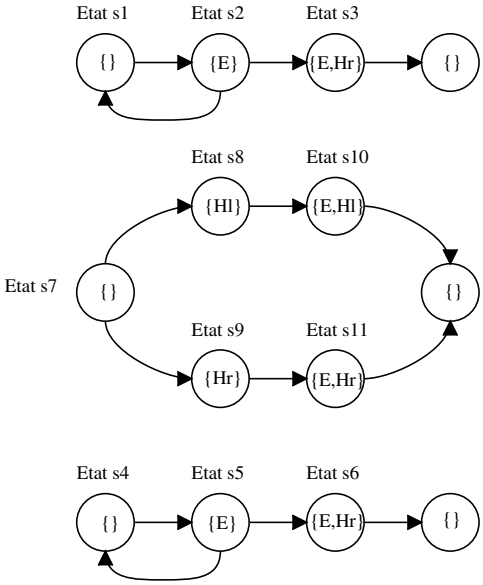


FIG. 3.3 – *Trois automates fonctionnant en parallèle.*

Les états courants des automates a_1, a_2, a_3 ainsi que leurs priorités sont respectivement : (s_1,11), (s_5,4), (s_7,6). En supposant que les transitions associées aux automates possèdent toutes des conditions vraies, la table de la fig. 3.4 fournit l'ensemble des états valides avec les ressources prises et héritées pour chaque automate ainsi que le poids qui leur est associé par l'algorithme d'ordonnancement.

	Etats Valides	Ressources prises	Ressources héritées	Poids
a1	s1	$\emptyset$	$\emptyset$	0
	s2	$\{E\}$	$\{E,H_r\}$	$1 \times 11 = 11$
a2	s4	$\emptyset$	$\emptyset$	$-1 \times 4 = -4$
	s5	$\{E\}$	$\{E,H_r\}$	0
	s6	$\{E,H_r\}$	$\emptyset$	$1 \times 4 = 4$
a3	s7	$\emptyset$	$\emptyset$	0
	s8	$\{H_l\}$	$\{E,H_l\}$	$0.7 \times 6 = 4.2$
	s9	$\{H_r\}$	$\{E,H_r\}$	$1 \times 6 = 6$

FIG. 3.4 – *Les attributs des états.*

Première itération. Sur la base des poids associés aux états, les calculs d'ordonnancement peuvent être effectués. Dans un premier temps, les états proposés par le premier automate sont pris en compte pour initialiser la première phase de calcul. La table de la figure 3.5 montre le résultat du calcul obtenu pour $\mathcal{F}^{(1)}(\mathcal{P}^{(1)}_{comp})$.

Ressources prises	Ressources héritées	Combinaison d'états	Poids
$\emptyset$	$\emptyset$	$(s1)$	0
$\{E\}$	$\{E,H_r\}$	$(s2)$	11

FIG. 3.5 – $\mathcal{F}^{(1)}(\mathcal{P}^{(1)}_{comp})$.

Seconde itération. Durant cette itération, la compatibilité entre les états de $\mathcal{F}^{(1)}(\mathcal{P}^{(1)}_{comp})$ et les états valides de l'automate a_2 est effectué. L'ensemble $\mathcal{F}^{(2)}(\mathcal{P}^{(2)}_{comp})$ est construit en filtrant les combinaisons valides (voir fig. 3.6). Les combinaisons (s_5,s_1) et (s_2,s_4) possèdent le même couple de ressources prises et héritées. La fonction de filtrage ne garde donc que la combinaison (s_2,s_4) car elle obtient le meilleur poids. La fonction de filtrage réduit en cours de calcul la taille de l'ensemble des combinaisons stockées en supprimant les combinaisons qui ne pourront pas participer à la construction de la solution fournie par l'algorithme.

Ressources prises	Ressources héritées	Combinaison d'états	Poids
$\{E\}$	$\{E,H_r\}$	(s2,s4)	7
$\{E,H_r\}$	$\emptyset$	(s1,s6)	4
$\emptyset$	$\emptyset$	(s1,s4)	-4

FIG. 3.6 – $\mathcal{F}^{(2)}(\mathcal{P}^{(2)}_{comp})$.

Troisième itération. Cette itération calcule la compatibilité des combinaisons d'états de $\mathcal{F}^{(2)}(\mathcal{P}^{(2)}_{comp})$ avec les états valides de l'automate a_3. Finalement, l'ensemble $\mathcal{F}^{(3)}(\mathcal{P}^{(3)}_{comp})$ est calculé en filtrant les combinaisons valides suivant leur ensembles de ressources prises et héritées (voir fig. 3.7).

Ressources prises	Ressources héritées	Combinaison d'états	Poids
$\{E,H_l\}$	$\{E,H_l,H_r\}$	(s2,s4,s8)	11.2
$\{E,H_l,H_r\}$	$\{E,H_l\}$	(s1,s6,s8)	8.2
$\{E\}$	$\{E,H_r\}$	(s2,s4,s7)	7
$\{E,H_r\}$	$\emptyset$	(s1,s6,s7)	4
$\{H_l\}$	$\{E,H_l\}$	(s1,s4,s8)	0.2
$\emptyset$	$\emptyset$	(s1,s4,s7)	-4

FIG. 3.7 – $\mathcal{F}^{(3)}(\mathcal{P}^{(3)}_{comp})$.

Choix de la meilleure combinaison. Finalement, la combinaison de l'ensemble $\mathcal{F}^{(3)}(\mathcal{P}^{(3)}_{comp})$ maximisant le poids associé est choisie. Il s'agit de la combinaison d'états ($s2$, $s4$, $s8$). L'automate $a1$ est donc contraint de retourner dans l'état $s2$ pour permettre la transition de l'automate $a2$. L'automate $a3$, pour sa part, adapte son exécution en transitant dans l'état $s8$ alors qu'il aurait préféré transiter dans l'état $s9$.

Cet exemple démontre l'influence des priorités dans l'algorithme d'ordonnancement, ainsi que l'adaptation automatique des transitions des automates pour respecter les contraintes de ressources. Cette adaptation se passe sans communication entre les automates, facilitant ainsi le travail de description et assurant la cohérence de leur déroulement.

3.2.3 Discussion sur le modèle

Grâce à l'introduction des notions de ressources, de degré de préférence et de priorité à l'intérieur de la description des automates, il devient possible de manipuler des comportements sans avoir aucune connaissance sur leur déroulement.

Les ressources permettent de spécifier les contraintes d'exclusion mutuelles entre les différents comportements. Une fois l'humanoïde caractérisé par un ensemble de ressources, la description des comportements peut se faire sans aucune connaissance sur les comportements déjà décrits. L'algorithme, par l'intermédiaire des contraintes de compatibilité de ressources et d'évitement d'interblocages, peut prendre en charge l'exécution et la synchronisation de plusieurs comportements sans intervention extérieure. La granularité de description de ressources est laissée au choix de l'utilisateur, tout comme leur granularité d'utilisation, qui est uniquement dépendante du nombre d'états constituant l'automate.

L'ajout des degrés de préférence permet au sein du même comportement de décrire plusieurs variantes d'exécution, en spécifiant le coût associé aux possibilités d'adaptation offertes. Dès lors, le système d'ordonnancement se charge, et une fois encore sans intervention extérieure, de trouver la façon d'exploiter au mieux les possibilités qui lui sont offertes. Comme ce dernier ne peut exploiter que des propositions de transitions liées aux automates et ne peut en aucun cas prendre une décision

en dehors du champ de description, la cohérence des comportements est assurée dans la mesure où l'on suppose que le comportement a été décrit de manière cohérente.

Enfin, le système de fonction de priorité permet de définir l'importance relative des comportements et donc d'influer sur la gestion de leur concurrence en gardant à l'esprit que les comportements les plus prioritaires seront favorisés dans leur exécution. Ces fonctions offrent donc un contrôle très abstrait sur le déroulement des comportements, qui sont alors manipulables de l'extérieur sans aucune connaissance sur leur déroulement.

Cependant, ce système est très dépendant des valeurs associées aux degré de préférences ainsi qu'aux priorités. Une mauvaise utilisation des ces valeurs, peut provoquer des instabilités qui se traduisent sous la forme d'oscillations. La fonction de priorité doit donc être définie de manière à ne pas osciller ou bien à osciller sur une longue période. D'autre part, le poids associé à une proposition d'un automate est à la fois dépendant de la priorité et du degré de préférence. Le même phénomène, de manière un peu différente, peut se produire avec les degrés de préférence. Supposons que nous nous trouvions dans le cas décrit par la figure 3.8. Les deux états entourés de noir sont considérés comme exclusifs et chaque transition est étiquetée avec la condition (vraie) et le degré de préférence qui lui sont associés. Les deux configurations d'états possibles et valides entre l'automate a et l'automate b sont donc $c_1 = \{a1, b2\}$ ou $c_2 = \{a2, b1\}$. Prenons c_1 comme configuration de départ. La configuration c_2 possède alors un poids de $-0.3 \times 2 + 1 = 0.4$, elle est donc élue comme nouvelle configuration. Mais, depuis c_2, la configuration c_1 possède alors un poids de $0.7 \times 2 - 1 = 0.4$, une transition va donc être effectuée pour revenir dans la configuration c_1. Le système oscille. Cette oscillation est due au déséquilibre introduit dans les degrés de préférence des transitions de l'automate a. Pour éviter ce type d'oscillation, il faut soit posséder des conditions correctes sur les transitions, soit assurer que le degré de préférence de la transition sortant de l'état $a2$ est l'opposé du degré de préférence de la transition y entrant. Dans ces deux cas, le système est stable.

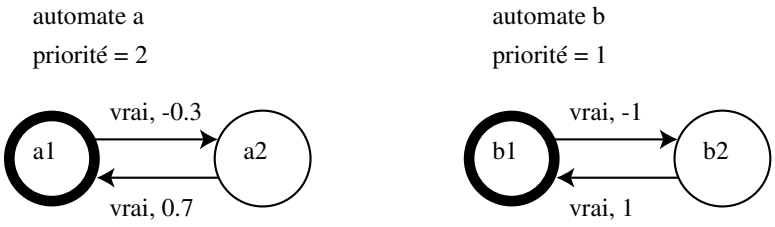


FIG. 3.8 – *Exemple d'oscillation due à un mauvais paramétrage du degré de préférence.*

Le système d'ordonnancement automatique offre de grandes opportunités quand à la conception des mondes virtuels et à l'exploitation de l'approche de Kallmann sur les *smart objects*. En effet, il devient possible non seulement d'acquérir par le biais de l'environnement les moyens d'interagir avec ce dernier, mais en plus de supprimer la contrainte d'exécution séquentielle imposée pour le respect de la cohérence du comportement global. Enfin, il devient possible de mélanger des comportements réflexes (attention visuelle passive, réflexe de peur...) avec des comportements associés à la réalisation d'un plan sans prendre explicitement en compte la concurrence alors imposée, le système s'en chargeant pour l'utilisateur.

3.3 HPTS++ : un langage et une architecture d'exécution

HPTS++ [1] est la dernière évolution du modèle HPTS, intégrant la notion de synchronisation automatique de comportements qui vient d'être décrite. Il s'agit d'une refonte complète intégrant de nouveaux concepts ainsi qu'une nouvelle architecture d'exécution. Le but ici est de fournir une architecture modulaire pour offrir un maximum de liberté dans la conception et l'utilisation du modèle dans le cadre de l'animation comportementale, mais aussi d'introduire dans la conception des comportements les notions issues des langages orientés objets. Cela permet d'offrir dans la modélisation des comportements les mêmes niveaux d'abstraction et de fonctionnalités que ceux offerts par la programmation objet en général. HPTS++ se scinde en deux parties :

- d'une part un compilateur permettant de compiler des automates décrits par l'intermédiaire du langage en classes C++ pouvant être elles même compilées avec les compilateurs C++ standards.
- d'autre part, un noyau d'exécution (fourni sous la forme d'un contrôleur d'exécution) proposant toutes les fonctionnalités nécessaires à l'exécution des automates ainsi que des modules effectuant les calculs d'ordonnancement qui viennent d'être décrits.

Le système permet de gérer des automates fonctionnant en parallèle hiérarchiques ou non. Le parallélisme est simulé. Autrement dit, l'exécution est synchrone et l'on considère l'évolution des automates par pas de temps de simulation (la durée d'un pas de temps étant laissée au choix de l'utilisateur).

3.3.1 Description des automates

L'utilisation de l'ancienne version d'HPTS nous a permis de constater un grand nombre de limitations, d'une part sur la facilité d'interfaçage avec les applications mais aussi sur le langage lui-même n'intégrant qu'une sous partie très limitée de la grammaire de C++ et ne gérant pas les notions objets telles que l'héritage ou le polymorphisme. Pour pallier ces problèmes et offrir des fonctionnalités très utiles à la description des comportements, le langage a été défini comme étant une sur-couche de C++. Cette notion permet de profiter de tous les mécanismes déjà mis en œuvre tels que :

- les notions d'héritage,
- les notions de polymorphisme,
- le RTTI (Run Time Type Information), permettant d'obtenir dynamiquement des informations sur les types de données manipulées.

La description se fait alors par l'intermédiaire d'un ensemble de champs définis par la grammaire du langage qui est résumée sur la figure 3.9 et dont l'intégralité est fournie en annexe B. A l'intérieur de ces champs, au même titre que pour une méthode d'objet, du code, exprimé en langage C++ natif, peut être ajouté. La communication avec le moteur d'exécution se fait alors par l'utilisation d'une API standardisée, laissant ainsi à l'utilisateur une très grande liberté de description.

1. Ce logiciel a fait l'objet d'un dépôt à l'A.P.P. (Agence de Protection des Programmes) sous le numéro $IDDN_{(1)}FR_{(2)}001_{(3)}290017_{(4)}000_{(5)}S_{(6)}P_{(7)}2003_{(8)}000_{(9)}10400_{(10)}$ et appartient à l'Université de Rennes I et au C.N.R.S. . Il est actuellement utilisé dans le cadre du projet AVA motion, regroupant les sociétés Kineo Works, Daesign et les laboratoires de recherche de l'université de Rennes II et l'IRISA. Dans le cadre de ce projet, la société Daesign utilise le modèle en remplacement de Motion Factory, pour la modélisation du comportement de personnages autonomes dans le cadre de la fiction interactive.

```
state machine ClasseAuto : state machine AutoAncetre, class ClasseAncetre
{
    variables
    {{ /* Code C++ : données membres, constructeur/destructeur, méthodes */ }}

    priority {{ /* Expression renvoyant une donnée de type float */ }}
    before step {{ /* Code factorisé exécuté en début de pas de temps */ }}
    after step {{ /* Code factorisé exécuté en fin de pas de temps */ }}
    kill {{ /* Réaction au signal arrêtant l'exécution de l'automate */ }}
    suspend {{ /* Réaction au signal suspendant l'exécution de l'automate */ }}
    resume {{ /* Réaction au signal reprenant l'exécution de l'automate */ }}

    initial state {{ /* Expression C++ renvoyant l'état initial de l'automate}}

    state identifiantEtat
    {
        uses {{ /* Liste de ressources */ }};
        entry {{ /* Code exécuté lors de l'entrée dans l'état */ }}
        during {{ /* Code exécuté tant que l'automate reste dans l'état */ }}
        exit {{ /* Code exécuté lors de la sortie dans l'état */ }}
        kill {{ /* Réaction au signal arrêtant l'exécution de l'automate */ }}
        suspend {{ /* Réaction au signal suspendant l'exécution de l'automate */ }}
        resume {{ /* Réaction au signal reprenant l'exécution de l'automate */ }}
    }
    transition identifiantTransition
    {
        origin {{ /* Etat d'origine de la transition */ }}
        extremity {{ /* Etat d'extrémité de la transition */ }}
        preference {{ /* Expression renvoyant une donnée de type float */ }}
        condition {{ /* Expression renvoyant une donnée de type bool */ }}
        action {{ /* Code exécuté lors du passage de la transition */ }}
    }
}
```

FIG. 3.9 – *Schéma de description d'un automate. La grammaire complète peut être consultée en annexe B. Les blocs entre doubles accolades "{{" et "}}" délimitent les zones dans lesquelles l'utilisateur exprime du code en C++ natif.*

Héritage. La notion d'héritage en programmation objet permet de créer des niveaux d'abstractions. Cette notion est utilisée dans HPTS++ qui permet de décrire des automates comme des classes C++ et donc offre deux types d'héritage : l'héritage d'automate et l'héritage de classes C++. La notion d'héritage d'automate est directement corrélée à la notion de comportement abstrait. Elle permet de catégoriser les comportements : comportements de préhension, comportements de navigation... Ces catégories de comportement peuvent ensuite être raffinées par héritage (comportement de préhension à une main, comportements de préhension à deux mains). Le second type d'héritage permet de simplifier l'interfaçage entre une plate-forme de simulation quelconque et les automates. Pour donner un exemple, il est possible de créer un comportement par défaut héritant d'une classe gérant un humanoïde virtuel, on obtient alors un humanoïde pouvant se comporter dans un environnement.

Interface C++. Cette partie se décrit dans le champ **variables** associé à l'automate. Elle permet de spécifier l'ensemble des données membres associées à l'automate, ainsi que son constructeur et son destructeur. Il est aussi possible d'ajouter un certain nombre de méthodes permettant d'interagir avec ce dernier. Ce système est notamment utilisé pour la communication de données entre plusieurs automates simulant ainsi le système de flots de données associé au modèle HPTS originel.

Factorisation d'action au niveau de l'automate. Lorsqu'un automate est en cours d'exécution, il peut arriver qu'une action doive être répétée systématiquement à chaque pas d'exécution. Pour simplifier la description des automates et surtout éviter à l'utilisateur d'avoir à dupliquer l'action dans chacun de ses états, le langage offre la possibilité de décrire des actions à exécuter en début de pas de temps (champ **before step**) et/ou en fin de pas de temps (champ **after step**), quel que soit l'état dans lequel l'automate se trouve.

Priorité de l'automate. Le champ **priority** permet de décrire la fonction de priorité associée à l'automate. Il s'agit d'une expression C++ quelconque ayant pour contrainte de renvoyer un nombre flottant. Par ce biais, il est possible de décrire tout type de fonction de priorité sans contrainte d'expression. Il est donc possible d'utiliser des fonctions construites dynamiquement, par l'intermédiaire de classes appropriées, corrélant la priorité de l'automate à la raison de sa demande d'exécution.

Description des états. La description des états se fait par l'utilisation du mot clef **state**. Ces états sont nommés et sont décrits par l'intermédiaire de quatre champs :

- La spécification de l'ensemble des ressources utilisées par l'état s'effectue par l'intermédiaire du champ **uses**. Cet ensemble de ressources peut être déclaré dynamiquement à la construction de l'automate, permettant ainsi de décrire des automates génériques en terme de ressources.
- L'action à effectuer lorsqu'une transition provoque l'entrée dans l'état est décrite dans le champ **entry**.
- L'action à exécuter lorsqu'une transition provoque la sortie de état est décrite dans le champ **exit**.
- L'action à exécuter tant que l'automate reste dans l'état est décrite dans le champ **during**.

Les trois actions associées à l'état s'avèrent pratiques en terme d'utilisation. Par exemple, dans le cadre de la saisie d'un objet par un humanoïde virtuel. L'action associée à l'entrée dans l'état permet de commencer le mouvement de préhension et tant que l'automate reste dans l'état, il est possible de remettre à jour la cible associée à la préhension si celle-ci bouge. Enfin, la sortie provoque

l'arrêt du geste. Par ce biais, l'état devient une entité à part entière dotée de sa propre cohérence. Pour l'utilisateur, le compilateur génère des informations permettant de manipuler un état comme un objet, accessible par une méthode portant son nom, lui-même doté de méthodes permettant d'accéder à chacun de ses champs de description.

Description des transitions. La description des transitions s'effectue par l'intermédiaire du mot clef **transition**. Elles sont nommées et sont décrites par l'intermédiaire de cinq champs. Dans les champs permettant de spécifier les états d'origine (champ **origin**) et d'extrémité (champ **extremity**) de la transition, il faut spécifier les états fournis par l'intermédiaire des méthodes générées par le compilateur (portant le nom de l'état). Par ce biais, il devient possible de construire dynamiquement la structure d'un automate ou bien de générer une transition vers un état appartenant à un autre automate. Cette propriété peut, par exemple, permettre de générer des plans d'actions sous la forme d'un chaînage d'automates. Une transition possède aussi une expression stipulant sa condition de franchissement (champ **condition**). Dans certain cas, il peut être utile de faire des traitements lors d'une transition (envoi de signaux, de messages...), pour ce faire le champ **action** est utilisé. Enfin, le degré de préférence (champ **preference**) est spécifié par l'intermédiaire d'une expression C++ renvoyant une valeur numérique. Pour des notions de stabilité, il est préférable de remplir ce champ avec des constantes, cependant, pour ne pas restreindre le pouvoir d'expression, ce choix est laissé à l'utilisateur. Comme pour l'état, une méthode, portant le nom de la transition et renvoyant un objet dont les méthodes permettent d'accéder aux champs de description, est générée par le compilateur.

Réaction aux signaux. Comme dans beaucoup de systèmes, HPTS++ fournit trois type de signaux internes permettant de contrôler l'exécution des automates, un signal de suspension d'exécution, un signal de reprise d'exécution et enfin un signal de terminaison forcée d'exécution. Bien que leur utilisation puisse poser problème pour la cohérence des comportements, ces signaux ont été ajoutés pour ne pas limiter l'utilisateur dans son effort de description. D'autre part, leur utilisation peut s'avérer pratique dans le cas d'automates « utilitaires » ne correspondant pas à un comportement nécessitant des précautions particulières. Des possibilités de réaction à la réception d'un signal sont disponibles d'une part au niveau de l'automate et d'autre part au niveau de l'état (champs **kill, suspend et resume** de l'automate et de l'état). Elles rendent possible la gestion d'une certaine cohérence dans l'exécution, particulièrement au niveau de l'état qui peut, lors de la réception d'un signal, mettre le système dans un état stable.

État initial et final. L'état initial d'un automate est défini par l'intermédiaire du champ **initial state**, avec les mêmes propriétés que l'état d'origine et d'extrémité des transitions. La notion d'état final est plus difficile à définir. Dans le cadre de l'héritage d'automate, un état marqué comme final dans un automate ancêtre, n'est pas forcément final dans l'automate courant. La notion d'état final devrait donc être propre à chaque automate. Mais lors de la description des transitions, les états d'origine et d'extrémité sont fournis sous la forme d'une expression renvoyant une instance d'état (liée à un automate). Rien n'empêche l'utilisateur de créer un automate effectuant des transitions d'un état d'un premier automate, qui peut être localement considéré comme final, vers un état d'un second automate (il s'agit de la notion d'automate agrégat). Dans ce cas, comment définir les états finaux? Pour uniformiser et simplifier la description, les états ne possédant pas de transition sortante sont considérés comme finaux. Cela permet de s'adapter à toutes les situations sans connaissance

à priori sur la forme de l'automate, cependant il s'agit d'une propriété à ne pas omettre lors de la description.

Polymorphisme. L'ensemble des champs de description des automates peut être redéfini lors d'un héritage. Par exemple si l'automate B hérite de l'automate A possédant un état e, l'automate B peut redéfinir cet état e, de manière transparente. Les transitions définies dans l'automate A prendront automatiquement cette nouvelle définition en compte. Au même titre que les états, les transitions peuvent elles aussi être redéfinies. L'utilisation du polymorphisme dans la description des automates, et donc des comportements, possède les mêmes bonnes propriétés que dans la programmation objet en général. Il devient possible, par héritage de modifier le comportement associé à l'automate, de manière transparente.

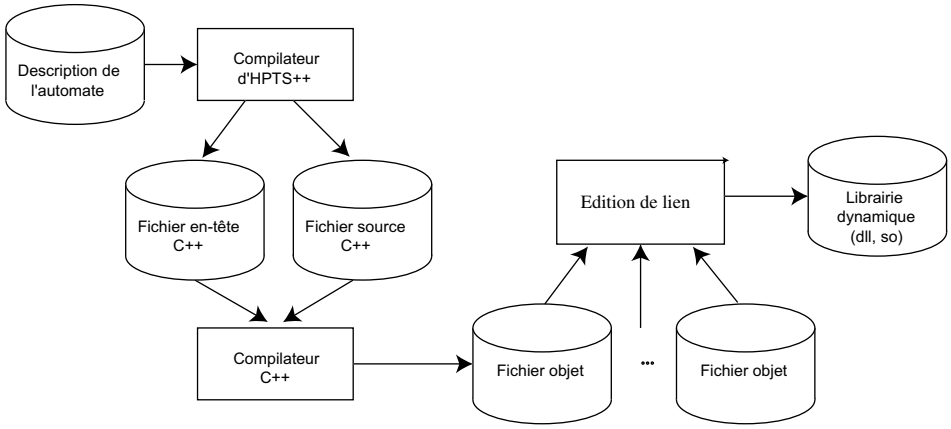


FIG. 3.10 – *Le processus de compilation des automates et la création de bibliothèques de comportement.*

Le couplage de toutes ces notions, permet d'obtenir un langage proposant une grande souplesse pour la description des automates. L'exploitation du C++ en interne offre de grandes facilités d'interfaçage avec les applications, permettant de ne pas restreindre l'utilisateur dans son champ d'expression. La compilation des automates se passe en deux phases (voir fig. 3.10), dans un premier temps, la description de l'automate est compilée par le compilateur d'HPTS++. Ce dernier génère en sortie les fichiers en-tête et source, en C++, associés à l'automate. Pour chaque automate, une classe portant son nom est générée, avec la partie interface décrite dans l'automate et les méthodes associées à la récupération des états et transitions. Ces classes peuvent ensuite être compilée par le compilateur C++. Il est ainsi possible de créer des bibliothèques de comportements génériques. Les comportements contenus dans ces bibliothèques peuvent alors être pris en charge par le moteur d'exécution d'HPTS++, sans aucune information spécifique sur leur déroulement, avec l'assurance d'une exécution cohérente dans la mesure où les ressources ont été correctement spécifiées.

3.3.2 Le contrôleur d'exécution

Les rôles du contrôleur d'exécution sont multiples. Il constitue l'interface entre l'automate et le moteur d'exécution de HPTS++. De fait, il offre toutes les fonctionnalités nécessaires à la gestion des ressources, des signaux et prend aussi en charge l'ordonnancement des automates.

Gestion des ressources. Le contrôleur, sur demande des automates, fournit les instances de ressources qui pourront être utilisées pour l'exclusion mutuelle. Ces ressources, pour une simplicité de description sont identifiées par chaînes de caractères, permettant ainsi aux concepteurs de se mettre d'accord sur leur dénomination avant de commencer la description des comportements. Il est aussi possible de déclarer dynamiquement une nouvelle ressource pour les besoins de l'exécution.

Les automates peuvent effectuer des requêtes sur la possession des ressources et plus particulièrement, savoir quel automate possède une ressource à un instant donné. Cette fonctionnalité permet de gérer une concurrence un peu particulière. Chaque automate peut être considéré comme un spécialiste, possédant la capacité d'évaluer la qualité d'un ou plusieurs résultats qu'il peut fournir. Chaque résultat fourni peut lui-même être associé à une ressource. Une transition allant vers un état possédant une ou plusieurs ressources consiste en une transition stipulant que l'automate peut fournir le ou les résultats qui y sont associés. D'autre part, la qualité du ou des résultats fournis est directement corrélée à la priorité de l'automate. Dès lors, l'automate possédant la ou les ressources est l'automate qui maximise la qualité et qui est donc à même de fournir la ou les meilleures réponses. Cette fonctionnalité permet de concevoir des systèmes de contrôle multi-agents, manipulant des résultats fournis par des agents spécialistes, sans pour autant nécessiter de connaître leur existence de manière explicite. La ressource joue alors le rôle de médiateur entre le système de contrôle et les agents spécialistes, les choix et la gestion de la cohérence étant laissés au mécanisme d'ordonnancement. D'autre part, seuls les agents décident des possibilités de la relâche d'une ressource, cela permet d'assurer une certaine continuité temporelle et de conserver une cohérence d'exécution globale au travers du lien implicite avec le système de contrôle.

Lancement d'un automate. La gestion du lancement d'un automate est aussi à la charge du contrôleur. Comme nous l'avons précisé précédemment, la structure des automates peut être dynamique, tout comme la définition des ressources. Cependant, pour être en mesure d'utiliser l'algorithme d'ordonnancement, il faut connaître cette structure. C'est la raison pour laquelle, lors du lancement d'un automate, le contrôleur fige un certain nombre d'informations, autorisant ainsi les calculs d'ordonnancement et surtout assurant la cohérence de ces calculs. Dans un premier temps, le contrôleur collecte l'ensemble des états constituant l'automate. Pour ce faire, l'état initial est récupéré et par fermeture transitive depuis cet état, l'ensemble des états accessibles ainsi que des transitions les reliant est extrait. Dans le même temps, les états d'origine et d'extrémité des transitions sont figés. Pour chaque état, l'ensemble des ressources utilisées est évalué et figé. Le contrôleur dispose alors d'une structure d'automate fixe autorisant le calcul des dépendances de ressources (voir section 3.2.1.1) et peut alors prendre en charge l'exécution et les calculs d'ordonnancement qui y sont associés.

Gestion des signaux La gestion des signaux (suspension, terminaison, reprise), aussi à la charge du contrôleur, nécessite certaines précautions. Un signal, lors de la description des automates, peut être envoyé à n'importe quel stade d'exécution. Supposons qu'un signal soit envoyé durant une transition. Sa prise en compte immédiate provoquerait une action dans une phase durant laquelle

l'automate cible ne peut réagir. Le système pourrait alors se trouver dans un état instable. Pour éviter cela, les signaux sont traités de manière différée. Lors de leur envoi, il sont conservés par le contrôleur pour être effectivement traités lorsque tous les automates se trouverons dans le même état d'exécution, autrement dit, au début du prochain pas de temps.

Exécution des automates. A chaque pas de temps, le contrôleur gère l'exécution des automates qui lui sont associés. Pour respecter la cohérence dans l'exécution des automates, et simuler un parallélisme synchrone, les actions du controleur sont effectuées dans un ordre précis :

1. Les signaux lancés au pas de temps précédent sont traités. Les méthodes de réaction aux signaux sont appelées pour l'automate cible, en commençant par la réaction au niveau de l'état puis la réaction au niveau de l'automate pour un signal de suspension ou de terminaison. Dans le cas d'un signal de reprise, cet ordre est inversé. Enfin, le contrôleur change état d'exécution de l'automate, en accord avec le type de signal envoyé. Tous les signaux qui sont envoyés durant cette phase, sont traités dans la foulée et ne sont pas différés. Cette propriété permet de propager un signal sur plusieurs automates. Elle s'avère nécessaire dans le cas d'un automate hiérarchique car sa suspension ou son arrêt provoque la suspension ou l'arrêt des ses automates fils.
2. Pour tous les automates, l'action à exécuter en début de pas de temps est exécutée.
3. Le contrôleur collecte les transitions franchissables pour chacun des automates en cours d'exécution.
4. Le calcul d'ordonnancement est effectué sur la base des transitions franchissables et de l'état courant des automates. Les nouveaux états, pour chaque automate, sont alors extraits de la combinaison d'états fournie par le système d'ordonnancement.
5. Pour chaque automate passant une transition, l'action de sortie de l'état est exécutée.
6. Pour chaque automate passant une transition, l'action associée à la transition est exécutée.
7. Pour chaque automate passant une transition, l'action d'entrée dans le nouvel état est exécutée.
8. Pour tous les automates, l'action associée à l'état courant est exécutée.

Cet ordre d'exécution garantit que les automates, se trouvent dans la même phase d'exécution à chaque phase de traitement ; cela permet d'obtenir une exécution cohérente : une action d'entrée dans un état ne sera pas exécutée avant une action de sortie d'un autre état. L'automate relâchant la ressource peut alors effectuer les traitements en conséquences, avant que l'automate ayant pris cette même ressource ne commence ses calculs.

Automates parallèles et automates hiérarchiques. Il n'est pas toujours nécessaire d'utiliser des automates hiérarchiques, particulièrement dans le cas où l'on dispose de moyens de synchronisation automatique, tels que ceux fournis par le mécanisme d'ordonnancement. Cette notion n'est donc pas prise en compte par le contrôleur, cependant, il permet de la gérer. Il existe deux différences entre les automates parallèles et les automates hiérarchiques. D'une part, dans les automates hiérarchiques, les automates fils doivent être exécutés avant l'automate père. D'autre part, l'automate père, à haut niveau, est le seul à être réellement connu du système. Les automates fils sont alors considérés comme des constituants du père. Donc lorsque l'automate père reçoit un signal, ce dernier doit être propagé dans les automates fils. La contrainte de précédence dans l'exécution des automates fils est gérée implicitement par le contrôleur qui exécute les automates dans l'ordre inverse de l'ordre chronologique de leur lancement. La notion de hiérarchie pour sa part, est gérée

par une classe particulière d'automate (les automates hiérarchiques) qui possède la particularité de propager les signaux à ses automates fils.

Concurrence locale. Dans certains cas, il est possible de vouloir gérer une concurrence entre plusieurs automates sur un ensemble de ressources locales. Pour ce faire, HPTS++ accepte la déclaration de contrôleurs locaux aux automates. De cette manière, ce contrôleur possède son propre ensemble de ressources et d'automates, permettant ainsi d'éviter des éventuels conflits de nom de ressources. D'autre part, cette fonctionnalité permet de réduire le nombre de ressources gérées par un contrôleur et donc d'améliorer les performances globales en réduisant la complexité des ordonnancements.

Le langage ainsi que le moteur d'exécution d'HPTS++ ont été pensés pour être souples. Les concepts introduits facilitent la description des comportements des entités tout en mettant l'accent sur la méthodologie de conception. Il est désormais possible de faire des patrons de comportements, devant être complétés par héritage. Cette notion permet d'une part de ne pas réécrire le même code plusieurs fois mais aussi, et surtout, elle permet de décrire et de manipuler des typologies de comportements sans pour autant connaître leur description précise ; ce qui est en harmonie avec le principe d'ordonnancement utilisé dans le contrôleur d'exécution. Il est cependant à noter que son domaine d'utilisation est plus large, car le langage et le moteur d'exécution ne possèdent pas de lien explicite avec les humanoïdes de synthèse.

3.4 Exemple du lecteur, buveur, fumeur

Le modèle qui vient d'être décrit est utilisé pour la modélisation du comportement d'humanoïde virtuels. Les mécanismes de priorité et d'adaptation permettent de gérer des mouvements complexes, traduisant un mélange automatique de plusieurs comportements, dans un soucis de conservation de la cohérence globale, avec un minimum de description. Pour montrer les capacités de ce système, nous allons nous intéresser à l'exemple du lecteur, buveur, fumeur [LD01]. Une personne est assise à la terrasse d'un café. Elle lit un journal, en fumant une cigarette et en buvant un café. Cet exemple est constitué de trois comportements indépendants, qui fonctionnent en parallèle avec des contraintes sur la gestion de l'attention visuelle ainsi que sur l'utilisation des mains pour la manipulation des objets. Avant de décrire la manière dont cet exemple peut être reproduit avec HPTS++, nous allons nous intéresser au mode de connexion entre le système et l'humanoïde virtuel.

3.4.1 Connexion au modèle humanoïde

La connexion entre les comportements et l'animation de l'humanoïde virtuel, pour être aisée, nécessite un certain nombre de prérequis. La réalisation des différentes démonstrations liées à cette thèse repose sur un certain nombre de travaux effectués au sein de l'équipe SIAMES. Parmi ceux-ci, l'humanoïde développé par Stéphane Ménardais [Men03] qui offre un certain nombre de fonctionnalités utiles pour l'animation comportementale. Ce système offre à l'utilisateur une collection d'actions élémentaires permettant d'effectuer des calculs de cinématique inverse sur les bras, les jambes, tout en mélangeant ces calculs avec des captures de mouvement pour augmenter le réalisme de l'animation. Dans ce cadre, Nicolas Courty [Cou02] a développé, à partir de techniques d'animation référencée vision, des actions permettant de gérer l'animation de la chaîne articulaire allant du bassin jusqu'aux yeux pour simuler le comportement de perception.

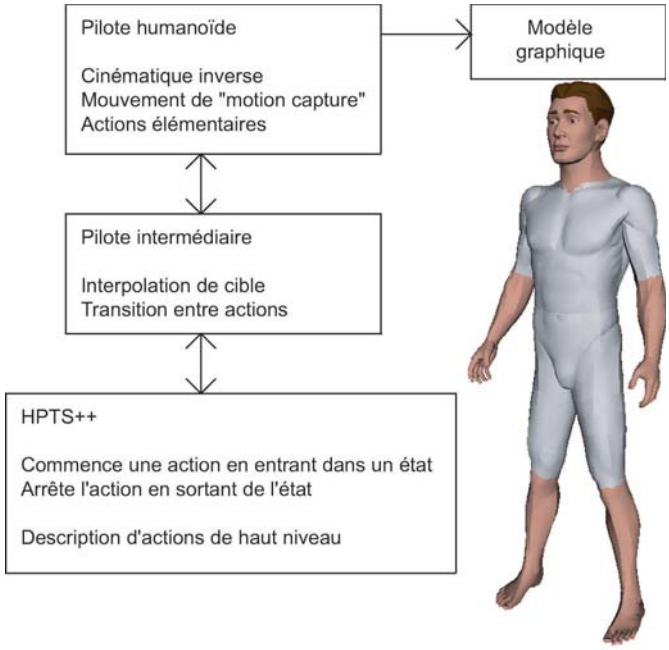


FIG. 3.11 – *Connexion d'HPTS++ au modèle humanoïde.*

Pilote intermédiaire. S'appuyant sur ces travaux, la connexion du modèle HPTS++ avec l'animation de l'humanoïde exploite donc la notion d'action élémentaire. Un certain nombre d'états sont définis à l'intérieur des automates pour traduire le comportement en animation. Le principe consiste, lors de la description des états de l'automate, à commencer une ou plusieurs actions élémentaires lors de l'entrée dans l'état et à arrêter systématiquement ces actions lors de la sortie d'un état. De cette manière, un état constitue une entité possédant sa cohérence propre, remettant le système dans une configuration stable à chaque sortie d'état. Cette caractéristique de description permet de générer facilement des transitions d'adaptation libérant des ressources et donc arrêtant automatiquement les mouvements associés à l'utilisation des ressources. Cependant, l'utilisation d'une telle technique requiert la présence d'un pilote d'actions intermédiaires (voir fig. 3.11) permettant de gérer la continuité des mouvements. Deux cas peuvent se présenter :

- Le comportement se déroule normalement et donc en passant d'un état à un autre commence et arrête la même action durant le même pas de temps.
- Le comportement relâche une ressource associée à un mouvement, il arrête donc ce mouvement, mais un autre automate ayant récupéré cette ressource relance le mouvement.

Dans ces deux cas, l'action n'est pas réellement arrêtée mais continuée, le pilote d'actions intermédiaire permet d'amortir l'arrêt du mouvement en détectant un arrêt directement suivi d'un démarrage. Dans ce type de configuration, ce pilote intermédiaire continue le même mouvement, en créant une interpolation entre les différentes cibles utilisées, rendant peu perceptibles ces changements d'état.

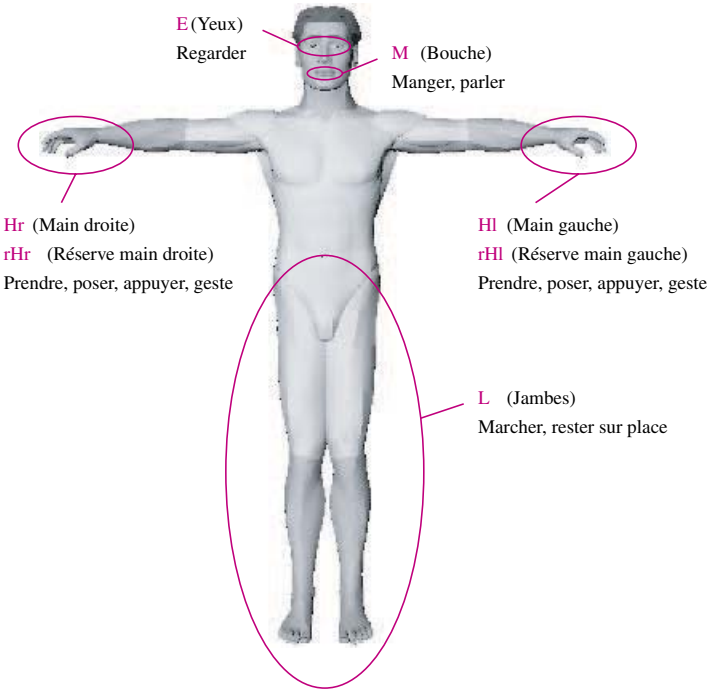


FIG. 3.12 – *Les ressources utilisées pour décrire un humanoïde.*

Ressources associées aux humanoïdes. La figure 3.12 décrit l'ensemble des ressources utilisées pour la synchronisation des comportements, en donnant quelques exemples de comportements associés. Ces ressources décrivent les parties du corps pouvant être manipulées indépendamment l'une de l'autre. Les yeux, la bouche et les jambes sont chacun représentés par l'intermédiaire d'une seule ressource alors que les mains sont représentées en utilisant deux ressources. Ceci s'explique par le rôle particulier des mains qui permettent de manipuler des objets. Elles ont donc besoin de la gestion d'une cohérence plus forte que pour les yeux par exemple. Alors que les yeux peuvent changer très rapidement de cible, les mains nécessitent parfois un comportement particulier pour leur libération, comme reposer un objet précédemment saisi par exemple. La seconde ressource associée, préfixée par "r", stipule la réservation de cette ressource. Dans ce type de configuration, un état possède une signification particulière en fonction de sa manière d'utiliser le couple de ressources. Prenons l'exemple des ressources *Hr* (main droite) et *rHr* (réservation de la main droite) :

- un état prenant la ressource *rHr* effectue une demande de réservation de la ressource *Hr* et donc indirectement une demande de libération de la ressource par un autre automate.
- un état prenant les ressources *rHr* et *Hr* est un état utilisant cette main pour manipuler un objet.
- un état utilisant la ressource *Hr* est un état libérant la main droite pour qu'elle puisse être utilisée par un autre comportement.

L'utilisation de cette convention est liée intrinsèquement à la nature de l'algorithme d'ordonnancement qui ne peut agir que sur la prochaine transition d'un automate. Le respect de cette convention permet d'obtenir beaucoup de souplesse dans le déroulement des comportements en terme d'adaptation automatique. Dans le cas d'un mélange de comportements ne la respectant pas avec un

comportement la respectant, le comportement de l'entité ne deviendra pas pour autant incohérent. Cependant, l'algorithme d'ordonnancement ne pourra pas forcer un comportement donné à libérer la ressource. Cela se traduira donc uniquement par une limitation des possibilités d'adaptation.

3.4.2 Description des comportements

L'exemple du lecteur, buveur, fumeur a été bâti en exploitant les propriétés du système d'ordonnancement. Le comportement global de l'humanoïde est décrit par l'intermédiaire de quatre automates représentés par la figure 3.13. Les automates sont en concurrence sur un ensemble de six ressources respectant la nomenclature définie au paragraphe précédent. Dans les schémas, les ressources (*H, rH*) doivent être remplacées par (*Hr, rHr*) dans le cas d'une manipulation avec la main droite et (*Hl, rHl*) dans le cas d'une manipulation avec la main gauche. Il est à noter que dans l'implémentation de l'exemple, ces deux ressources sont transmises en paramètre du constructeur de l'automate concerné. Les transitions sont étiquetées avec leur condition ainsi que le degré de préférence qui leur est associé, alors que les états sont étiquetés avec leur rôle ainsi que l'ensemble des ressources qu'ils utilisent. Une fois les automates décrits, il faut leur associer des fonctions de priorité traduisant l'évolution de leur importance en fonction du temps. Le système d'ordonnancement pourra, par la suite, se charger de reproduire le comportement émergeant automatiquement.

Boire et fumer. Cet automate générique décrit le comportement de boire ou de fumer. Il se déroule en trois phases :

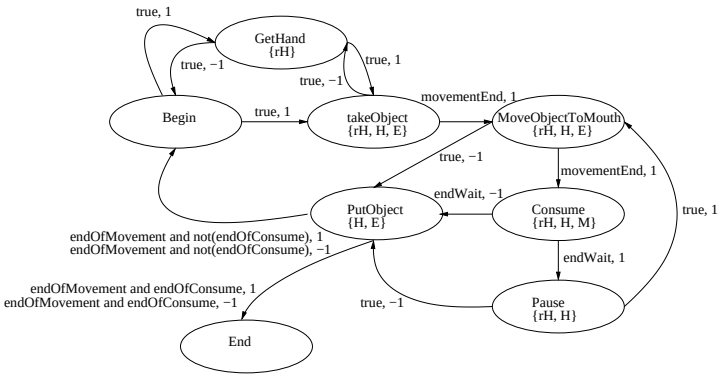
1. saisir l'objet d'intérêt en en essayant de prendre la ressource de réservation de la main.
2. garder l'objet dans la main et le déplacer régulièrement vers la bouche.
3. reposer l'objet pour libérer la ressource de la main.

Les phases 2 et 3 sont connectées par l'intermédiaire de transitions possédant des degrés de préférence négatifs. Ces degrés de préférence traduisent la possibilité de relâcher la ressource de la main si un comportement plus prioritaire en a besoin ou si le comportement est inhibé.

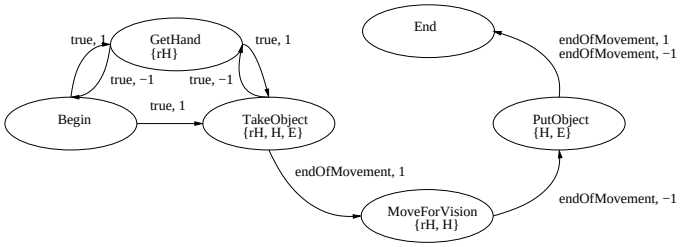
Lire la feuille de papier. Ce comportement consiste en deux comportements se déroulant en parallèle, d'une part la manipulation de la feuille de papier avec l'une des mains durant la lecture et d'autre part, le comportement de lecture en tant que tel. Dans le comportement de lecture, les degrés de préférence sont utilisés pour traduire le coût d'une interruption du comportement en fonction du découpage du texte (phrase, paragraphe, chapitre).

Déplacement de la feuille de papier. Cet automate est utilisé pour déplacer la feuille de papier sur le coté lorsque la lecture est terminée, puis pour saisir une nouvelle feuille et la déposer devant le lecteur.

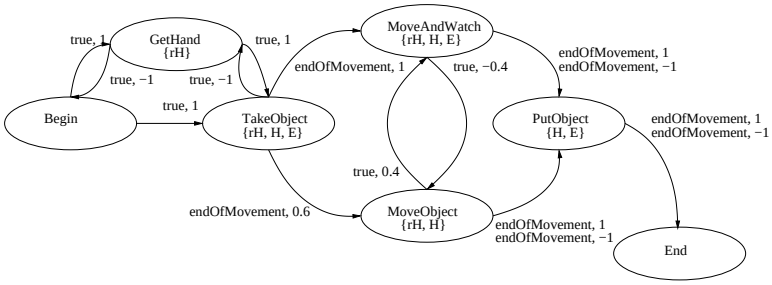
En observant les comportements de boire, de fumer, de manipulation de la feuille de papier pendant la lecture et de déplacement de la feuille de papier, il est possible de remarquer que ces comportements sont bâtis suivant la même structure. Dans un premier temps, l'objet d'intérêt est saisi, puis il est manipulé et reposé. Il s'agit donc en réalité de comportements de même nature : des comportements de manipulation d'objet avec une seule main. Lors de leur description les notions d'héritage offertes par HPTS++ ont été exploitées. Une classe d'automate de manipulation peut être décrite comme un patron de comportement. Elle comporte la phase de saisie de l'objet ainsi que



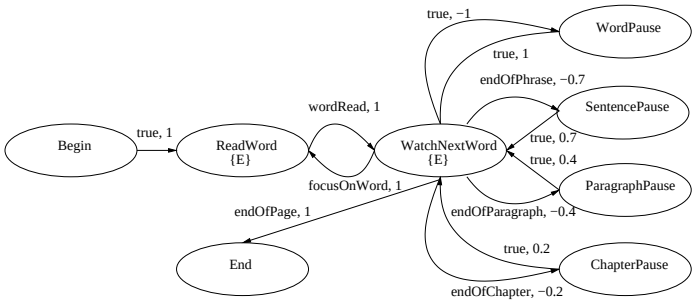
Automate permettant de boire ou fumer.



Automate de manipulation de la feuille de papier.



Automate permettant de déplacer la feuille de papier.



Automate de lecture.

FIG. 3.13 – *Les quatre automates principaux de la simulation du lecteur, buveur, fumeur.*

la phase consistant à reposer l'objet (voir fig. 3.14). Ensuite, par héritage, les quatre comportements, boire/fumer, manipuler la feuille pendant la lecture et déplacer la feuille, ont été définis en ajoutant à l'automate de manipulation la description de la partie qui leur est spécifique. Cette notion d'héritage permet d'une part de définir des grandes classes de comportement, liées à leur nature et d'autre part de réduire le coût de description en exploitant la notion de patron d'automate.

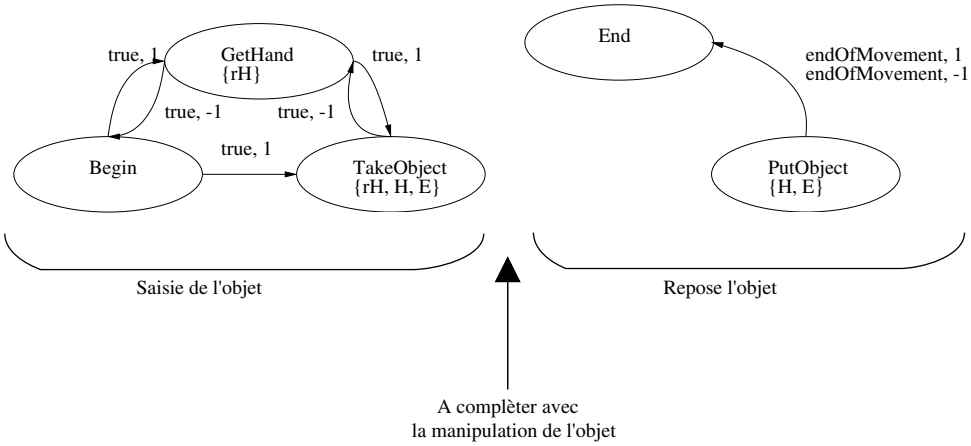


FIG. 3.14 – *Patron d'automate de manipulation d'objet à une main.*

Définition des priorités. Les fonctions de priorités associées aux automates sont dépendantes de paramètres différents : l'envie de lire, le besoin de nicotine et la soif. Dans l'exemple, le comportement de lecture possède une priorité constante. Les comportements de boire et fumer ont une priorité variable, associée au paramètres physiologiques de soif et de besoin de nicotine, dont l'évolution temporelle est définie comme suit :

$$\begin{cases} p(t) = p(t - dt) + dp_1 \times dt \ si \ non(consommation) \\ p(t) = p(t - dt) - dp_2 \times dt \ si \ consommation \end{cases} \tag{3.7}$$

où dp_1 représente le gain par défaut du paramètre et dp_2 le gain lors de la consommation. Pour les besoins de l'exemple, les priorités ont été réglées comme suit :

- le déplacement des feuilles et le comportement de lecture ont une priorité constante de 2.0 ;
- la manipulation de la feuille avec la main gauche ou droite possède une priorité de 1.5 ;
- la priorité du comportement de boire a été paramétrée avec $dp_1 = 0.02$ et $dp_2 = 0.08$;
- La priorité du comportement de fumer a été paramétrée avec $dp_1 = 0.05$ et $dp_2 = 0.07$.

Une fois les comportements et les fonctions de priorités définis, l'algorithme d'ordonnancement peut prendre en charge l'exécution des automates. La figure 3.15 décrit l'activité des comportements, définie par le passage dans un état utilisant au moins une ressource, ainsi que l'évolution des priorités de boire et fumer durant la simulation. Les comportements, décrits indépendamment les uns des autres, s'exécutent en parallèle dans le mesure où l'utilisation des ressources le permet. Dans les cas de conflits, le système d'ordonnancement force certaines transitions pour libérer des ressources utiles à des comportements plus prioritaires. Par exemple, au temps 30-40, le comportement de fumer est

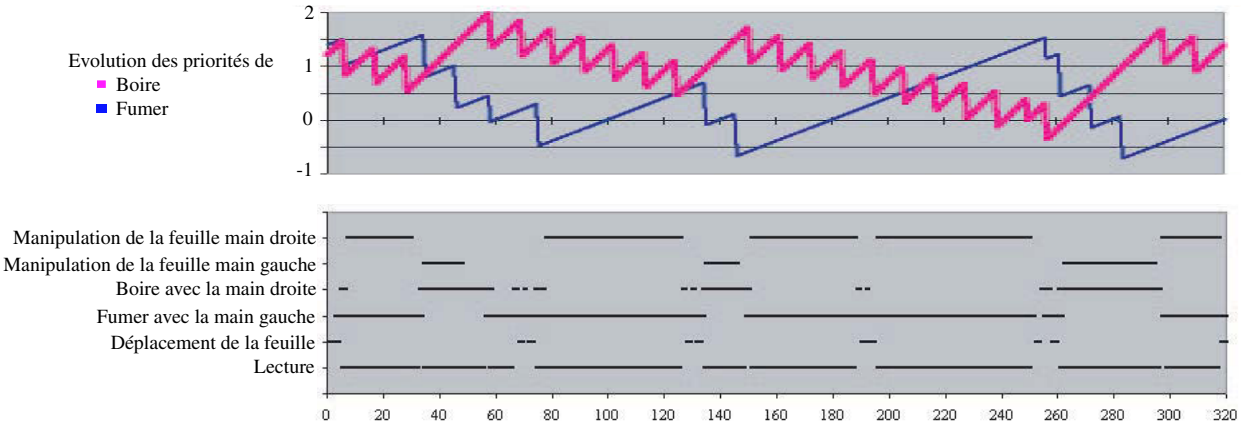


FIG. 3.15 – *Activité des comportements lors de la simulation*

interrompu pour laisser la ressource de la main gauche au comportement de manipulation de la feuille de papier qui possède une priorité plus grande. Cette adaptation s'effectue sans communication entre les comportements, assurant ainsi leur indépendance en terme de description. Le comportement émergeant ainsi généré est cohérent (voir fig. 3.16 p. 182) sans pour autant avoir nécessité de description explicite de la concurrence des sous comportements le constituant. Cette concurrence est gérée implicitement par l'utilisation des ressources, des degrés de préférence et des fonctions de priorité.

Conclusion

Le modèle que nous venons de proposer s'attache à offrir des mécanismes permettant de gérer les comportements de façon abstraite, tant dans leur description que dans leur réalisation. La modélisation des comportements se base sur la notion de ressources permettant de décrire les exclusions mutuelles sur ce qu'ils manipulent. Une fois ces ressources explicitées, il est possible de décrire toute forme de comportement pouvant les manipuler. La description ne nécessite donc plus la connaissance des autres comportements décrits dans la mesure où le mécanisme d'ordonnancement gère les conflits et évite les inter-blocages lors de l'exécution. La notion d'adaptation permet de décrire un comportement de manière atomique. Cette description contient à la fois le déroulement normal et les possibilités offertes lors des cas de conflits, sans pour autant expliciter avec qui ces conflits peuvent intervenir, mais seulement comment ils peuvent être gérés automatiquement. Le comportement devient alors une brique décrivant toutes les éventualités, sans en décrire toutes les causes. Enfin, les priorités permettent de manipuler le comportement de manière abstraite, à l'image du marionnettiste et de sa marionnette. Cette priorité devient alors le fil à tirer pour faire émerger un comportement ou au contraire le stopper, mais toujours dans un souci de cohérence, et donc sans utiliser de mécanismes de préemption, nuisibles à la fois au réalisme et à la stabilité du système.

Au travers du langage et du moteur d'exécution qui ont été décrits, le système global s'attache à être ouvert. Les notions de programmation objet qui ont été incluses permettent une connexion aisée avec toute sorte de systèmes ainsi que la manipulation abstraite de classes de comportement. Le système global rend possible et facilite la connexion de comportements venant de l'extérieur tout

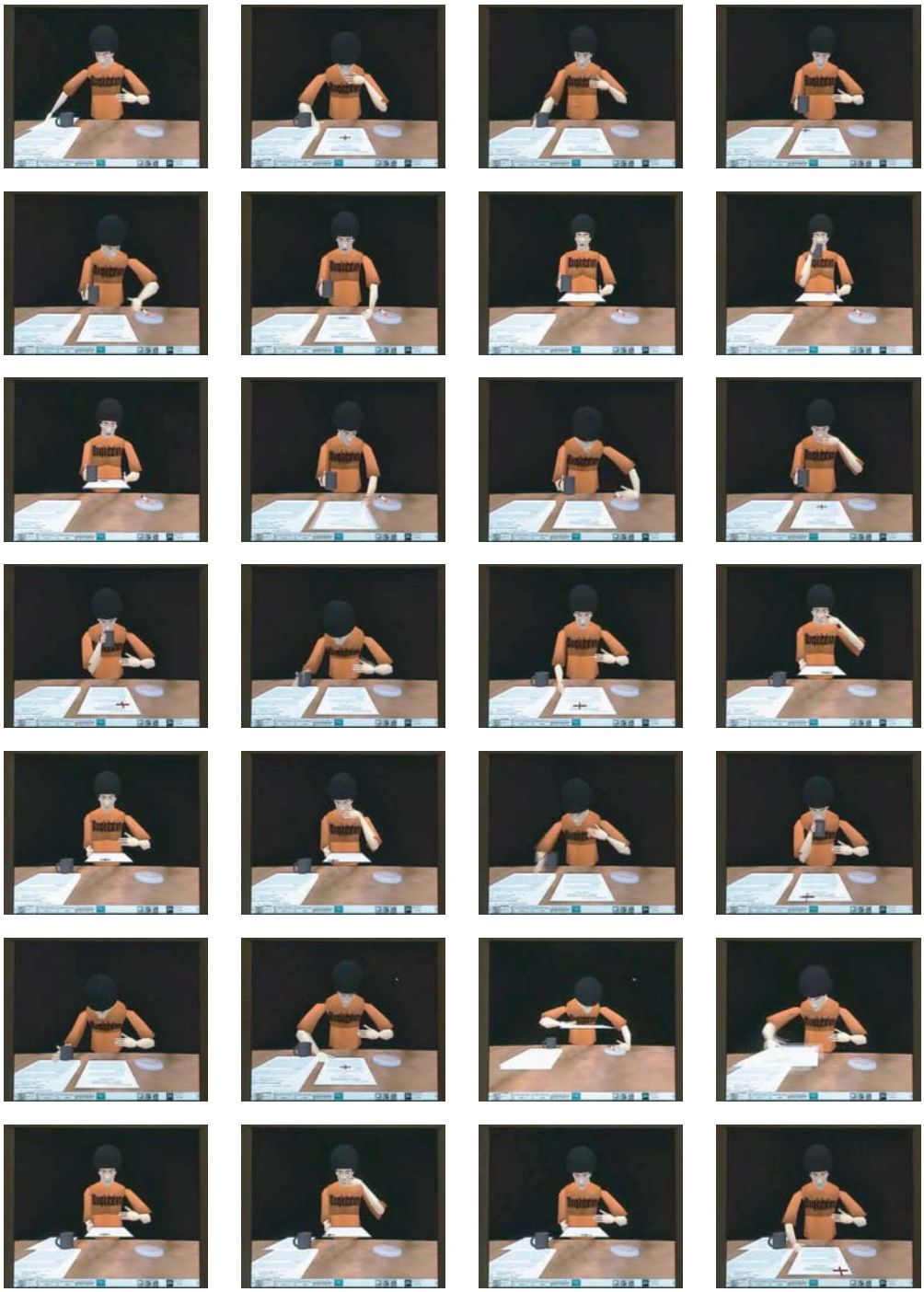


FIG. 3.16 – *Un humanoïde virtuel qui lit, boit et fume.*

en assurant la cohérence de l'exécution. Il devient alors possible de mélanger des comportements réflexes et purement réactifs avec des comportements issus d'un mécanisme de planification sans aucune précaution. Si ce mécanisme accepte de planifier des actions en parallèle, les fonctionnalités

offertes par HPTS++ permettent de gérer automatiquement leur réalisation en prenant en compte d'éventuels conflits sur des ressources corporelles qui ne sont que rarement décrits dans les modèles de raisonnement ou assimilés.

Chapitre 4

Comportements cognitifs

Lorsque quelqu'un adopte un comportement dans une situation donnée, ce comportement peut être qualifié de réactif ou de pro-actif. Un comportement réactif est adopté en réaction à une situation perçue. Un comportement pro-actif, est un comportement adopté consciemment. Il est une brique considérée comme nécessaire à la réalisation d'un but fixé. Ces comportements sont donc la conséquence d'un raisonnement qui a été mené au préalable. Le raisonnement s'effectue sur un monde symbolique, permettant de se projeter dans le futur pour analyser les conséquences des actions qui peuvent être effectuées. Cela rend possible la recherche d'un enchaînement d'actions permettant de satisfaire un but. Cependant cette suite d'actions interagit avec un environnement dynamique dont nous n'avons qu'une perception partielle, les plans alors élaborés font l'objet d'adaptations face à des situations que l'on peut qualifier d'imprévues. D'un autre coté, les comportements possibles à adopter sont en grande partie déterminés par les possibilités d'interaction que l'environnement nous offre. Pour doter les humanoïdes de tels comportements, il semble alors nécessaire de leur fournir une représentation abstraite de l'environnement. Il s'agit de l'approche adoptée par l'intelligence artificielle, en corrélation avec les abstractions que nous manipulons. Cependant, ces approches prennent rarement en compte la dynamique de l'environnement. Un plan complètement instancié, lorsque l'on se trouve dans un environnement dynamique peuplé de plusieurs agents avec de buts différents, peut rarement être complètement réalisé tel quel et nécessite des adaptations.

Dans le chapitre précédent, nous nous sommes intéressés à la modélisation de tâches ne nécessitant pas de représentation explicite des connaissances. Ces tâches permettent de traduire des comportements d'interaction de l'agent, en les connectant directement à l'animation de l'humanoïde de synthèse. Elles constituent désormais des sortes de briques unitaires, que l'agent va pouvoir utiliser en vue de satisfaire un but fixé. Dans ce chapitre, nous proposons d'aborder le problème de la sélection d'action et de la représentation des connaissances de l'agent. La représentation des connaissances est à la base d'un comportement cognitif, permettant d'enchaîner de manière cohérente, des actions pour réaliser un but fixé. Dans ce cadre, nous proposons le langage BCOOL [1], un langage orienté objet permettant de décrire les connaissances d'un agent autonome en le dotant d'une représentation abstraite de l'environnement. Il s'inspire des théories écologiques de Gibson [Gib86] en permettant de ne plus centrer la description sur l'agent mais sur les opportunités que l'environnement lui offre. L'agent, doté d'une représentation des connaissances, peut alors choisir ses actions. Ces choix sont orientés par la volonté de satisfaire un but mais doivent prendre en compte la dynamique de l'environnement, ainsi que les opportunités qu'il offre. Il s'agit donc là d'une sorte

1. Behavioral and Cognitive Object Oriented Language

de « planification réactive », nécessaire lorsque plusieurs agents peuvent interagir dans un même environnement. Pour ce faire, nous proposons un mécanisme de sélection d'actions, associé au langage BCOOL. Il manipule des actions effectives et perceptives, en exhibant une certaine forme de réactivité par rapport aux changements perçus de l'environnement.

4.1 Approche générale.

4.1.1 Motivations

Il semble difficile d'imaginer accomplir un but sans posséder la symbolique suffisante pour, d'une part, décrire ce but et, d'autre part, choisir un enchaînement d'actions cohérentes permettant de le satisfaire. Les approches classiques de l'intelligence artificielle offrent un certain nombre de langages et de moteurs d'inférence permettant de modéliser des aspects de l'intelligence et du raisonnement. Les travaux dans ce domaine ont beaucoup porté sur la puissance de résolution des moteurs de planification et leur complétude, au détriment des moyens de description de la connaissance et de son lien avec le comportement d'un agent autonome. En terme d'animation comportementale, le raisonnement est une notion dont l'utilisation est limitée, ne serait-ce que par le besoin d'un rendu temps-réel. Il n'est pas nécessaire de posséder un agent capable de jouer aux échecs (et surtout de gagner) pour lui permettre de se comporter de façon cohérente dans son environnement.

Il existe peu de systèmes dont l'orientation première est la modélisation des connaissances en vue de générer des comportements cohérents. Les moyens de description offerts aux utilisateurs sont généralement limités, ils se traduisent par l'utilisation de propriétés sur l'environnement ainsi qu'un typage simple des objets pouvant le peupler. La description d'un environnement en vue de générer des comportements peut alors devenir fastidieuse. L'inclusion d'une nouvelle typologie d'objets offrant de nouvelles formes d'interactions nécessite le plus souvent une nouvelle description de l'objet, alors que dans les descriptions précédentes, des objets possédant des particularités similaires ont pu être déjà décrits. Cela soulève deux problèmes : un problème de qualité logicielle et un problème de représentation des connaissances. Le problème de qualité logicielle se situe dans la description de ce qui a déjà été décrit. Le problème de représentation des connaissances se situe dans le manque d'abstraction dans la classification des typologies d'objets. Ces considérations ont motivé le travail effectué sur BCOOL. Le but est d'offrir à la personne qui décrit un monde « cognitif » pour les agents autonomes, des moyens de description directement inspirés de la modélisation objet. Cela permet d'une part, d'utiliser des mécanismes de modélisation répandus, et d'autre part, d'introduire automatiquement des niveaux d'abstraction dans la description de la connaissance. Enfin, ce langage tente d'offrir un lien très fort entre le monde cognitif et les comportements qui peuvent être adoptés de manière effective par l'agent. Cela permet de simplifier la mise en correspondance entre les actions modélisées pour le raisonnement et les actions réelles correspondantes.

La deuxième caractéristique à prendre en compte lorsqu'un agent évolue et sélectionne des actions en vue de réaliser un but est la dynamique du monde. L'agent n'étant potentiellement pas seul à évoluer dans l'environnement, de son point de vue, ce même environnement évolue de manière imprévisible. Les conséquences sont doubles. D'une part, il est peu probable de pouvoir générer un plan d'actions et le suivre sans rencontrer d'incohérence par rapport à ce qui était prévu. D'autre part, la connaissance de l'agent évolue constamment et le monde peut lui offrir des opportunités

d'actions qui étaient jusqu'alors imprévisibles. L'agent devrait donc être à même de prendre cette nouvelle connaissance en compte pour éventuellement changer la course de ses actions, en exploitant ces nouvelles possibilités. Une dernière caractéristique est à prendre en considération, il s'agit de la confiance dans la connaissance. Dans la mesure où l'évolution du monde est imprévisible, l'agent doit être à même de prendre en compte l'incertitude que cela provoque. Sa confiance dans la connaissance peut donc évoluer au cours du temps, pour finalement provoquer des actions perceptives, nécessaires à l'acquisition d'informations en vue de satisfaire le but fixé. Il s'agit là des considérations qui ont conduit au choix du mécanisme de sélection d'action associé à BCOOL. Son but est de sélectionner des actions de manière incrémentale, pour être à même de prendre en compte la dynamique du monde, et donc de remettre en cause, à chaque sélection, les choix effectués jusqu'alors.

4.1.2 Architecture du système

Avant de commencer la description détaillée de BCOOL, nous allons nous intéresser à l'architecture globale du système. BCOOL se scinde en deux parties : une partie compilation et génération et un mécanisme de sélection d'actions. La figure 4.1 présente les différentes briques de haut-niveau utilisées dans la conception des agents autonomes.

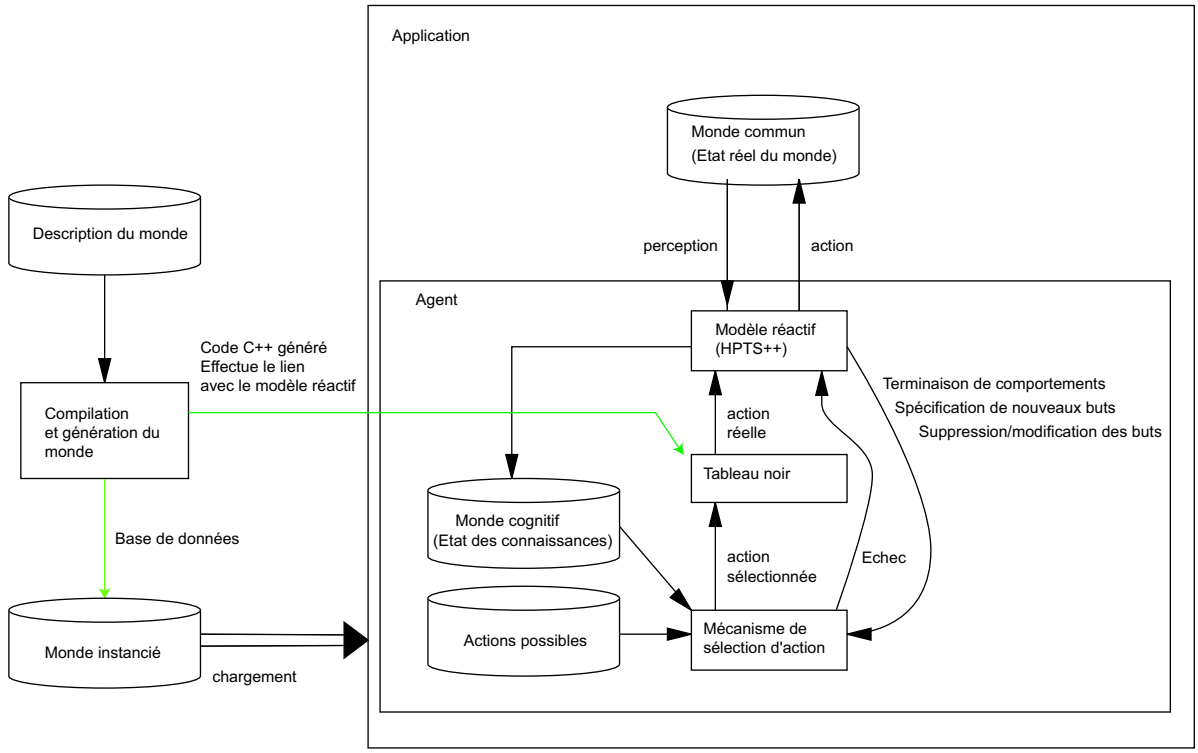


FIG. 4.1 – *Architecture d'utilisation de BCOOL.*

La description des mondes BCOOL se fait par l'intermédiaire d'un langage dédié qui va être présenté dans la section suivante. Cette description passe au travers d'un processus de compilation pour, au final, générer une base de données décrivant le monde ainsi que les possibilités d'interac-

tions offertes aux agents. Cette base de données est orientée raisonnement, elle ne contient que des informations qui permettent la représentation abstraite du monde ainsi que différents paramétrages décrivant les actions. Comme nous le disions dans l'introduction, BCOOL possède aussi un lien très fort avec le modèle réactif permettant d'exécuter les actions. Ce lien est décrit à l'intérieur du langage. Lors de la phase de compilation, un tableau noir est généré ; son rôle est de faire la jonction entre les actions conceptuelles issues du raisonnement et les actions réelles exécutables par le moteur d'HPTS++ en charge de la gestion des comportements réactifs. Nous reviendrons, lors de la description de l'architecture, sur son fonctionnement.

Au chargement de l'application, la base de données représentant le monde BCOOL est chargée. Elle permet l'initialisation du monde commun. Il s'agit d'une base de connaissances, omnisciente, qui regroupe les effets des actions de tous les humanoïdes. Elle peut ensuite être utilisée, comme base de référence, pour les actions de perceptions qui synchronisent alors l'état des connaissances de l'agent avec l'état réel du monde commun.

L'agent pour sa part manipule deux bases de données locales, l'une contenant l'ensemble de ses connaissances (le monde cognitif) et l'autre l'ensemble des interactions qui lui sont offertes par l'environnement. L'état de ses connaissances est cohérent mais n'est pas à jour par rapport au monde commun. Il n'est donc pas omniscient, mais possède sa propre connaissance ainsi qu'une certaine confiance dans celle-ci. L'une des propriétés du mécanisme de sélection d'actions est de pouvoir exploiter une connaissance imparfaite, et de générer, éventuellement, les actions permettant d'acquérir des informations. Il utilise les deux bases de données pour choisir, en fonction des buts ainsi que de l'état des connaissances de l'agent, une action à réaliser. Cette action est alors transformée en comportement par le tableau noir qui le greffe sur le moteur d'exécution d'HPTS++. Le comportement ainsi généré, en concurrence potentielle avec les autres comportements que peut posséder l'agent, est assuré de s'exécuter de façon cohérente, grâce aux mécanismes de synchronisation et d'adaptation. Ces comportements, lors de leur exécution, sont responsables de :

- l'acquisition des connaissances par l'intermédiaire de la base de données du monde commun,
- la mise à jour de la base de donnée représentant la connaissance de l'agent,
- la mise à jour de la base de données représentant le monde commun, pour spécifier les effets des actions,
- des différentes animations traduisant le comportement de l'agent.

La partie réactive du comportement de l'agent est aussi en charge de la gestion de l'évolution du degré de confiance qu'il accorde à ses croyances. Les mécanismes offerts par BCOOL ne prennent pas cette évolution en charge, laissant à l'agent le soin de définir son propre protocole de confiance. Cependant, le mécanisme de sélection d'actions prendra automatiquement en compte les mises à jours lors de ses choix.

4.2 BCOOL : Behavioural and Cognitive Object Oriented Language

BCOOL est un langage de description permettant de modéliser la connaissance des agents. Il offre la possibilité de décrire un monde, peuplé d'objets et d'agents, ainsi que les relations qui peuvent exister entre eux. Il s'inspire des théories de Gibson sur les *affordances*. L'idée, de manière similaire à la notion de *Smart object* introduite par Kallmann [KT02], consiste à décrire des objets fournissant eux-mêmes à l'agent les comportements d'interaction. L'expertise est alors déportée dans les objets considérant l'agent comme une sorte de coquille sur laquelle il est possible de greffer des comportements. Cependant, contrairement à l'approche de Kallmann, les objets ne peuvent

pas prendre le contrôle d'un agent. Les possibilités d'interaction sont proposées, sous une forme permettant de les intégrer dans le processus de raisonnement. De cette manière, un comportement d'interaction ne sera adopté que si sa réalisation favorise la réalisation d'un but que l'agent s'est fixé. Après une discussion sur le choix de la modélisation objet pour BCOOL, nous présenterons les éléments du langage avant de montrer comment une base de donnée contenant les informations utiles à la planification est générée.

4.2.1 Représentation des connaissances et modélisation objet

La modélisation objet a fait ses premiers pas en informatique dans les années 1980 avec la naissance du langage SMALLTALK. L'idée intrinsèque, liée à la naissance de ce mode de programmation, était d'offrir un formalisme permettant aux programmeurs de raisonner de manière plus conceptuelle lors de la création de programme.

La notion de classe d'objet permet de décrire des entités conceptuelles ayant une existence physique ou non. La notion d'héritage qui y est associée permet de décrire des hiérarchies de concepts allant de la notion la plus abstraite vers la notion la plus concrète. Comparativement à la représentation que l'on peut se faire du monde qui nous entoure, cette notion de concept et de hiérarchie de concepts semble justifiée. Par exemple, une femme et un homme sont avant tout des être humains. Le concept d'être humain se décline donc en deux grandes classes : la femme et l'homme. Les exemples de ce type sont nombreux et ont motivés la création d'un langage orienté objet pour la modélisation des connaissances des agents. L'utilisation de ce type de modélisation offre deux avantages : d'une part en terme de programmation et d'autre part en terme de connaissance implicite fournie à l'agent. En terme de programmation, la modélisation objet est reconnue, très utilisée, et possède la bonne propriété de disposer de méthodes d'analyse bien formalisée telles qu'UML[2]. L'organisation de la description, sous la forme de classes d'objets et de hiérarchie de concepts, fournit à l'agent une représentation des connaissances calquée sur celle du concepteur. Par ce biais, le concepteur du monde et l'agent, utilisant le monde conçu, possèdent la même abstraction. A longue échéance, cette notion peut s'avérer plus qu'utile pour être en mesure de communiquer avec l'agent.

Les méthodes associées à une classe d'objet représentent l'ensemble des fonctionnalités offertes à l'utilisateur. Elles décrivent donc les possibilités d'interaction avec l'objet. La notion de polymorphisme couplée à la notion d'héritage, permet de spécialiser des méthodes en changeant leur comportement le long de la hiérarchie. Les interactions offertes peuvent donc changer en fonction de la classe de l'objet, permettant tout de même de manipuler cet objet au niveau d'abstraction le plus haut tout en utilisant la méthode redéfinie dans les classes les plus spécialisées. Cette notion peut être directement assimilée à la notion d'interaction physique, ce qui est l'une des caractéristiques de BCOOL. Un objet est décrit sous la forme d'une classe le caractérisant, de propriétés le définissant et d'interactions offertes à l'agent. Cette notion d'interaction permet de décrire des comportements que l'agent peut adopter vis-à-vis de cet objet. La notion de polymorphisme est alors utilisée pour spécialiser les comportements d'interaction en fonction de la classe réelle de l'objet. En prenant un exemple très schématique, on ne prend pas de la même manière un verre et une casserole, même si l'on peut considérer à un niveau d'abstraction supérieur que ces deux objets sont manipulables.

Le langage BCOOL met en œuvre ces concepts, pour offrir un langage haut-niveau permettant de décrire un monde ainsi que les objets qui le peuplent.

2. Universal Modelling Language

4.2.2 Structure du langage

Le langage se scinde en deux parties : la description du monde conceptuel et la description d'un monde réel. La description du monde conceptuel consiste à décrire l'ensemble des catégories d'objets qui peuvent peupler un monde ainsi que les possibilités d'interaction qu'ils offrent aux agents. La description du monde réel permet de décrire un monde par l'intermédiaire des intances d'objets qui le peuplent. C'est à partir de la description du monde réel qu'un agent va savoir quelles sont les actions qu'il va pouvoir effectuer et quels sont les objets avec lesquels il va pouvoir interagir.

Notations. Dans la suite de nos explications, nous allons présenter des parties de la grammaire de BCOOL. La description s'effectue de la manière suivante : les terminaux sont représentés en gras, les non-terminaux en style normal. Les constructions suivantes seront employées :

- $A*$ pour une répétition de A entre 0 et n fois,
- $A+$ pour une répétition de A entre 1 et n fois,
- { } pour un bloc optionnel,
- pour grouper,
- | pour l'opérateur « Ou ».

D'autre part, les non terminaux *Type* et *Identifiant* représenterons des suites de lettres décrivant un type ou un identifiant fourni par l'utilisateur du langage. Ils sont différenciés pour des notions de compréhension mais sont équivalents dans leur définition. Enfin, le non-terminal *cpp* qualifie du code C++ natif.

4.2.2.1 Monde conceptuel, faits et connaissance.

Le langage BCOOL permet la manipulation de deux sortes de faits permettant de représenter l'état du monde : les propriétés et les relations. Pour chacun de ces faits, un opérateur permettant de travailler sur la connaissance de leur état est défini.

Les propriétés. Les propriétés sont des faits atomiques pouvant prendre deux valeurs : vrai ou faux. La syntaxe les décrivant est la suivante :

Propriété ::= **property** *Identifiant* { = **true** | **false** }

Ces propriétés permettent de décrire des états possibles pour des objets. Par exemple, l'objet *verre* peut posséder une propriété *vide* traduisant que le verre est vide si la propriété est vraie et n'est pas vide sinon. Il est aussi possible, par mesure de simplification de description, d'associer une valeur initiale à une propriété. Par défaut, toute propriété non explicitement initialisée sera considérée comme fausse.

Les relations. Les relations sont des faits traduisant une mise en relation de plusieurs objets du monde. Par exemple, le fait qu'un objet soit à l'intérieur d'un placard représente une relation entre l'objet et le placard, qui peut se traduire sous la forme : *dans*(*objet*, *placard*). La syntaxe suivante est utilisée pour traduire une relation :

Relation ::= **relation** *Identifiant* (*Type Identifiant* (**,** *Type Identifiant*)*)
(corpsRelation | **;**)
corpsRelation ::= **{ constraint** expressionContrainte **}**

Une relation possède donc un identifiant et des paramètres. Ces paramètres sont typés, spécifiant ainsi quelles classes d'objets peuvent être mises en relation. Le corps, optionnel, de la relation permet de définir les contraintes à appliquer sur les paramètres. Il s'agit d'une expression booléenne traduisant des contraintes d'égalité ou d'inégalité entre les paramètres. Cette expression des contraintes possède aussi une expression particulière : $from(instance)$. Elle permet d'accéder à l'instance d'objet dont *instance* est une donnée membre. A titre d'exemple, elle peut s'avérer très utile dans le cas de la mise en relation d'une main, appartenant à un humanoïde donné, avec un objet saisi. Dans ce cas, cette expression peut permettre de vérifier que la main passée en paramètre appartient bien à l'humanoïde lui même passé en paramètre. Ces contraintes sont donc d'ordre structurel et définissent le domaine de validité de la relation.

La connaissance. L'ensemble des faits (propriétés ou relations) permettent de décrire l'état du monde. Cependant, leur valeur de vérité n'est pas forcément connue de l'agent. Il s'agit là du principe de l'agent situé dans un monde, qui ne possède qu'une perception restreinte et donc une connaissance fiable relativement réduite. Un agent se promenant dans la rue ne connaît pas ce qu'il se passe chez lui pendant qu'il n'est pas là. Même si l'agent peut avoir conscience de l'ensemble des faits décrivant le monde, il ne peut à un instant donné connaître leur valeur de vérité. Pour être à même de planifier des actions, malgré sa connaissance incomplète, il doit être à même de manipuler cette connaissance. Pour ce faire, l'opérateur K, prenant en paramètre un fait, a été introduit. Il s'agit d'une forme de fait traduit la connaissance de la valeur de vérité de son paramètre. En reprenant l'exemple de la propriété *vide* d'un verre, $K(vide)$ représentera la connaissance de la valeur de vérité de la propriété *vide*.

Nous reviendrons, lors de la description du mécanisme de sélection d'actions, sur le mode de représentation des faits ainsi que de la connaissance. Il faut cependant garder à l'esprit que le langage n'offre pas de fonctionnalités autres que celles permettant de vérifier l'état de la connaissance. L'évolution de cette connaissance est laissée sous la responsabilité de l'agent, qui est le seul à maîtriser ses croyances et leur évolution. Cet ensemble de faits permet de décrire des mondes et des interactions assez complexes. Comme nous allons le voir, leur utilisation, couplée à la connaissance, va permettre d'exprimer diverses formes d'actions, perceptives et/ou effectives.

4.2.2.2 Monde conceptuel et classe d'objet

Une classe d'objet BCOOL décrit des informations de quatre types : l'héritage, les propriétés de l'objet, les données membres et enfin les actions associées à l'objet. La syntaxe générale est la suivante:

Objet	::=	**object** *Identifiant* { Héritage } corpsObjet
Héritage	::=	**inherits** *Type* (**,** *Type*)*
corpsObjet	::=	**{** (donnéeMembre | Action)* **}**

Nous reviendrons plus précisément sur la notion d'héritage dans la section suivante. Pour le moment nous allons nous concentrer sur la description interne de l'objet.

Les données membres. Les données membres d'un objet BCOOL peuvent être de deux types. Il peut s'agir de propriétés telles que nous les avons décrites au paragraphe précédent ou bien d'un

autre objet BCOOL. La syntaxe de déclaration est la suivante :

donnéeMembre ::= (Propriété | *Type Identifiant*) ;

Une donnée membre est une partie intégrante de l'objet, mais, de l'extérieur, elle est utilisée comme un objet à part entière.

Les actions. Les actions décrivent les comportements d'interaction que l'agent peut adopter vis à vis de l'objet. Pour autoriser leur manipulation dans un processus de raisonnement, leur description utilise les notions de pré-conditions et d'effets. Sa description s'effectue par l'intermédiaire de la syntaxe suivante :

Action	::=	**action** Identifiant (Type Identifiant (, Type Identifiant)*) corpsAction
corpsAction	::=	**{** {contrainte} précondition effet cible { coût } actionRéelle **}**
contrainte	::=	**constraint** expressionContrainte ;
précondition	::=	**precondition** condition (, condition)* ;
effet	::=	**effect** condition (, condition)* ;
condition	::=	{ **!** } (**K(** fait **)** | fait)
cible	::=	**target** Identifiant
coût	::=	**cost** cpp
actionRéelle	::=	**action** cpp

L'action accepte un certain nombre de paramètres typés, tout en permettant de spécifier des contraintes structurelles au même titre que pour les relations. Les propriétés et données membres des objets passés en paramètre peuvent être utilisées dans toutes les expressions liées à la description de l'action. D'autre part, le mot clef *this* est disponible pour manipuler les informations dépendantes de l'instance de l'objet à laquelle l'action appartient. L'accès aux données membres des diverses instances s'effectue à l'aide de l'opérateur « . ».

La pré-condition est une conjonction décrivant l'ensemble des faits (propriétés, relations et connaissances) qui doivent être vrais pour permettre l'exécution de l'action. Pour des notions d'efficacité, qui seront montrées dans le mécanisme de sélection d'actions, l'ordre des faits décrivant les pré-conditions d'une action joue un rôle important. Nous nous appuyons ici sur la même contrainte que la planification HTN (voir section 1.2.3). Les pré-conditions associées à une action seront satisfaites dans l'ordre de leur description. D'une part, cette contrainte permet de simplifier le travail du mécanisme de sélection d'actions et, d'autre part, elle permet d'introduire une connaissance opératoire sur l'ordre de réalisation des actions permettant de satisfaire les pré-conditions.

Les effets d'une action sont décrits de la même manière que les pré-conditions. Entre autres, il est possible de spécifier qu'après l'exécution d'une action, l'état d'un fait sera connu. Cette propriété permet de traduire sur le même plan les actions effectives et les actions perceptives. L'agent peut alors générer des actions de perception en vue d'acquérir suffisamment d'informations pour être en mesure de satisfaire le but qu'il s'est fixé. Implicitement, tout fait appartenant aux conséquences d'une action sera considéré comme connu après la réalisation de cette même action. De plus, il est possible de décrire une action dont la conséquence est la non-connaissance d'un fait. Cette particularité permet de stipuler qu'une action peut modifier un fait, mais que ce résultat n'est pas déterministe.

Le coût associé à l'action se traduit sous la forme d'une expression C++ en charge de renvoyer une valeur réelle, supérieure à 0. Il permet de prendre en compte des notions de dépense énergétique,

de temps associées à son exécution. Dans la mesure où il est optionnel, sa valeur par défaut est fixée à 1. Cela implique qu'un mécanisme tentant de minimiser le coût d'une suite d'actions cherchera, par défaut, à minimiser le nombre d'actions.

L'information de cible permet de spécifier quel agent présent dans le monde va pouvoir exécuter l'action. L'action réelle est ensuite décrite par l'intermédiaire d'un code C++ ayant pour contrainte de fournir un automate HPTS++. De cette manière, lorsqu'une action est choisie pour être exécutée, l'automate peut être greffé sur l'agent cible de l'action. Grâce aux bonnes propriétés de synchronisation de HPTS++, nous sommes assurés que, quel que soit le comportement adopté par l'agent, la réalisation de l'action sera cohérente avec les autres comportements en cours. L'agent peut alors mélanger des comportements purement réactifs, définis en dehors du mode de raisonnement avec des comportements que l'on peut qualifier de pro-actifs, contribuant à la réalisation d'un but.

4.2.2.3 Héritage et polymorphisme

Les conséquences de l'utilisation de la notion d'héritage à l'intérieur de BCOOL sont doubles : d'une part, sur le passage d'objets en paramètre des actions ou des relations, et, d'autre part, sur la possibilité d'introduire une notion de polymorphisme permettant de redéfinir des actions le long de la hiérarchie.

La notion d'héritage permet de créer des objets pouvant posséder plusieurs types conceptuels. Par exemple, il est possible de définir la classe des objets manipulables. Ces objets peuvent être pris et posés par un agent. D'autre part il est possible de définir des objets de type récipient qui peuvent contenir quelque chose. Si l'on considère le cas du verre, il s'agit à la fois d'un récipient et d'un objet manipulable. Si cette définition est utilisée pour décrire le verre, toutes les actions ou relations prenant en paramètre des récipients et/ou des objets manipulables accepteront le verre. En fonction du cas, il sera utilisé comme un objet manipulable ou un récipient. Cependant, toute action ou relation prenant en paramètre un verre, n'acceptera pas les objets de type récipient et/ou manipulables à l'exception du verre et de ses classes descendantes.

La notion de polymorphisme permet de redéfinir une action précédemment définie dans une classe mère. Cette notion est utilisée d'une manière un peu particulière et peu classique dans BCOOL. Toutes les actions associées à une classe peuvent être redéfinies par héritage. Cependant, seul le nom d'une action est utilisé pour signifier sa redéfinition, contrairement aux langages objets classiques utilisant aussi le typage des paramètres pour la différentiation. Cette notion, qui peut sembler surprenante de prime abord, trouve sa justification dans la nature même des actions. Conceptuellement l'action d'ouvrir une bouteille de vin et l'action d'ouvrir une canette de bière sont les mêmes. Dans les faits, l'une nécessite l'utilisation d'un tire-bouchon, l'autre d'un décapsuleur. Ces deux objets ne sont pas les mêmes bien qu'il existe des objets faisant à la fois décapsuleur et tire-bouchon, qui constituent la classe des ouvre-bouteilles. Si l'on considère la classe des choses pouvant être ouvertes, la bouteille et la canette de bière en font partie, même si dans les faits les objets utilisés (et donc pris en paramètre de l'action) pour les ouvrir ne sont pas les mêmes. De fait, la redéfinition d'une action ne se fait que par nom et permet de modifier le nombre de paramètres ainsi que leur type à travers les différents niveaux d'héritage.

Là encore, l'on peut se poser la question de la validité de cette approche en terme d'utilisation des actions associées aux objets. Comment allons-nous pouvoir les paramétrer correctement si leurs paramètres changent constamment. La réponse est simple : le paramétrage des actions n'est pas fait par l'utilisateur mais par un moteur d'inférence que nous allons décrire. Ce mécanisme est donc transparent pour l'utilisateur et ne contraint en rien son utilisation de ce pouvoir d'expression accru.

4.2.2.4 Exemple de monde conceptuel BCOOL.

L'exemple développé ici, fait partie des exemples les plus simples. Nous allons décrire un humanoïde, équipé de deux mains et pouvant se déplacer dans un environnement représenté par plusieurs lieux. Ces lieux peuvent contenir des objets, qui peuvent être pris et posés. Ce problème est représenté par quatre classes d'objets : *Lieu*, *Humain*, *Manipulable* et *Verre*. Les instances de ces objets peuvent être liées par l'intermédiaire des trois relations suivantes :

```
relation objetEstDans(Manipulable m, Lieu l) ;

relation humainEstDans(Humain h, Lieu l) ;

relation prisPar(Manipulable objet, Humain h, Main m)
{ constraint from(m)=h ; }
```

Les relations *objetEstDans* et *humainEstDans* traduisent la localisation des objets et des humains dans l'environnement. La relation *prisPar* utilise une contrainte structurelle précisant que la main passée en paramètre doit appartenir à l'humain lui même passé en paramètre. Cette contrainte permet d'éviter de produire des instances de relations incohérentes. Les objets sont ensuite définis de la manière suivante[3] :

```
object Lieu {}

object Main { property vide = true ; }

object Humain
{   Main mainGauche ;
    Main mainDroite ;

    action allerA(Lieu l1, Lieu l2)
    {   precondition    connexe(l1,l2), humainEstDans(h,l1) ;
        effect          !humainEstDans(h,l1), humainEstDans(h,l2) ;
        target          this ;
        action ...
    }
}

object Manipulable
{   property pris = false ;

    action prendre(Humain h, Main m, Lieu l)
    {
        constraint      from(m)=h ;
        precondition    K(objetEstDans(this,l)), objetEstDans(this,l),
                        main.vide, humainEstDans(h,l);
        effect          !objetEstDans(this,l), !main.vide,
```

3. Les actions réelles, sous la forme d'automates HPTS++ ne sont pas détaillées.

```
                        prisPar(this,h,m), this.pris ;
        target h ;
        action ...
    }

    action poser(Human h, main m, Lieu l)
    {
        constraint      from(m)=h ;
        precondition    prisPar(this,h,m), humainEstDans(h,l);
        effect          main.vide, !this.pris,
                        !prisPar(this,h,m), objetEstDans(this,l) ;
        target h ;
        action ...
    }
}

object Verre inherits Manipulable {  property    vide = true ; }
```

Ces quatre classes d'objets permettent de définir des mondes dans lesquels des humanoïdes peuvent utiliser des objets manipulables (y compris les verres) et les déplacer d'un lieu à un autre. L'action *allerA* possède, en pré-condition, une relation qui n'est pas définie dans la description des relations que nous avons précédemment effectuée : *connexe*. Il s'agit d'une relation structurelle, elle est fournie lors de la description du monde réel et n'est pas modifiable. Si elle est présente dans le monde, l'action pourra être effectuée, dans le cas contraire, l'action ne sera pas valide. Intéressons nous de plus près à l'action *prendre* associée aux objets manipulables et à ses pré-conditions ordonnées. Dans un premier temps, pour pouvoir prendre un objet, l'humanoïde doit savoir où il se trouve. Dans un second temps, il cherche à libérer la main pour pouvoir prendre l'objet. Enfin, il se rend dans le lieu où il peut prendre ce même objet. La notion d'ordre de satisfaction des pré-conditions d'une action est très importante, elle permettra de guider le processus de sélection d'actions. Cependant, elle implique certaines précautions de description. Dans certains cas, l'inversion de cet ordre peut conduire à des impossibilités de réalisation. Prenons l'exemple de l'action *poser* de la classe *manipulable*. En inversant l'ordre des pré-conditions, l'humanoïde devrait d'abord être dans la pièce puis prendre l'objet. Si l'objet ne se trouve pas dans cette pièce, l'action ne peut être réalisée.

4.2.2.5 Description d'un monde réel

La partie précédente s'est focalisée sur la description des classes d'objets peuplant le monde ainsi que les possibilités d'interaction qu'elles peuvent offrir. Une fois cette description « schématique » réalisée, il faut décrire un monde réel, contenant des objets et des agents. La description du monde se résume à :

- la description des objets peuplant l'environnement ;
- l'initialisation éventuelle de leurs propriétés ;
- la description des relations vraies dans l'état initial.

Elle permet donc de décrire un état cohérent au départ de la simulation. La description du monde s'effectue par la grammaire suivante :

monde	::=	(Objet | Relation)*
Objet	::=	*Type* Instance
Instance	::=	*Identifiant* { **true** | **false** | **{** (Instance)+ **}** }
Relation	::=	*Identifiant* (listeParamètres)
listeParamètres	::=	donnée (**,** donnée)*
donnée	::=	*Identifiant* (**.** *Identifiant*)*

Le monde est donc décrit comme un ensemble d'objets typés pour lesquels les données membres peuvent être initialisées différemment de leur valeur par défaut, suivant une syntaxe proche de VRML. Les relations, étant par défaut considérées comme fausses, sont initialisées à vrai lorsqu'elles sont présentes dans cette description. Pour donner un exemple d'une telle description, nous allons reprendre l'exemple de la section 4.2.2.4. Supposons que nous voulions décrire un monde peuplé par deux humanoïdes, deux verres et deux lieux. Ajoutons la contrainte que l'un des humanoïdes possède un verre dans sa main droite. La description sera la suivante :

```
Lieu Cuisine
Lieu Salon

Humain paul { mainDroite { empty false } }
Humain jean

Verre verre1
Verre verre2
{   pris true
    vide false }

humainEstDans(paul, salon)
humainEstDans(jean, cuisine)

prisPar(verre2, paul, paul.mainDroite)

objetEstDans(verre1, cuisine)

connexe(Cuisine, Salon)
connexe(Salon, Cuisine)
```

Cette situation traduit le fait que Paul se trouve dans le salon avec un verre rempli dans la main, alors que Jean se trouve dans la cuisine, peut-être à la recherche du verre qui s'y trouve. Les instances de relations de type *connexe* sont fournies pour autoriser les déplacements des humanoïdes d'une pièce à une autre. Dans cet exemple, nous retrouvons la difficulté inhérente à la description de mondes dans un formalisme permettant le raisonnement : il faut pouvoir fournir une situation cohérente lors de l'initialisation du système. Si une erreur s'introduit dans cette description, le système ne peut assurer la cohérence des actions qu'il produira.

4.2.3 Génération du monde

La génération du monde se base sur les deux descriptions qui viennent d'être présentées : d'une part la description du monde conceptuel et d'autre part la description d'un monde réel bâti sur les concepts précédemment décrits. Ces deux descriptions sont ensuite compilées pour permettre la génération d'une base de données contenant toutes les informations utiles pour les agents autonomes. Cette génération se base sur le moteur d'inférence PROLOG et sur un noyau de calcul dépendant de BCOOL.

Compilation. Le principe de la compilation consiste à produire deux fichiers PROLOG : un fichier contenant des prédicats de description des classes et relations décrites dans le monde conceptuel et un fichier de déclaration d'instances. Ces deux fichiers, sont fournis en entrée du moteur d'inférence, pour produire la base de données.

Génération de la base de données. Les fichiers générés lors de la compilation sont utilisés par le noyau PROLOG de BCOOL. Ce noyau possède quatre rôles :

- il déclare l'ensemble des instances de données membres et d'actions propres à chaque objet décrit dans le fichier de description d'un monde réel.
- il paramètre les relations en respect des contraintes structurelles qui leur sont associées.
- il paramètre les actions, en respect des contraintes structurelles qui leurs sont associées et en propageant les contraintes liées à l'utilisation de relations à l'intérieur des pré-conditions et des effets des actions. Dans le cas d'actions redéfinies par héritage, seul le paramétrage de la dernière action déclarée est tenté, les autres actions ne sont pas prises en compte.
- il associe aux relations, et aux propriétés leurs valeurs d'initialisation extraites des deux fichiers de description, en fournissant en priorité les valeurs fournies lors de la description du monde réel.

Durant la phase d'inférence du moteur, les informations sur les instances des objets, leur type, des données membres, des relations paramétrées, les actions paramétrées et les valeurs d'initialisation sont stockées dans une base de données. Les informations qu'elle contient peuvent alors être manipulées sous la forme de requêtes portant sur tous les champs de description.

Complexité. Cette phase de génération est un pré-calcul, elle est exécutée avant le déroulement de la simulation. Le temps de génération dépend de plusieurs facteurs dont la qualité du moteur d'inférence PROLOG utilisé et le nombre d'instances d'objets à manipuler. Cependant, ce temps ne constitue pas le problème majeur dans la mesure où le calcul n'est pas fait en temps réel. La taille de la base de donnée, qui sera utilisée par le mécanisme de sélection d'action, peut pour sa part poser problème. Le nombre d'instances de relations et d'actions ainsi générées est polynômial, même si les contraintes structurelles et le typage permettent de réduire la combinatoire. La taille des mondes à gérer doit donc être raisonnable. Si la base de données associée à la description d'une ville peuplée ne peut être envisageable, le système est cependant à même de décrire des environnements permettant l'interaction de plusieurs humanoïdes avec des dizaines d'objets. Ce problème de combinatoire est inhérent à une grande partie des systèmes de planification dont, par exemple, GRAPHPLAN qui utilise, lors de la planification, des structures de données ayant une combinatoire encore plus élevée [BF97].

L'idée véhiculée par ce langage est celle de la modélisation. Nous avons cherché à offrir des mécanismes, adaptés de la modélisation objet, facilitant la description des connaissances. La notion de classe couplée au mécanisme d'héritage facilite d'une part la description et permet, implicitement, à l'agent de travailler sur des concepts catégorisant les objets peuplant l'environnement. La description des interactions à l'intérieur des objets permet pour sa part de déporter la conception en la centrant sur ce que l'environnement permet et non plus sur ce que l'agent peut faire. Finalement, ce mode de représentation considère que l'agent est situé dans l'environnement et que cet environnement contraint en grande partie son comportement par les opportunités qu'il lui offre, ce qui correspond grandement à la théorie écologique de Gibson [Gib86].

4.3 Mécanisme de sélection d'actions

A partir de la base de données générée en sortie de BCOOL, les agents sont capables de posséder leur propre représentation du monde. Le paramétrage des actions qu'elle contient permet de connaître les possibilités d'interaction de l'agent avec son environnement. Il s'agit désormais d'exploiter cette information pour être à même de permettre à l'agent de sélectionner des actions à réaliser en vue de satisfaire un but qu'il se sera fixé. Il s'agit du rôle du mécanisme de sélection d'action que nous allons décrire. Dans un premier temps, nous allons décrire la représentation interne utilisée pour gérer les connaissances et les croyances de l'agent, ainsi que les contraintes qui régissent leur évolution et garantissent la cohérence du système. Cependant, l'évolution des connaissances n'est pas modélisée, nous considérons que cette évolution est dépendante de chaque agent qui est donc responsable de sa gestion. Puis nous présenterons les formes de buts que le système admet pour découler sur le mécanisme de sélection d'actions proprement dit. L'idée à garder en mémoire tout au long de l'explication qui va suivre est la dynamique. Les agents évoluent dans un monde qui peut changer à tout instant, sans leur intervention et sans qu'ils en soient au courant. Ce mécanisme a donc pour rôle de sélectionner des actions qui semblent appropriées en fonction de l'état des connaissances de l'agent et doit prendre en compte, dès que possible, les nouvelles connaissances acquises.

4.3.1 Représentation interne

Faits. Le monde généré à partir du moteur d'inférence contient plusieurs faits et actions. Les faits sont de trois types : les propriétés associées aux objets, les relations entre objets et enfin les applications. Pour être à même d'effectuer les calculs permettant de sélectionner les actions à réaliser, une représentation interne dans laquelle chaque fait généré est représenté par quatre faits est utilisée. Notons f un fait (propriété/relation/application) présent dans le monde généré. L'ensemble F des faits utilisés pour la sélection d'actions contiendra :

- f le fait originel, présent dans le monde généré. Nous allons noter F_x l'ensemble de ces faits.
- $\overline{f}$ le fait réciproque de f. Prenons l'exemple d'une propriété *vide* associée à un verre, $\overline{vide}$ représente le fait que le verre n'est pas *vide*. D'après cette définition, nous avons $f = \overline{\overline{f}}$. Nous allons noter $F_{\overline{x}}$ l'ensemble de ces faits.
- $K(f) \Leftrightarrow K(\overline{f})$ représente la connaissance de la valeur de vérité du fait f ou $\overline{f}$. Nous allons noter F_K l'ensemble de ces faits.
- $\overline{K(f)} \Leftrightarrow \overline{K(\overline{f})}$ représente la non-connaissance de la valeur de vérité du fait f ou $\overline{f}$. Nous allons noter $F_{\overline{K}}$ l'ensemble de ces faits.

Par définition, nous avons $F = F_x \cup F_{\overline{x}} \cup F_K \cup F_{\overline{K}}$. Chaque fait est donc représenté de quatre manières, prenons l'exemple d'un fait *vide* associé à un verre, ce fait génèrera : *vide*, $\overline{vide}$, $K(vide)$ et $\overline{K(vide)}$. Ces quatre manières de représenter le fait sont intimement liées, l'une ne peut évoluer sans l'autre.

Pour simplifier les notations, l'opérateur K associé à la connaissance des faits et étendu à un ensemble de faits pour symboliser l'ensemble des connaissances qui leurs sont associées :

$$\forall E \subset F,\, K(E) = \{K(x) \mid x \in F_x \cup F_{\overline{x}} \wedge x \in E\} \cup (E \cap F_K)$$

De même, soit un ensemble $E \subset F$, l'ensemble $\overline{E}$ définit l'ensemble des faits réciproques des faits contenus dans E :

$$\forall E \subset F,\, \overline{E} = \{\overline{x} \mid x \in E\}$$

Valeur de vérité. La notion de valeur de vérité est gérée de manière booléenne dans le langage BCOOL ; la raison étant que ce langage décrit un monde effectif et non un monde approximatif. Maintenant, nous nous situons du coté de l'agent et de sa propre représentation du monde. La connaissance n'est désormais plus forcément booléenne mais peut devenir floue, permettant ainsi de qualifier une incertitude. Une valeur de vérité, se trouvant dans l'intervalle $[0; 1]$, est associée à chaque fait de F. Dans cet intervalle, 0 est équivalent à faux et 1 à vrai, les valeurs intermédiaires permettent de qualifier une incertitude qui peut évoluer en fonction du temps. Pour chaque fait f et $\overline{f}$, la valeur de vérité (notée $val(f)$) respecte constamment l'équation suivante :

$$\forall f \in F,\, val(f) = 1 - val(\overline{f})$$

D'autre part, la valeur de vérité associée à $K(f)$, traduisant la connaissance de la valeur de vérité du fait f est définie par :

$$\forall f \in F_x \cup F_{\overline{x}},\, val(K(f)) = 2 \times |0.5 - val(f)|$$

Il est à noter que cette équation respecte l'équivalence entre $K(f)$ et $K(\overline{f})$. Ces informations permettent de décrire l'état des croyances de l'agent sur le monde dans lequel il évolue. La mesure fournie par l'opérateur K permet de manipuler la connaissance sur l'état du monde. En fonction de $val(K(f))$, nous distinguons deux cas :

- $val(K(f)) \geq 0.5$ signifie que la valeur de vérité du fait est considérée comme connue. Cette inéquation implique que $(0 \leq val(f) \leq 0.25) \vee (0.75 \leq val(f) \leq 1)$.
- $val(K(f)) < 0.5$ signifie que la valeur de vérité du fait est considéré comme non fiable. Ce type de configuration nécessite d'acquérir de l'information pour être en mesure de fixer la valeur de vérité de f en vue de son utilisation pour la sélection d'actions. Cette inéquation implique que $0.25 < val(f) < 0.75$.

La notion de connaissance d'un fait peut être utilisée pour gérer des actions perceptives diverses permettant d'acquérir de la connaissance sur le monde. La figure 4.2 traduit les différents cas qui peuvent se produire en fonction de la valeur de vérité associée à un fait. Lors de la planification, la connaissance des faits servira dans les pré-conditions des actions. Cependant, pour ne pas bloquer le processus, si la connaissance de la valeur de vérité d'un fait n'est pas fiable, une hypothèse sera faite. Si $0.25 < val(f) < 0.5$ le fait sera considéré comme faux et si $0.5 \leq val(f) < 0.75$ le fait sera considéré comme vrai, cependant $K(f)$ sera considéré comme faux. La notion de connaissance d'un

fait est utilisée de manière explicite dans la description des actions (voir section 4.2.2.2). Si aucune pré-condition ne stipule explicitement un besoin de connaissance sur un fait, les hypothèses sur les valeurs de vérité seront utilisées pour la sélection d'actions.

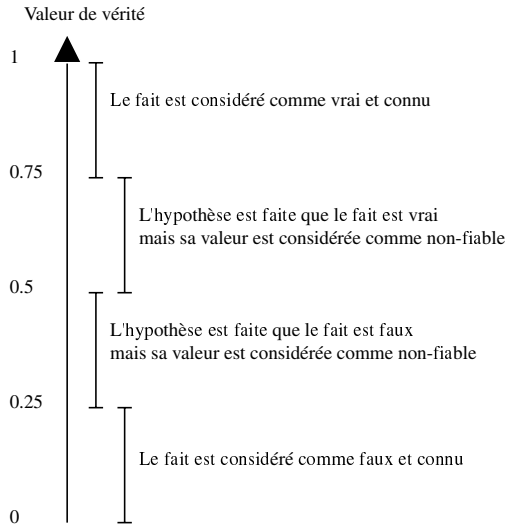


FIG. 4.2 – *Valeur de vérité d'un fait, hypothèses et connaissance*

Modification des valeurs de vérité. L'évolution des valeurs de vérité des faits n'est pas directement gérée par le système, elle doit être gérée par l'agent qui est le seul à savoir dans quelle mesure il peut ou non faire confiance à sa connaissance. Cependant, la modification d'une valeur de vérité doit être correctement gérée pour assurer la cohérence du système. Deux cas sont à distinguer :

- la mise-à-jour de la valeur de vérité de f implique les mises-à-jour de $\overline{f}$, $K(f)$ et $\overline{K(f)}$ comme suit :

$$\begin{aligned} val(\overline{f}) &= 1 - val(f) \\ val(K(f)) &= 2 \times |0.5 - val(f)| \\ val(\overline{K(f)}) &= 1 - val(K(f)) \end{aligned}$$

  de manière réciproque, la mise à jour de $\overline{f}$ provoque la mise à jour de f, $K(f)$ et $\overline{K(f)}$.
- la mise-à-jour de la connaissance d'un fait, autrement dit de $K(f)$ est plus délicate. Elle doit rester en concordance avec l'hypothèse de véracité qui a été exposée. Pour ce faire, nous utilisons les formules suivantes :

$$\begin{aligned} val(f) &= (val(f) - 0.5) \times K(f) + 0.5 \\ val(\overline{f}) &= (val(\overline{f}) - 0.5) \times K(f) + 0.5 \\ val(\overline{K(f)}) &= 1 - val(K(f)) \end{aligned}$$

Actions. Les actions sont décrites par l'intermédiaire de leurs pré-conditions, de leurs effets, de la connaissance qu'elles permettent d'acquérir ainsi que de leur coût (énergétique ou temporel). Pour

la suite de l'explication, les notations suivantes vont être utilisées :

- $Pre(op) \subset F$ sont les pré-conditions de l'opérateur op. Dans ce cas, nous considérons les pré-conditions d'une action comme un ensemble. Cependant, comme nous l'avons décrit précédemment, l'ordre des pré-conditions joue un rôle important et doit donc être conservé.
- $E_{ff}(op) \subset F$ sont les effets d'une action op.
- $nextPre(op)$ est une fonction permettant de fournir la première occurrence de pré-condition insatisfaite, en parcourant les pré-conditions dans leur ordre de déclaration. Prenons l'exemple d'une action op_e possédant comme pré-condition $\{b,a,c\}$. Supposons que b et c soient vrais et que a soit faux. Nous avons $nextPre(op_e) = a$.
- $truePre(op)$ est une fonction fournissant l'ensemble des pré-conditions considérées comme vraies et précédant dans l'ordre de déclaration des pré-conditions $nextPre(op)$. En reprenant l'exemple précédent, $truePre(op_e) = \{b\}$, c n'est pas pris en compte car il se trouve après a (qui est faux) dans l'ordre de déclaration.
- $cost(op) \in \mathbb{R}$ est le coût (temporel / énergétique...) associé à la réalisation de l'action op.
- $exe(op)$ est une fonction booléenne permettant de savoir si l'action est exécutable ou non. Elle est vraie si toutes les pré-conditions de l'action sont supposées vraies par l'agent.

A partir des informations fournies sur les actions, nous calculons l'ensemble des faits ajoutés par cette action :

$$Add(op) = (E_{ff}(op) \cup K(E_{ff}(op))) - Pre(op)$$

Cet ensemble traduit l'ensemble des faits ajoutés par l'exécution de l'action, y compris la connaissance qu'elle permet d'acquérir. Pour justifier l'ajout de $K(E_{ff}(op))$, prenons l'exemple d'une action ayant pour pré-condition la connaissance d'un fait mais pas son état (vrai ou faux). Toute action affirmant l'état de ce fait, engendre la connaissance de cet état. Cependant, si cette connaissance n'est pas explicitée, il est difficile de corréler ces deux actions. C'est la raison pour laquelle la connaissance est automatiquement ajoutée aux effets des actions, même si celle-ci n'y était pas à l'origine. D'autre part, nous pouvons définir l'ensemble des faits supprimés par cette action comme l'ensemble des faits réciproques des faits ajoutés :

$$Del(op) = \overline{Add(op)}$$

A partir de ces définitions, de $Add(op)$ et $Del(op)$, nous allons pouvoir travailler sur les enchaînements des actions et détecter d'éventuelles incompatibilités lors de la sélection d'action.

4.3.2 Les buts

Le système offre la gestion de deux grandes classes de buts : les buts atomiques, opérant sur un fait, et les buts évolués, permettant de caractériser une situation à obtenir. Avant de rentrer dans les détails de la spécification de ces buts, il est nécessaire d'introduire les notions d'inhibition et d'activation liées aux faits :

- $\forall f \in F$, $inhibition(f) \in [0;1]$ qualifie l'inhibition associée au fait, plus sa valeur est grande, plus le fait est inhibé. Elle traduit la volonté de ne pas satisfaire ce fait.
- $\forall f \in F$, $activation(f) \in [0;1]$ qualifie l'activation associée au fait, plus sa valeur est grande plus le fait est activé. Elle traduit la volonté de satisfaire ce fait.

En fonction des valeurs d'inhibition et d'activation, il est possible de définir $sat(f)$ qui traduit la volonté de satisfaire ce fait, que par la suite nous appellerons satisfaction du fait f :

$$\forall f \in F,\ sat(f) = activation(f) - inhibition(f)$$

En fonction de la valeur renvoyée par cette fonction, se situant dans l'intervalle $[-1;1]$, il est possible de distinguer deux cas :

- $sat(f) \leq 0$ signifie que ce fait ne doit pas être satisfait. Dans ce cas précis, toutes les actions op telles que $f \in Add(op)$ ne devraient pas être exécutées.
- $sat(f) > 0$ signifie que ce fait doit être satisfait, et dans ce cas, la valeur traduit la volonté de satisfaire ce fait et devrait donc provoquer l'exécution des actions op telles que $f \in Add(op)$.

Buts atomiques. Les activations et les inhibitions des faits, par l'intermédiaire de leur influence sur le résultat de la fonction sat permettent de qualifier plusieurs formes de buts et/ou d'inhibitions contraignant le comportement de l'agent. Les trois formes de but sont décrites de la manière suivante :

- $avoid(f,v)$ avec $(f \in F) \wedge (v \in [0;1])$ qualifie un fait à éviter de réaliser, ces types de buts sont stockés dans l'ensemble $B_a \subset (F \times [0;1])$. Il s'agit d'une forme de but passif. L'agent ne doit pas effectuer une action en vue de le satisfaire, mais plutôt éviter les actions le réalisant. Cette expression se traduira sous la forme $inhibition(f) = v$, diminuant ainsi la valeur de $sat(f)$.
- $realize(f,v)$ avec $(f \in F) \wedge (v \in [0;1])$ qualifie un fait à réaliser, ces types de buts sont stockés dans l'ensemble $B_r \subset (F \times [0;1])$. L'agent devrait donc effectuer une suite d'actions en vue de satisfaire ce but. Une fois satisfait, il est automatiquement supprimé de de l'ensemble B_r des buts à réaliser. Cette expression se traduira sous la forme $activation(f) = v$, augmentant ainsi la valeur de $sat(f)$.
- $maintain(f,v)$ avec $(f \in F) \wedge (v \in [0;1])$ qualifie un fait à maintenir, ces types de buts sont stockés dans l'ensemble $B_m \subset (F \times [0;1])$. Il s'agit là d'un but un peu particulier. Si le le but n'est pas satisfait, une suite d'actions devrait être effectuée pour le réaliser. D'autre part, les actions conduisant à $\overline{f}$ ne devraient pas être exécutées, dans la mesure où elles ne maintiennent pas la vérité de f. Enfin, lorsque le fait est satisfait, le but n'est pas supprimé de B_m dans la mesure où des événements extérieurs (ou l'agent lui même dans certains cas), pourraient le rendre faux. Cette expression se traduira sous la forme $(activation(f) = v) \wedge (inhibition(\overline{f}) = v)$, augmentant ainsi $sat(f)$ et diminuant $sat(\overline{f})$.
- $lock(f,v)$ avec $f \in F \wedge v \in [0;1]$ qualifie des faits verrouillés, ces types de buts sont stockés dans l'ensemble V. Il s'agit d'une forme de but utilisée dans le moteur de planification pour gérer la cohérence des choix d'actions. Il s'agit d'une expression ayant les mêmes caractéristiques qu'un but de type $maintain(f,v)$ avec une différence qui est que ce but n'est pas actif. Il ne peut donc pas être directement générateur d'actions.

Ces trois formes de buts permettent de traduire des situations relativement complexes comportant des choses à faire et à ne pas faire. Les buts de type $avoid(f,v)$ permettent de décrire des situations qui ne doivent pas arriver. Elles peuvent être utiles pour traduire des habitudes de vie, des comportements à adopter ou à ne pas adopter en fonction du contexte. Les buts de type $realize(f,v)$ traduisent les choses que l'entité voudrait voir réalisées dans l'environnement. Les buts de type $maintain(f,v)$ qualifient une forme d'habitude, qui peut être qualifiée d'obsessionnelle dans le cas d'une valeur de v très élevée consistant à toujours maintenir un fait à une certaine valeur dans

l'environnement. Dès que ce fait est perçu comme faux, il doit de nouveau devenir vrai et ne devrait pas changer de valeur en conséquence des actions de l'agent. Finalement, les verrous ($lock(f,v)$) seront utilisés par le mécanisme de sélection d'action pour prendre en compte les éventuels conflits entre actions. Nous reviendrons sur cette notion lors de la description du mécanisme.

Buts évolués. Les formes de buts qui viennent d'être décrites ne prennent en compte que des faits atomiques. Généralement, un but constitue une situation à atteindre, autrement dit, un ensemble de faits qui doivent être satisfaits simultanément dans l'environnement. Pour traduire cette notion, nous utilisons la notion de but évolué. Ce type de but est de la forme des buts atomiques de type *realize*, mais traduit une liste de faits, à satisfaire dans leur ordre de déclaration et simultanément. Cette notion présente de grandes similarités avec la gestion des pré-conditions d'une action ; il s'agit d'une particularité que nous allons exploiter pour transformer un but évolué en but atomique.

Un but évolué est spécifié au travers d'une liste l de faits et de son importance v. Lorsqu'un tel but est spécifié, une action « fantôme » op_b est générée, possédant l comme pré-condition. Un fait b, lui aussi « fantôme », est généré et est utilisé comme effet de l'action op_b. Le but b est alors ajouté sous la forme $realize(b,v)$. L'action op_b est donc la seule action à pouvoir satisfaire le but b, qui ne peut être atteint qu'en rendant vraie la liste de pré-conditions de l'action op_b, qui constitue elle-même le but évolué. Dès que le but b est atteint, il est supprimé de l'ensemble B_r des buts à réaliser et l'action op_b est elle-même supprimée de la base d'actions. Cet « artifice » nous permet de gérer les buts évolués exactement de la même manière qu'un but simple, simplifiant ainsi leur mode de gestion sans nécessiter de traitement particulier autre que celui qui vient d'être décrit.

4.3.3 Graphe de planification et mise en couche

La sélection d'action va se baser sur les concepts qui viennent d'être évoqués. Cependant, avant de pouvoir choisir une action à exécuter, il faut posséder une structure de données permettant d'effectuer le choix. Pour être en mesure de créer une heuristique permettant de trouver l'action nécessaire à la réalisation d'un but, nous allons utiliser un graphe de planification ainsi qu'un traitement lié à ce graphe. Ce traitement consiste en une mise en couche à partir des informations de connaissance, d'activation, d'inhibition ainsi que du coût des actions. Cette mise en couche sera par la suite utilisée comme heuristique pour la recherche de l'action à effectuer.

Construction du graphe. Le graphe de planification possède deux types de nœuds :

- Les nœuds de type action, représentant une action qui peut être effectuée par un agent en vue de modifier l'état de l'environnement et/ou de ses connaissances.
- Les nœuds de type fait, représentant tous les faits de l'ensemble F.

Ces nœuds sont ensuite interconnectés par l'intermédiaire de liens. Il existe deux types de lien :

- le lien de pré-condition qui lie un fait appartenant à $Pre(op)$ à l'action op.
- le lien de conséquence, qui lie une action op à un fait appartenant à $Add(op)$.

Ce graphe ne contient donc pas l'intégralité de l'information qui peut être extraite à partir de la description du monde. Il ne contient pas les informations en rapport avec les exclusions entre les actions qui peuvent être obtenues par l'intermédiaire des faits supprimés par cette même action. Cette propriété constitue l'une de ses lacunes mais, comme nous le verrons par la suite, la notion de verrou (buts de type *lock*) sera utilisée pour gérer les éventuels conflits.

Ce graphe est ensuite mis en couche en partant des faits connus et/ou considérés comme vrais, en allant ensuite vers les faits considérés comme faux. Cette mise en couche se scinde en deux phases :

1. les activations et inhibitions liées aux faits sont propagées dans le graphe en partant des buts.
2. le graphe est mis en couche en fonction des coûts associés aux actions, tout en exploitant les informations liées à l'activation.

Initialisation des activations et inhibitions. Les inhibitions et les activations des faits sont initialisées par les valeurs associées aux buts de B_a, B_r, B_m et V. L'activation d'un fait f est définie en fonction de l'ensemble des buts $A(f) = \{(f,v) \mid \exists v, (f,v) \in B_r \cup B_m \cup V\}$:

$$activation(f) = \begin{cases} Max_{(a,v)\in A(f)} v & si \quad A(f) \neq \emptyset \\ 0 & sinon \end{cases}$$

Cette équation traduit que l'activation du fait f est l'activation maximale des buts de B_r, B_m et V dont il est constituant, ou 0 s'il ne participe à aucun but. Pour sa part, l'inhibition est définie sur la base de l'ensemble $I(f) = \{(f,v) \mid (f,v) \in B_a\} \cup \{(f,v) \mid (\overline{f},v) \in B_m \cup V\}$ comme suit :

$$inhibition(f) = \begin{cases} Max_{(i,v)\in I(f)} v & si \quad I(f) \neq \emptyset \\ 0 & sinon \end{cases}$$

Cette équation traduit que l'inhibition du fait f est l'activation maximale entre les buts de B_a dont il est constituant et les buts de $B_m \cup V$ concernant $\overline{f}$.

La définition de ces activations et inhibitions initiales des faits met en balance tous les buts concernant les faits. Elle a pour rôle de collecter l'activation dans le meilleur des cas et l'inhibition dans le pire des cas. Nous pouvons alors obtenir par l'intermédiaire de $sat(f)$ la propension générale à vouloir satisfaire le fait f.

Propagation des activations. Chaque fait, comme il vient d'être décrit, possède une valeur d'activation et d'inhibition. Cette valeur pour le moment est fixée par la spécification des buts. Cependant, un fait peut s'avérer inhibé mais être nécessaire à la réalisation d'un autre but. Dans ce cas, si l'importance du but auquel il participe surpasse son inhibition, il semble normal de permettre de passer outre cette même inhibition. Il s'agit du rôle de la propagation des activations. Cette propagation se fait à partir des buts activés dont la satisfaction est supérieure à zéro. Pour permettre la propagation, la notion d'activation et d'inhibition est étendue à l'action ; l'activation d'une action *op* est définie comme suit :

$$activation(op) = Max_{(f\in Add(op) \wedge val(\overline{f}) \geq 0.5)} activation(f) \tag{4.1}$$

L'activation de l'action est donc fonction de l'activation des faits faisant partie de ses conséquences et sa valeur correspond au maximum de l'activation de ses conséquences fausses (qui ont donc besoin d'être réalisées). Pour sa part, l'inhibition est définie comme suit :

$$inhibition(op) = Max_{f\in Add(op)} inhibition(f) \tag{4.2}$$

A la différence de l'activation, les inhibitions sont héritées de toutes les conséquences de l'action, y compris celles qui sont déjà satisfaites. Cette propriété est liée à la nature d'une inhibition, si l'on ne veut pas que quelque chose se réalise, on ne va pas réaliser une action renforçant sa véracité. Dès

lors, il est possible de définir la tendance à exécuter une action, au même titre que la tendance à satisfaire un fait :

$$sat(op) = activation(op) - inhibition(op) \tag{4.3}$$

Si $sat(op) \leq 0$ alors l'action est verrouillée et ne peut être exécutée. Cela traduit que le fait d'exécuter des actions viole des inhibitions bien plus importantes que les buts qu'elle contribue à satisfaire. Pour propager cette notion d'activation aux pré-conditions de cette action, une *mise à jour* de l'activation des pré-condition est effectuée. Elle se traduit par la formule suivante :

$$activation(f) = Max(activation(f), Max_{\{op \mid f \in Pre(op) \wedge sat(op) > 0\}} sat(op)) \tag{4.4}$$

Cette formule se base sur l'expression de $sat(op)$ plutôt que sur l'expression de $activation(op)$. Cela s'explique par le verrouillage des actions en fonction de $sat(op)$, si une action est considérée comme verrouillée, elle ne peut activer ses pré-conditions. D'autre part, si la tendance à exécuter l'action est faible, la tendance à vouloir satisfaire ses pré-conditions l'est aussi. Cela nous permettra de favoriser les actions possédant des conséquences qui ne violent pas les inhibitions.

Les formules 4.1, 4.2, 4.4 de mise à jour des activations et inhibitions associées aux actions et aux faits sont répétées par un algorithme jusqu'à convergence des valeurs. Cet algorithme fonctionne en initialisant un ensemble de recherche contenant, au départ, tous les faits constituant des buts. Ensuite, tant que cet ensemble n'est pas vide, un fait est tiré. Pour chaque action permettant d'obtenir ce fait, les inhibitions et activations sont calculées. S'il y a modification, pour une action, de sa valeur d'activation ou d'inhibition, ses pré-conditions sont ajoutées dans l'ensemble. L'algorithme itère jusqu'à ce que l'ensemble soit vide.

Mise en couche du graphe. La mise en couche du graphe de planification permet de traduire la distance qui sépare les faits répertoriés comme vrais, des faits constituant des buts pour l'agent. Pour ce faire, l'algorithme associe à chaque fait et à chaque action un rang. Pour chaque fait $f \in F$, le rang est calculé de la manière suivante :

$$rang(f) = \begin{cases} 0 & si\ val(f) \geq 0.5 \\ Min_{\{op \mid f \in Add(op) \wedge sat(op) > 0\}} rang(op) & si\ \exists\ op,\ f \in Add(op) \wedge sat(op) > 0 \\ \infty & sinon \end{cases} \tag{4.5}$$

Cette expression associe à chaque fait, le rang minimal de l'action le possédant comme effet. Pour sa part, le rang d'une action est défini comme suit :

$$rang(op) = \begin{cases} \frac{cost(op)}{sat(op)} + \sum_{f \in Pre(op)} rang(f) & si\ sat(op) > 0 \\ \infty & sinon \end{cases} \tag{4.6}$$

La notion de rang pour l'action prend en compte plusieurs facteurs. Ce rang dépend de la somme des rangs des pré-conditions de l'action. Il s'agit d'une expression similaire à l'heuristique non admissible utilisée dans la planification HSP (voir section 1.2.2), considérant que les pré-conditions sont indépendantes. Cependant, comme décrit par Bonet [BG01], cette heuristique fournit une grande quantité d'informations. D'autre part, il prend en compte le coût de l'action pondéré par la propension à exécuter l'action. Cette pondération permet de favoriser les actions dont les conséquences ne violent pas les inhibitions.

A chaque changement répertorié de l'état du monde, cette mise en couche va immédiatement prendre en compte ses conséquences. Ces changements incluent les conséquences des actions réalisées mais aussi les changements annexes : effets de bord non-décrits des actions et changements perçus. Elle prend donc en compte la dynamique de l'environnement, tout en offrant une heuristique traduisant la distance entre la situation courante et les buts, pondérée par les inhibitions et activations qui leurs sont associées. Elle constitue le goulet d'étranglement en terme de calculs. Dans les cas où beaucoup de buts sont exprimés et où le nombre d'actions et de faits contenus dans le graphe est élevé, elle peut être coûteuse à calculer.

4.3.4 Sélection d'action

Le mécanisme de sélection d'actions cherche à trouver une suite d'actions à effectuer pour permettre la réalisation d'un but fixé. Plutôt que d'effectuer le calcul d'un plan complet permettant de réaliser un but, il cherche de manière incrémentale l'action qui lui semblera le mieux participer à la réalisation du but. Dès qu'une telle action sera trouvée, une requête d'exécution sera lancée et l'algorithme attendra l'information de terminaison de l'action avant de pouvoir choisir la prochaine. Les calculs seront donc répartis dans le temps en permettant la prise en compte des modifications perçues sur l'état de l'environnement durant la réalisation de l'action sélectionnée. Pour guider sa recherche, il utilise la mise en couche du graphe de planification comme heuristique. D'autre part, la notion de verrou va être utilisée, pour permettre de prendre en compte les interactions entre les actions et empêcher l'exécution des actions invalidant des faits nécessaires à la réalisation d'un but.

Dans un premier temps, nous allons décrire l'algorithme permettant de sélectionner une action en vue de satisfaire un but donné. Dans un deuxième temps, nous montrerons comment cette méthode de sélection est utilisée dans le cadre du mécanisme de sélection d'action.

Sélection d'une action. La sélection d'une action en vue de satisfaire un but s'effectue en utilisant la mise en couche du graphe de planification comme heuristique. Il est possible de considérer que le but b à atteindre est la racine d'un arbre d'actions qui convergent vers b. Les actions possédant ce but pour effet et dont le rang est inférieur ou égal au rang de b, peuvent alors être sélectionnées pour tenter de le satisfaire. L'ensemble des actions permettant de satisfaire b est fourni par la fonction R suivante :

$$R(b) = \{op \mid (b \in Add(op)) \wedge (rang(op) \leq rang(b))\}$$

Parmi les actions appartenant à $R(b)$, nous choisissons celle qui minimise le rang. Elle est donc fournie par la fonction sel suivante :

$$sel(b) = op,\ rang(op) = Min_{x \in R(b)} rang(x)$$

Dans le cas où plusieurs actions minimisent le rang, la première action trouvée est sélectionnée.

Nous sommes donc à même de choisir une action permettant de satisfaire un but en s'aidant de l'heuristique fournie par la mise en couche du graphe. Cependant, les pré-conditions de cette action ne sont pas forcément satisfaites. Les faits la constituant peuvent alors être interprétés comme un ensemble de sous-buts à satisfaire, par définition, dans l'ordre de leur déclaration.

```
action((b,v), sb)
début
    si rang(b) = ∞ alors
        retourne échec
    fin si
    si rang(sb) = ∞ alors
        retourne action((b,v),b)
    fin si
    op = sel(b)
    si Del(op) ∩ V ≠ ∅ alors
        miseEnCouche
        retourne action((b,v),sb)
    sinon si exe(op) alors
        retourne op
    fin si
    ∀f ∈ truePre(op), lock(f,v)
    retourne action((b,v),nextPre(op))
fin
```

FIG. 4.3 – *Algorithme de sélection d'action.*

L'algorithme de sélection d'action est présenté sur la figure 4.3. Il s'agit d'une méthode récursive, recherchant une action à exécuter en considérant le but comme la racine d'un arbre d'actions dont les feuilles sont exécutables. Le parcours de cet arbre s'effectue en s'aidant de la mise en couche du graphe comme heuristique. La fonction $action((b,v),sb)$ décrite prend en paramètre un but à réaliser b, son importance v ainsi qu'un sous but sb, non satisfait et sélectionné pour être réalisé. Voici un commentaire du fonctionnement de l'algorithme :

1. Dans un premier temps, une vérification est faite sur le rang du but pour vérifier qu'il est toujours atteignable. Si ce but n'est plus atteignable, il ne le sera jamais dans cette configuration du système. Dans ce cas un échec est renvoyé et le but est supprimé de l'ensemble des buts actifs qui le contenait.
2. Si le sous-but ne peut pas être atteint mais que le but est encore atteignable, il existe peut-être une autre suite d'actions pouvant le satisfaire. Dans ce cas, l'algorithme recommence une nouvelle sélection d'action : $action((b,v),b)$. Le fait de ne pas remettre à jour les verrous rend l'algorithme incomplet, mais cette incomplétude assure la stabilité. Cela invalide la sélection des actions (désormais de rang infini) ayant provoqué cet échec partiel, sans sélectionner d'actions invalidant les pré-conditions déjà verrouillées.
3. Dans le cas où le but b et le sous-but sb sont atteignables, une action $op = sel(sb)$ permettant de satisfaire sb est alors sélectionnée.

4. Plusieurs cas peuvent alors se présenter :
   (a) Les effets de l'action invalident des faits, déjà réalisés et appartenant à l'ensemble des verrous associés au but. Dans ce cas, l'action *op* n'est pas valide car elle entre en conflit avec ce qui a déjà été réalisé. Une nouvelle mise en couche est alors calculée, pour prendre en compte l'influence des verrous et donc remettre à jour l'heuristique. Les verrous, d'une importance équivalente à celle du but, permettent d'invalider les actions qui interfèrent avec les sous-buts déjà réalisés. Dès lors, une nouvelle sélection d'action est effectuée récursivement avec pour paramètres b et sb : le but ainsi que le sous-but que l'on cherche toujours à satisfaire.
   (b) Les effets de l'action ne provoquent pas de conflit et l'action est exécutable. Dans ce cas, l'action est sélectionnée pour être réalisée.
   (c) L'action n'est pas exécutable et ne provoque pas de conflit avec les verrous. Dans ce cas, toutes les pré-conditions vraies (dans l'ordre de leur description et en s'arrêtant à la première pré-condition fausse) sont verrouillées ; il s'agit des faits fournis par $truePre(op)$. Le nouveau sous-but à satisfaire devient alors $nextPre(op)$, l'algorithme est donc appelé récursivement avec b et $nextPre(op)$ comme paramètre, demandant ainsi la satisfaction du sous-but $nextPre(op)$.

Cet algorithme fonctionne donc d'une manière similaire aux algorithmes qui peuvent être utilisés en planification HTN. Cependant, il n'effectue pas explicitement de calcul de situation. Il se base sur un état courant et cherche, au travers de l'heuristique fournie par la mise en couche du graphe de planification, la prochaine action à sélectionner. Lors de la recherche de l'action, la comparaison entre les faits supprimés par l'action et les verrous posés permet de prendre en compte les incompatibilités qui ne sont pas détectées par la mise en couche.

Sélection du but et algorithme de sélection d'action. L'algorithme de sélection d'action se déroule en 4 phases :

1. Initialisation des activations et inhibitions.
2. Mise en couche du graphe.
3. Sélection du but à réaliser.
4. Sélection de l'action en vue de réaliser le but sélectionné, avec prise en compte des conflits par la pose de verrous dans V et une remise en couche si nécessaire.
5. Suppression des verrous de V.

La première phase de traitement initialise le système. La seconde phase calcule une première mise en couche du graphe de planification. Le but choisi pour être réalisé est alors le but possédant la plus grande propension à être réalisé, qui n'est pas encore réalisé et qui possède un rang spécifiant qu'il est atteignable. Ce but est sélectionné par l'intermédiaire de la fonction $g(f)$, qualifiant la propension à réaliser le but, définie comme suit [4] :

$$\forall f \in (B_r \cup B_m),\ g(f) = Min(sat(f),(Max_{(f,v)\in A(f)} v - Max_{(f,v)\in I(f)} v))$$

La valeur de $g(f)$ est donc définie comme le minimum entre $sat(f)$ et la valeur d'activation initiale du but. Elle permet de ne pas favoriser la réalisation d'un but si celui-ci est indirectement

4. $A(f)$ et $I(f)$ sont définis section 4.3.3 p. 204

activé par un but de plus grande importance. Notons s_{max} la plus grande valeur de $g(f)$ telle que f soit un but atteignable de B_r ou B_m :

$$s_{max} = Max_{\{(f,v) \mid ((f,v)\in(B_r\cup B_m))\wedge(rang(f)\neq\infty)\}} g(f)$$

L'ensemble des buts de plus grande importance est alors défini par :

$$G = \{f \mid \exists v,(f,v) \in (B_r \cup B_m) \wedge g(f) = s_{max} \wedge rang(f) \neq \infty\}$$

Pour assurer la continuité dans la sélection du but à réaliser, l'algorithme conserve le but (b_{t-1}) choisi lors de la dernière sélection d'action. Le nouveau but b_t est alors sélectionné comme suit :

- si $b_{t-1} \in G$ alors $b_t = b_{t-1}$.
- sinon, un but est tiré au hasard dans G pour pour être affecté à b_t.

L'action est alors choisie par l'intermédiaire de la fonction *action* (décrite dans la figure 4.3 p. 207) en utilisant le paramétrage suivant : $action((b_t,g(b_t)),b_t)$. Dans le cas d'un échec lors de la recherche d'une action, le but est considéré comme impossible à satisfaire. Il est donc supprimé de l'ensemble des buts et l'agent est prévenu de cet échec. Il peut alors prendre en compte le problème et choisir un nouveau but à réaliser. Une fois l'action sélectionnée ou l'échec détecté, les verrous sont supprimés, pour qu'ils n'interfèrent pas lors de la prochaine sélection d'action.

Cet algorithme est déroulé lors de chaque sélection d'action. Le principal goulet d'étranglement, du point de vue des performances, se situe dans la mise en couche du graphe de planification. En effet, ce calcul s'avère beaucoup plus coûteux que les appels récursifs de la fonction de sélection d'action. C'est la raison pour laquelle un nouveau calcul de mise en couche n'est effectué que lors de la détection d'une incompatibilité. Dans le pire des cas, ce calcul est effectué une fois par niveau d'appel récursif.

4.4 Discussion sur la sélection d'action

Dans cette section, nous allons donner un exemple du déroulement de l'algorithme de sélection d'action, en montrant l'influence de l'ordre lors de la description des buts évolués. Puis, nous discuterons du modèle de sélection d'action, de ses avantages et de ses inconvénients.

4.4.1 Influence de l'ordre dans les but évolués

Nous allons étudier un exemple concret, simplifié, permettant de mettre en évidence l'importance de l'ordre des buts. Pour ce faire, nous partons d'un ensemble de deux faits $\{a,b\}$, utilisés pour décrire les actions. Cet ensemble se décline, lors de son utilisation dans le moteur de sélection d'action, en un ensemble de huit faits : $\{a, \overline{a},K(a),\overline{K(a)},b,\overline{b},K(b),\overline{K(b)}\}$. Nous considérons alors deux actions possédant la définition suivante :

action	A		**action**	B	
Pre(A)	=	a	Pre(B)	=	$\overline{a}$, b
Eff(A)	=	$\overline{a}$	Eff(B)	=	a, $\overline{b}$
Add(A)	=	$\overline{a}$, $K(a)$	Add(B)	=	a, $\overline{b}$, $K(a)$, $K(b)$
Del(A)	=	a, $\overline{K(a)}$	Del(B)	=	$\overline{a}$, b, $\overline{K(a)}$, $\overline{K(b)}$
cout(A)	=	1	cout(B)	=	1

La situation initiale est caractérisée par la véracité de $\{a,\ b,\ K(a),\ K(b)\}$.

Nous allons comparer la réaction du système sur deux exemples. Le premier exemple consiste à fournir un but évolué de la forme $b_1 = \{\overline{b},\overline{a}\}$. Le second consiste à fournir le même but évolué mais, en inversant l'ordre des faits : $b_2 = \{\overline{a},\overline{b}\}$.

But $b_1 = \{\overline{b},\overline{a}\}$. La figure 4.4 montre le graphe de planification associé à cet exemple, dans lequel les cercles symbolisent les actions et les rectangles les faits. Les liens traduisent les effets contenus dans $Add(A)$ ou $Add(B)$, dans le cas d'un lien d'une action vers un fait. Les liens d'un fait vers une action, symbolisent les pré-conditions de l'action et sont étiquetés avec leur numéro d'ordre dans la liste des pré-conditions. L'action F et le fait f ajoutés, correspondent à l'action et au fait « fantômes » ajoutés pour la gestion des buts évolués. Il est à noter que, pour respecter la cohérence, le fait f se décline, comme tous les faits, en $\overline{f}$, $K(f)$ et $\overline{K(f)}$. Cependant, pour des raisons de lisibilité, ces faits ne seront pas présents dans les calculs et ne sont pas présentés dans le schéma.

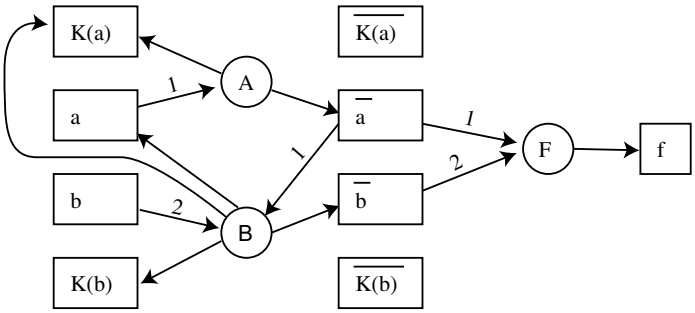


FIG. 4.4 – *Graphe de planification.*

Le fait à satisfaire devient donc le fait f, avec une valeur d'activation de 1. La table de la figure 4.5 résume les calculs de mise en couche effectués durant le traitement avec les valeurs de vérité ainsi que les activations/inhibitions associées à chaque fait. Il en est de même pour les actions, à l'exception des valeurs de vérité qui ne sont pas définies, mais dans leur cas, le symbole + indique l'action sélectionnée à chaque passe. Les arbres sous le tableau, montrent le déroulement de la sélection d'action. Chaque étage des arbres caractérise un appel récursif, lié à la demande de réalisation d'un sous but. Les faits grisés représentent les verrous posés durant l'exploration. L'algorithme de sélection d'action résout ce problème en trois passes, sélectionnant successivement les actions A,B puis de nouveau A. Lorsqu'une action est sélectionnée, elle est translatée en un automate HPTS++ qui est greffé sur l'agent. Lorsque l'action se termine, le mécanisme est rappelé pour choisir la prochaine action à exécuter. Une fois ces trois actions exécutées dans l'ordre, le but évolué b_1 est satisfait car l'action F est exécutable.

But $b_1 = \{\overline{a},\overline{b}\}$. De la même manière que pour l'exemple précédent, la figure 4.6 p. 212 présente le déroulement de l'algorithme. A la différence du déroulement pour le but b_1, lors de la seconde passe, une incompatibilité est détectée lors de la sélection de l'action B. En effet, elle invalide $\overline{a}$, qui est verrouillé. Une nouvelle mise en couche est alors effectuée, résumée par la passe 2.2 du tableau. Les verrous étant actifs, l'action B reçoit autant d'activation que d'inhibition. Elle devient donc

Faits et actions	Passe 1 vérité	(a,i)	rang	Passe 2 vérité	(a,i)	rang	Passe 3 vérité	(a,i)	rang
a	vrai	(1,0)	0	faux	(0,0)	1	vrai	(1,0)	0
$\overline{a}$	faux	(1,0)	1	vrai	(1,0)	0	faux	(1,0)	1
$K(a)$	vrai	(0,0)	0	faux	(0,0)	0	faux	(0,0)	0
$\overline{K(a)}$	faux	(0,0)	∞	vrai	(0,0)	∞	vrai	(0,0)	∞
b	vrai	(1,0)	0	vrai	(1,0)	0	faux	(0,0)	∞
$\overline{b}$	faux	(1,0)	2	faux	(1,0)	1	vrai	(1,0)	0
$K(b)$	vrai	(0,0)	0	vrai	(0,0)	0	vrai	(0,0)	0
$\overline{K(b)}$	faux	(0,0)	∞	faux	(0,0)	∞	faux	(0,0)	∞
A	+	(1,0)	1	-	(0,0)	2	+	(1,0)	1
B	-	(1,0)	2	+	(1,0)	1	-	(0,0)	∞
f	faux	(1,0)	3	faux	(1,0)	1	faux	(1,0)	1
F	-	(1,0)	3	-	(1,0)	1	-	(1,0)	1

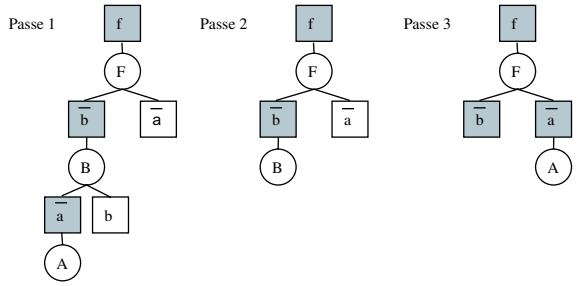


FIG. 4.5 – *Déroulement de l'algorithme pour le but* b_1.

verrouillée à son tour, lui attribuant un rang ∞. Ce rang se propage alors jusqu'au fait f, le rendant ainsi inaccessible. Un échec est alors détecté. Le but est supprimé et l'échec est transmis à l'agent.

Cet exemple traduit la sensibilité du système à l'ordre des pré-conditions des actions, tout comme à l'ordre des faits lors de la description des buts évolués. Les buts b_1 et b_2 sont potentiellement équivalents, cependant, en fonction de l'ordre des sous-buts, la situation devient réalisable ou non. Il s'agit là de la conséquence la plus néfaste de l'utilisation d'un ordre, qui est exactement celle de la planification HTN.

4.4.2 Un exemple concret

L'exemple précédent a montré l'influence de l'ordre des buts dans la sélection d'action. Dans cette section, nous allons nous intéresser à un exemple concret et montrer l'influence de la connaissance sur la sélection d'actions.

Nous définissons un monde constitué d'un acteur (*jean*) pouvant se déplacer dans une maison. Cette maison est constituée de quatre pièces : un salon, une cuisine, une chambre et un couloir. L'accès à toutes les pièces se fait par l'intermédiaire du couloir. Un verre est aussi présent dans la

Faits et actions	Passe 1 vérité	(a,i)	rang	Passe 2.1 vérité	(a,i)	rang	Passe 2.2 vérité	(a,i)	rang
a	vrai	(1,0)	0	faux	(0,0)	1	faux	(0,1)	∞
$\overline{a}$	faux	(1,0)	1	vrai	(1,0)	0	vrai	(1,0)	0
$K(a)$	vrai	(0,0)	0	faux	(0,0)	0	faux	(0,0)	0
$\overline{K(a)}$	faux	(0,0)	∞	vrai	(0,0)	∞	vrai	(0,0)	∞
b	vrai	(1,0)	0	vrai	(1,0)	0	vrai	(0,0)	0
$\overline{b}$	faux	(1,0)	2	faux	(1,0)	1	faux	(1,0)	∞
$K(b)$	vrai	(0,0)	0	vrai	(0,0)	0	vrai	(0,0)	0
$\overline{K(b)}$	faux	(0,0)	∞	faux	(0,0)	∞	faux	(0,0)	∞
A	+	(1,0)	1	-	(0,0)	2	-	(0,0)	∞
B	-	(1,0)	2	-	(1,0)	1	-	(0,0)	∞
f	faux	(1,0)	3	faux	(1,0)	1	faux	(1,0)	∞
F	-	(1,0)	3	-	(1,0)	1	-	(1,0)	∞

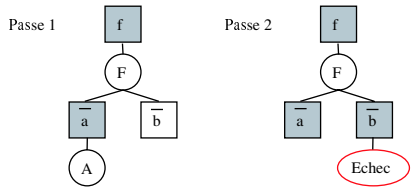


FIG. 4.6 – *Déroulement de l'algorithme pour le but b_2.*

maison. L'agent peut :

- se déplacer de pièce en pièce ;
- prendre le verre dans chaque pièce, il est à noter que cette action possède comme pré-condition la connaissance de la localisation du verre ;
- poser le verre dans chaque pièce ;
- regarder dans une pièce pour savoir si le verre s'y trouve ;
- remplir le verre à l'aide du lavabo de la cuisine.

Au départ, *jean* se trouve dans le salon et le verre dans la chambre. La description, en langage BCOOL, de cet exemple est disponible en annexe D p. 253. Nous allons présenter deux scénarios consistant à remplir le verre et mettant en jeu la connaissance de la localisation du verre.

Scénario 1. L'agent connaît la situation. La suite d'actions (décrites par leur nom, l'objet dont elles dépendent ainsi que leurs paramètres) suivante est alors générée par le mécanisme de sélection d'actions :

```
action(jean.gotoTo, salon, couloir)
action(jean.gotoTo, couloir, chambre)
action(verre.take, jean, jean.leftHand, chambre)
action(jean.gotoTo, chambre, couloir)
action(jean.gotoTo, couloir, cuisine)
action(cuisine.lavabo.fill, jean, jean.leftHand, verre, cuisine)
```

L'agent se déplace dans la maison, prenant le verre et allant le remplir au lavabo de la cuisine.

Scénario 2. L'agent ne sait pas où est le verre. Pour ce faire, la connaissance de la valeur de vérité des relations traduisant la localisation du verre sont initialisées à faux. La suite d'actions suivante est alors générée :

```
action(verre.watch, jean, salon)
action(jean.gotoTo, salon, couloir)
action(verre.watch, jean, couloir)
action(jean.gotoTo, couloir, cuisine)
action(verre.watch, jean, cuisine)
action(jean.gotoTo, cuisine, couloir)
action(jean.gotoTo, couloir, chambre)
action(verre.watch, jean, chambre)
action(verre.take, jean, jean.leftHand, chambre)
action(jean.gotoTo, chambre, couloir)
action(jean.gotoTo, couloir, cuisine)
action(cuisine.lavabo.fill, jean, jean.leftHand, verre, cuisine)
```

Dans la mesure où l'agent ne possède pas d'informations sur la localisation du verre, il regarde dans chaque pièce pour savoir si ce dernier s'y trouve. A chaque action de perception, sa connaissance est remise à jour, provoquant soit le déplacement, soit la prise du verre.

Les traces qui vont être présentées sont concises et ne listent que les actions sélectionnées ; des traces, plus précises, fournissant les actions exécutables à chaque sélection d'action ainsi que le rang du but sont disponibles en annexe D p. 253.

Cet exemple concret montre l'influence de la connaissance sur les suites d'actions que le mécanisme génère. Dans la mesure où la connaissance n'est pas suffisante, la sélection d'actions génère des actions perceptives en vue d'acquérir les informations nécessaires. Le comportement de recherche d'un objet, comme nous venons de le montrer, devient alors une conséquence intrinsèque de l'utilisation de la connaissance.

4.4.3 Avantages et inconvénients

Taille de la base de données. Comme nous le décrivions lors de la génération du monde, au travers de BCOOL, la taille de la base de données est polynomiale. Cette taille conditionne donc grandement la rapidité de calcul du mécanisme de sélection d'action. Cependant, en étudiant les modèles de planification ainsi que les mécanismes de sélection d'action, il semblerait que ce problème puisse difficilement être évité. Comme dans presque tous les mécanismes de sélection d'action, nous utilisons une instanciation complète du monde. Pour leur part, les logiciels de planification sont eux aussi soumis à ce problème de mémoire lors de leur recherche d'un plan. Ce qui peut d'ailleurs les conduire à une saturation totale de la mémoire, ayant pour conséquence un échec de la planification. Le fait de conserver la base de données complètement instanciée s'avère certes coûteuse mais une fois cette mémoire allouée, le déroulement de l'algorithme ne nécessite pratiquement pas d'espace mémoire supplémentaire assurant ainsi une stabilité lors des calculs.

Prise en compte des connaissances. L'un des gros avantages de ce système est la prise en compte des connaissances. Il est possible de décrire des actions perceptives au même titre que des actions effectives. Si une connaissance fait partie des pré-conditions d'une action, le système exécutera automatiquement des actions de perception (si ces dernières existent) en vue de l'acquérir. De ce fait, le système accepte l'évolution dynamique de la connaissance. A chaque nouvelle sélection d'action, la connaissance acquise est prise en compte, permettant l'exploitation des opportunités offertes par l'environnement au travers d'une remise en cause continuelle du plan appliqué. Enfin, il est possible de gérer l'évolution de la confiance que l'agent peut avoir dans sa propre connaissance, modélisant ainsi l'évolution des croyances. Cette évolution des croyances de l'agent gouverne le comportement d'acquisition d'information. Actuellement, cette possibilité est offerte, mais n'est pas gérée directement par le système de sélection d'action ou pris en compte dans BCOOL. La raison en est que nous pensons que chaque agent doit pouvoir être doté de son propre système d'évolution des croyances. Ce sujet est un sujet de recherche à part entière, devant s'inspirer des résultats sur la mémoire issus des sciences cognitives. Notre choix est donc de laisser le système ouvert sur ce point pour faciliter dans le futur la connexion d'un modèle d'évolution des croyances.

Ordre des pré-conditions et buts évolués. La notion d'ordre de faits dans les pré-conditions et les buts évolués est à la fois une qualité et un défaut. Cette propriété permet d'introduire un biais dans la description en introduisant une connaissance experte dans l'ordre de satisfaction des faits. Cela peut conduire à des comportements semblant plus cohérents car plus naturels dans l'ordre dans lequel les actions sont effectuées. Cependant, l'introduction de cet ordre peut conduire le système à ne pas trouver de solution, contrairement à des algorithmes ne possédant pas cette contrainte. D'autre part, la détection d'un échec est faite au dernier moment, rendant inutiles toutes les actions qui ont pu être réalisées précédemment. Il s'agit là du prix à payer pour obtenir un système relativement rapide permettant la prise en compte continuelle des changements de l'environnement et générant le plan de manière incrémentale.

Les buts atomiques. Les buts de type *maintain* et *avoid*, s'avèrent très utiles pour traduire des effets de la personnalité de l'agent sur ses actions. Il est possible de créer l'agent maniaque, ayant du mal à supporter que le pot de fleur ne soit pas sur le guéridon du salon. Ce comportement se traduit aisément par un but de type *maintain*. D'autre part, avec les buts de type *avoid*, il est possible d'empêcher l'agent de réaliser quelque chose. Par exemple, un agent ayant peur des araignées n'en prendra jamais une dans ses mains (à moins d'y être vraiment obligé). Lors de la phase de répartition des activations et inhibitions, ces buts jouent un grand rôle et peuvent conduire à un échec dans la réalisation d'un plan, car l'agent ne peut surpasser ses inhibitions. L'utilisation de ces buts, permet de traduire de manière factuelle et en terme d'influence directe sur le comportement un certain nombre de traits de personnalité. Plutôt que d'utiliser un modèle de profil psychologique associé aux actions [Mon02], il devient possible de décrire des profils psychologiques en terme d'influence sur le comportement et plus précisément en terme de ce que l'agent a tendance ou n'a pas tendance à faire. Ce type de modèle permettra la prise en compte automatique dans le modèle de sélection d'action du profil psychologique, sans pour autant avoir spécialisé chaque action.

Stabilité. En régime normal, le système est stable et n'oscille pas entre plusieurs actions. Cette stabilité est due à plusieurs facteurs. D'une part la notion d'ordre introduite dans la gestion des pré-conditions et des buts évolués permet de contraindre la recherche en posant des verrous, qui

empêchent les oscillations en bloquant l'insatisfaction d'un fait utile et déjà réalisé. Ce blocage est assuré car le verrou possède une importance équivalente à celle du but sélectionné, qui lui même maximise cette importance. D'autre part, lors de la sélection du but à réaliser, la conservation du but précédent assure la continuité temporelle. Enfin, la sélection du but ne dépend pas de sa distance à la situation courante, ce qui constitue le facteur d'instabilité du mécanisme de sélection d'action de Maes [Mae90] par exemple. Cependant, dans le cas d'une confiance dans la connaissance qui varie trop vite, ou dans le cas d'un environnement dont les valeurs de vérité oscillent, indépendamment de l'agent, cette stabilité n'est pas assurée, et ne peut l'être.

Conclusion

Dans ce chapitre, nous avons présenté un langage de programmation dédié à la description du monde cognitif de l'agent. Il effectue le lien entre les actions « abstraites » et les comportements adoptés par l'agent. Ce langage permet de capitaliser les descriptions des mondes, de les réutiliser, de les enrichir, avec les mêmes caractéristiques que celles associées à la modélisation objet. Il permet, suivant en cela les théories de Gibson, de centrer la description sur ce que le monde offre à l'agent et non plus sur ce que l'agent sait faire. Cette propriété permet de limiter automatiquement le champ d'action de l'agent en le corrélant automatiquement à ce que son environnement lui autorise. Enfin, il permet de gérer l'acquisition des connaissances en mettant sur le même plan les actions perceptives et les actions effectives. Cela facilite la gestion de la perception qui est alors automatiquement corrélée aux intentions de l'agent. L'agent choisit des actions en fonction de la connaissance qu'il désire acquérir. Le mécanisme de sélection d'actions qui lui est associé, prend en compte la dynamique du monde, et accepte des mise-à-jours externes des connaissances. Cela lui procure une réactivité par rapport aux changements perçus de l'environnement, tout en offrant de grandes opportunités quand à l'intégration d'un modèle de mémoire et de confiance dans la connaissance. Enfin, le système de gestion de buts permet de contraindre les actions de l'agent en décrivant des inhibitions et des activations sous la forme de conséquences de ses traits psychologiques sur les actions qu'il peut ou non effectuer. La formalisation de ce modèle, en fonction d'études sur le comportement, fait partie des extensions futures. Le but étant à plus long terme d'automatiser la prise en compte du profil psychologique de l'agent au travers de son influence sur le comportement.

Chapitre 5

Des outils au service d'une architecture

Les outils et algorithmes qui ont été présentés dans les chapitres précédents sont indépendants d'une plate-forme de simulation. Pour être à même de réaliser des démonstrations et d'exploiter les divers développements réalisés au sein de l'équipe SIAMES, ils ont été insérés à l'intérieur de la plate-forme OpenMASK [MAC+02]. Dans ce chapitre, nous allons présenter l'architecture globale de gestion du comportement des humanoïdes au travers de son intégration dans la plate-forme. Dans un second temps, nous présenterons des réalisations qui ont été effectuées sur la base des outils présentés dans cette thèse :

- une démonstration de visite de musée virtuel, en exposition permanente à la cité des sciences et de l'industrie de la Villette depuis juin 2001,
- une démonstration de fiction interactive présentée au festival Imagina 2002,
- une première réalisation d'un système de cinématographie autonome se basant sur les propriétés d'HPTS++.

5.1 Architecture des humanoïdes et intégration dans OpenMASK

Les chapitres précédents ont présenté les différentes briques de l'architecture proposée pour la gestion du comportement des humanoïdes virtuels. Dans cette section, nous allons discuter de l'intégration de ces composants dans la plate-forme OpenMASK au travers de l'environnement logiciel et de l'architecture d'exécution retenue. Nous décrirons plus précisément la connexion entre le mécanisme de sélection d'actions et le modèle HPTS++ qui, par manque de temps, n'est pas encore effective mais uniquement spécifiée.

5.1.1 Environnement logiciel

La figure 5.1 présente l'insertion des composants logiciels de l'architecture dans la plate-forme OpenMASK ainsi que les niveaux d'intervention de l'utilisateur. L'ensemble des bibliothèques propres à HPTS++, à BCOOL, à la subdivision spatiale, à la planification de chemin, au graphe de voisinage et à la navigation réactive sont indépendantes de la plate-forme de simulation, tout comme le code généré par les deux compilateurs. Ces bibliothèques fournissent l'intégralité des structures de données nécessaires à la gestion des différents algorithmes. Parmi les composants que nous avons proposé, seuls les automates HPTS++, le mécanisme de sélection d'actions de BCOOL et le graphe

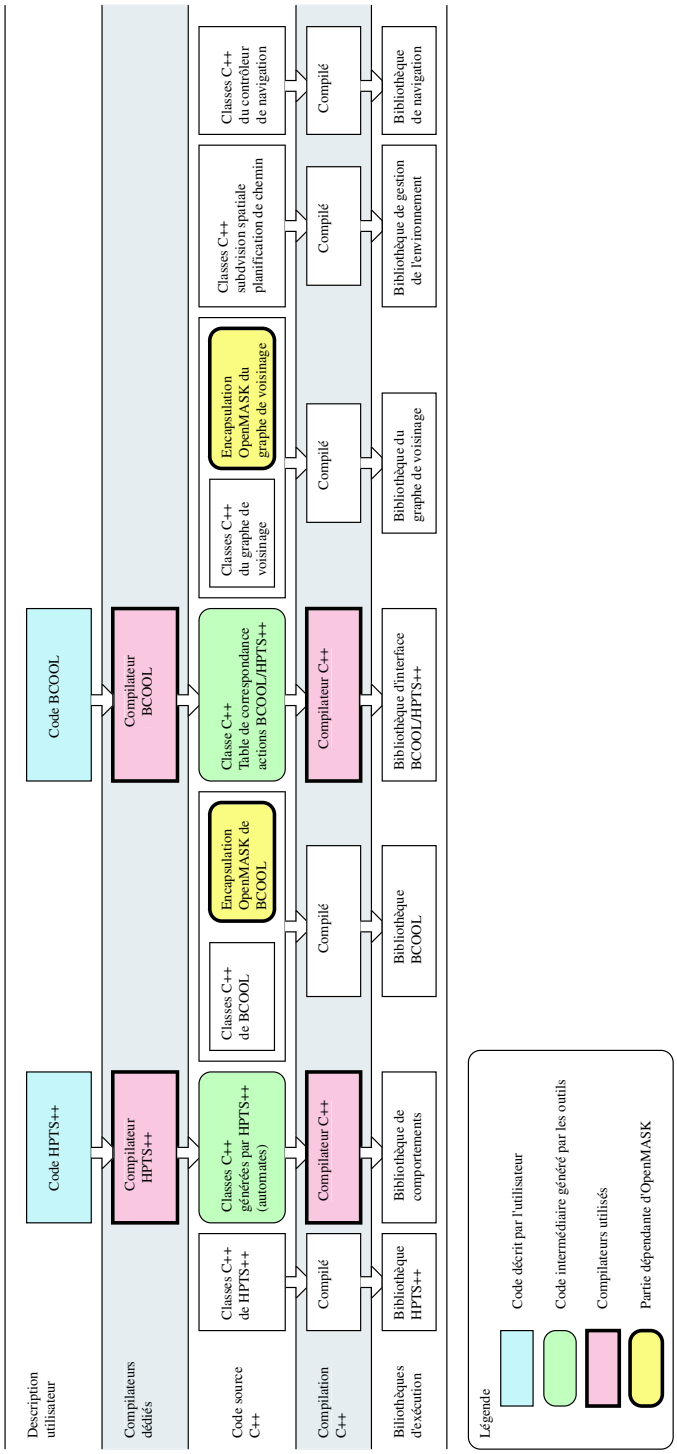


FIG. 5.1 – *Insertion de l'architecture dans OpenMASK et interventions de l'utilisateur.*

de voisinage sont des « entités » actives nécessitant une remise à jour régulière de leurs informations ; les autres composants peuvent être vus comme des prestataires de service ne fournissant une information que sur demande d'un mécanisme de contrôle (généralement HPTS++).

Afin d'insérer les composants à l'intérieur de la plate-forme, deux encapsulations ont été nécessaires pour :

- le mécanisme de sélection d'actions de BCOOL,
- le graphe de voisinage.

Ces deux encapsulations permettent de transformer ces composants en objets de simulation OpenMASK qui pourront se voir attribuer une fréquence de calcul cadençant le rafraîchissement des informations qu'ils fournissent.

Aucun lien apparent n'existe pour l'intégration des automates HPTS++ à l'intérieur de la plate-forme. Cela s'explique par la nature du langage qui lui est associé. En effet, il offre une grande souplesse d'emploi de par son utilisation du C++. Les automates décrits sont transformés en classes C++ et peuvent hériter d'automates ou de classes. Cette propriété permet d'effectuer le lien avec la plate-forme à l'intérieur du langage lui-même, sans avoir à offrir à l'utilisateur des composants pré-décrits. Ce lien s'effectue par héritage d'une classe particulière ; la création d'un automate héritant de cette classe est donc suffisante pour créer un objet de simulation [1]. Cependant, dans le cadre de nos applications, nous disposons d'un automate de comportement par défaut pour les humanoïdes (démarrage de mouvements de repos, battements de paupières, configuration des modules d'évitement de collision...) qui effectue le lien avec le pilote de l'humanoïde qui est lui-même un objet de simulation OpenMASK [Men03]. Par héritage de cet automate, il est possible de définir de nouveaux humanoïdes en enrichissant ou en changeant ce comportement.

Lors de la description des comportements associés aux humanoïdes l'utilisateur n'intervient plus qu'en deux points :

- la description des automates de comportement qui permettent d'enrichir la bibliothèque de comportements propre aux humanoïdes,
- la description du monde cognitif au travers de BCOOL pour générer un nouvel environnement et décrire les moyens d'interaction en effectuant la connexion avec HPTS++.

Ces deux descriptions génèrent du code C++ qui doit être compilé sous la forme de bibliothèques pour permettre leur utilisation à l'intérieur d'une simulation.

La figure 5.2 décrit les dépendances entre les différents modèles présentés. Il est possible de distinguer deux niveaux de dépendance :

- le niveau associé à la navigation et à la représentation de l'environnement,
- le niveau associé au comportement.

La dépendance entre le niveau comportemental et le niveau associé à la navigation s'effectue par l'intermédiaire d'un automate HPTS++ qui est en charge de la gestion du comportement de déplacement.

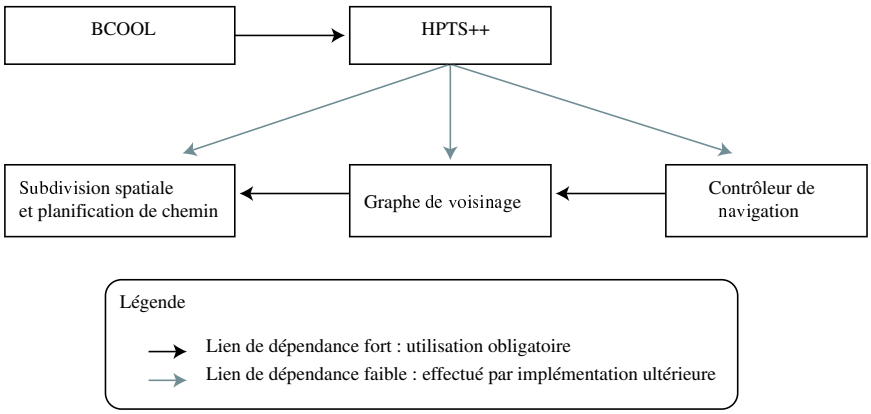


FIG. 5.2 – *Graphe de dépendance entre les composants logiciels.*

5.1.2 Architecture d'exécution dans OpenMASK

La figure 5.3 résume l'organisation d'une application utilisant les modèles présentés pour la gestion du comportement des humanoïdes virtuels. Dans le cadre de l'intégration dans la plate-forme OpenMASK (voir section 3.3 p. 81), deux types d'objets sont utilisés :

- les *objets de simulation* qui sont caractérisés par une fréquence de calcul et éventuellement un déphasage.
- les *objets de service* qui sont inactifs par défaut et ne sont activés que sur une requête d'un objet simulé.

Nous allons successivement présenter l'intégration des différents composants de l'architecture dans cette plate-forme. Comme nous allons le voir, les propriétés sur les fréquences sont utilisées lors des simulations pour limiter le coût de calcul.

Graphe de voisinage. Le graphe de voisinage est calculé par un objet de simulation OpenMASK. Cet objet se voit attribuer une fréquence de calcul lors du démarrage de l'application, définissant la fréquence de rafraîchissement des informations de voisinage. L'utilisation d'une fréquence de l'ordre de 5 à 10Hz s'avère suffisante pour prédire finement les collisions. Lorsque l'humanoïde est créé, il se référence pour signaler qu'il constitue un obstacle dynamique potentiel. Ce graphe n'est pas uniquement réservé aux humanoïdes, il peut être utilisé pour référencer toute forme d'obstacle ponctuel gênant la navigation (tabouret, poteau...) ou d'autres formes d'entités dynamiques. Il offre une structure unifiée, permettant d'accéder à une information uniformisée contenant le rayon, la vitesse courante et la position des obstacles ponctuels (en déplacement ou non).

Objets informés. Les objets informés sont des objets de simulation OpenMASK possédant par défaut une forme géométrique, une position spatiale et une orientation. Ils contiennent aussi des informations dédiées au comportement. Elles peuvent être utilisées dans le cadre de comportements d'interaction pour obtenir des coordonnées pour la préhension, pour savoir comment regarder l'objet ou bien encore pour placer l'humanoïde relativement à l'objet. Par exemple, un canapé possédera

1. Cette propriété est aussi utilisée pour créer des objets actifs, autres que des humanoïdes, contrôlés par l'intermédiaire d'automates HPTS++.

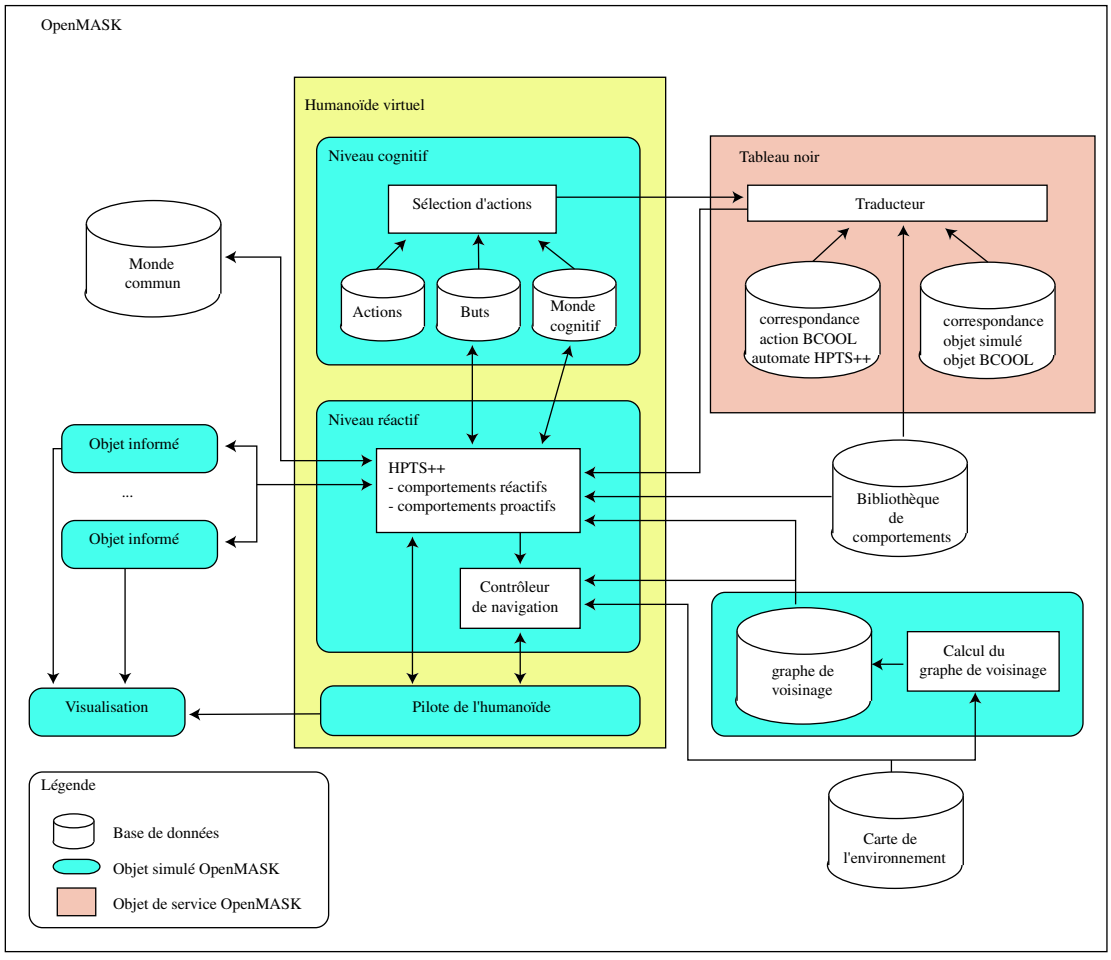


FIG. 5.3 – *Architecture d'une application utilisant les agents cognitifs.*

des informations sur la position et l'orientation à adopter par l'humanoïde pour pouvoir s'asseoir. Ces objets ne sont pas forcément passifs, il peuvent être dotés d'un comportement décrit par l'intermédiaire d'HPTS++. A titre d'exemple, une porte automatique peut être définie comme un objet informé actif, surveillant la proximité des humanoïdes par l'intermédiaire du graphe de voisinage. La porte s'ouvre alors automatiquement lorsqu'elle détecte quelqu'un à une certaine distance.

Architecture de l'humanoïde. L'humanoïde se scinde en trois objets de simulation : le niveau cognitif, le niveau réactif, le pilote humanoïde ; cela permet d'exploiter les propriétés sur les fréquences de calcul de la plate-forme. Les trois niveaux peuvent alors fonctionner à des fréquences différentes, en utilisant le déphasage pour répartir la charge de calcul d'un pas de temps à l'autre. A titre d'exemple, le pilote humanoïde, en charge de l'animation, possède généralement une fréquence de calcul de l'ordre de 25 à 30 Hz, comme la visualisation. Pour sa part, le niveau réactif ne nécessite que rarement des fréquences de calcul supérieures à 10 Hz (ce qui correspond au niveau *action délibérée* de la hiérarchie définie par Newell, voir figure **??** p. **??**), enfin le niveau cognitif peut être cadencé à une fréquence de 1 à 5 hertz (correspondant au niveau *opération* de la hiérarchie de Newell). L'utilisation de cette propriété permet d'optimiser les simulations en contrôlant la charge de calcul associée à chaque niveau de contrôle de l'humanoïde et permet d'obtenir une sorte de niveau de détail fréquentiel.

Le niveau réactif est le cœur de l'architecture. Il gère les comportements réactifs tels que la perception passive par exemple, ainsi que les comportements proactifs issus du mécanisme de sélection d'actions. Ces divers comportements sont responsables de :

- l'envoi des commandes d'animation au pilote de l'humanoïde,
- de requêtes sur la base de données du monde commun pour mettre à jour le monde cognitif de l'agent (perception active/passive),
- de la vérification de l'état du monde commun durant la réalisation des comportements proactifs pour s'assurer de la véracité des hypothèses qui ont pu être faites à partir de l'état du monde cognitif,
- de la mise à jour des buts fournis au mécanisme de sélection d'action,
- de l'évolution temporelle des paramètres internes (faim, soif...),
- de la récupération des informations d'interaction avec les objets informés,
- de l'envoi de l'information de réussite ou d'échec du comportement au niveau cognitif.

La phase de vérification de l'état du monde commun durant la phase de réalisation est importante. D'une part, elle permet d'assurer la cohérence des comportements en vérifiant la véracité des hypothèses éventuelles, d'autre part, elle permet d'assurer la cohérence globale du système en empêchant les interférences entre les comportements adoptés par plusieurs humanoïdes en interaction. Si une différence est constatée entre l'état du monde commun et le monde cognitif de l'agent, des mesures doivent être prises pour remettre à jour le monde cognitif, envoyer une information d'échec et donc demander une nouvelle sélection d'actions à partir du monde cognitif à jour.

L'ensemble des comportements provient de la bibliothèque de comportements qui fournit toutes les classes d'automates génériques décrites par l'intermédiaire du langage d'HPTS++. Ces automates représentent des comportements utilisant les système de ressources pour la synchronisation ainsi que les degrés de préférence pour l'adaptation. Les fonctions de priorité qui leur sont associées sont fournies en paramètre de construction ; il devient alors possible de leur associer dynamiquement une fonction qui permet de corréler la priorité du comportement à la cause de son exécution.

Le niveau cognitif possède un mode d'exécution particulier. Il est représenté par un objet de simulation OpenMASK qui se synchronise avec un processus léger responsable de la sélection d'actions. Le temps de réponse du mécanisme de sélection d'actions dépend grandement de la taille des bases de données d'actions et du monde cognitif. Si ces bases sont grosses, le temps de sélection augmente, ne permettant pas toujours d'assurer une animation fluide. L'utilisation d'un processus léger permet de répartir la charge de calcul sur plusieurs pas de temps, indépendamment de la fréquence de calcul associée à l'objet simulé. Le mécanisme de sélection d'action peut être dans deux états : actif s'il est en cours de sélection d'action, inactif si une action a été sélectionnée et que le mécanisme est en attente de réussite ou d'échec de cette même action. Cet état conditionne sa synchronisation avec son objet de simulation qui possède quatre rôles :

1. Il débloque le calcul d'une sélection d'action lorsque l'action précédemment exécutée envoie un message de réussite ou d'échec.
2. Il accumule les modifications de l'état du monde cognitif de l'agent et des buts tant que le mécanisme de sélection d'action est actif.
3. Il met à jour les buts et le monde cognitif lorsque le mécanisme de sélection d'action est inactif.
4. Il demande la traduction d'une action sélectionnée par le mécanisme de sélection d'action au tableau noir.

Tableau noir. Le tableau noir est un objet de service OpenMASK. Son rôle est d'établir le lien entre le monde cognitif des humanoïdes et le monde des objets simulés (objets informés, agents...). Il permet de traduire les actions abstraites du mécanisme de sélection d'actions en actions concrètes sous la forme d'automates. Il contient deux tables de correspondance :

1. Une table de correspondance effectue le lien entre les objets nommés du monde cognitif et les instances d'objets de simulation correspondantes. Par exemple, un verre possède une instance comme objet simulé possédant une position dans l'espace et une représentation géométrique, ainsi qu'une instance dans le monde cognitif des agents pour leur permettre de raisonner sur son utilisation.
2. Une table de correspondance met en relation les actions BCOOL et les actions décrites par l'utilisateur sous la forme d'automates HPTS++. Cette table est générée lors du processus de compilation de BCOOL.

Lors d'une demande de traduction, le tableau noir crée une instance de comportement en utilisant la table 2 et transforme les paramètres en utilisant la table 1. Le comportement ainsi construit est connecté à l'agent cible de l'action en demandant son exécution au contrôleur HPTS++ gérant l'exécution de ses comportements. La priorité de ce comportement est directement corrélée à l'importance du but qu'il contribue à satisfaire. Une fois la demande d'exécution effectuée, ce comportement entre en concurrence avec les autres comportements actifs. Grâce au mécanisme d'ordonnancement, tout ces comportements sont assurés de la cohérence de leur réalisation.

Parallélisme des actions. Lorsqu'une action est sélectionnée par le mécanisme de sélection d'actions, ce dernier reste en attente d'un message de réussite ou d'échec pour effectuer une nouvelle sélection. Par défaut, il ne génère donc pas plusieurs actions se déroulant en parallèle. Cependant, il est à noter que l'envoi du message provoquant une nouvelle sélection n'est pas dépendant de la terminaison du comportement. Le comportement proactif peut donc continuer son exécution après l'envoi du message. Sa seule contrainte est d'envoyer ce message lorsque l'action a eu les conséquences escomptées ou a été un échec. En prenant un exemple simple, si le but de l'agent est de fumer, ce

but est réalisé lorsque l'agent possède une cigarette allumée dans la main et est en train de fumer. Cette situation provoque donc l'envoi d'un message de réussite. Cependant, dans cet état de fait, une nouvelle sélection débute alors que l'agent est toujours en train de fumer, provoquant ainsi lors de la prochaine sélection d'action un parallélisme de plusieurs actions proactives.

L'état d'avancement en terme de développement ne nous permet pas, pour le moment, d'effectuer le lien entre le niveau réactif et le niveau cognitif. Le tableau noir est spécifié mais n'est pas encore implémenté. Cependant, cela ne remet pas en cause la validité de l'architecture. Dans son utilisation couplée à OpenMASK, elle exploite les propriétés fréquentielles de la plate-forme pour maîtriser le coût de calcul tout en tentant de respecter des fréquences propres à l'animation mais aussi au temps de réaction des divers niveaux de contrôle. Son aspect modulaire permet d'enrichir les comportements et d'adopter une approche de développement incrémentale, orientée objet, en autorisant la réutilisation des modules dans des contextes divers.

5.2 Exemples d'applications

Deux grandes démonstrations grand public ont permis de valider une grande partie des travaux réalisés durant cette thèse : l'application du musée virtuel, en démonstration permanente à la cité des sciences et de l'industrie de la Villette, et une démonstration de fiction interactive présentée au festival Imagina 2002. Nous allons présenter ces deux applications et, dans un dernier temps, nous présenterons une nouvelle utilisation d'HPTS++ au travers de l'architecture d'un système de cinématographie semi-autonome dont le besoin a été mis en évidence lors de la réalisation de la fiction interactive.

5.2.1 Le musée virtuel

Le musée virtuel a été la première démonstration à utiliser l'outil HPTS++. Elle constitue une refonte de la démonstration de la thèse de F. Devillers [Dev01]. Elle a été réalisée pour la cité des sciences et de l'industrie de la Villette pour présenter au grand public les travaux effectués sur l'animation comportementale ; elle fait désormais partie de l'exposition permanente sur l'image qui a été inaugurée en juin 2001. Sa réalisation a été confiée à un ingénieur expert (J. F. Taille) et nous sommes intervenus dans le cadre de la conception des comportements et sur la formation à HPTS++ et au système de ressources.

L'action se situe dans un musée virtuel contenant une exposition de tableaux et de statuettes. Ce musée est peuplé d'entités autonomes possédant des rôles divers :

- des humanoïdes autonomes visitent le musée ;
- un groupe de visiteurs effectue une visite menée par un guide ;
- une mère suit la visite guidée et surveille son enfant qui a tendance à s'éclipser pour aller toucher les œuvres ;
- un photographe tente de prendre des photographies mais dégrade les œuvres à chaque prise de vue ;
- une personne restaure les œuvres dégradées lorsque le public est mécontent de la qualité de l'exposition ;
- une voleuse tente discrètement de voler des statuettes ;

Entité concernée	Paramètres	Influence sur le comportement
Fils	Turbulence Vitesse	Abandonne sa mère pour toucher les œuvres. Vitesse adoptée pour s'enfuir devant le gardien ou sa mère.
Mère	Intérêt pour la visite Surveillance du fils Vitesse	Suit la visite guidée et ne surveille pas son fils. Vérifie que son fils est présent, le poursuit et le fâche s'il n'est pas a côté d'elle. Vitesse adoptée pour poursuivre son fils.
Voleuse	Prudence Vitesse	Surveille attentivement autour d'elle avant d'effectuer un vol. Vitesse adoptée pour s'enfuir devant le gardien.
Photographe	Prudence Vitesse	Surveille attentivement autour de lui avant de prendre une photographie. Vitesse adoptée pour s'enfuir devant le gardien.
Gardien	Attention enfant Attention photographe Attention voleuse Vitesse	Surveille attentivement l'enfant. Surveille attentivement la voleuse. Surveille attentivement le photographe. Permet de limiter son champ d'action sur la voleuse, l'enfant et le photographe.
Les visiteurs	Niveau de désordre	Modifie la réaction des visiteurs devant les œuvres endommagées ou volées. Ils se plaignent au gardien et font de grands gestes.

FIG. 5.4 – *Résumé des principaux paramètres du musée virtuel et de leur influence.*

- un gardien surveille le musée et peut intervenir auprès de la voleuse, du photographe ou de l'enfant ;
- des suribirds (sortes d'oiseaux au comportement de suricat) possèdent leur nid dans le musée ; ils peuvent voler dans la salle et sont effrayés par le public.

Les différents comportements peuvent être paramétrés par l'intermédiaire d'une interface graphique, accessible aux visiteurs de la cité des sciences et de l'industrie. La figure 5.4 présente un résumé des principaux paramètres et de leur influence sur la vie du musée.

La figure 5.5 présente l'architecture globale de l'application. Cette application est répartie sur deux machines distinctes :

- Un PC sous Windows gère l'interface graphique pour le paramétrage du comportement et joue aussi le rôle de serveur sonore.
- Un PC sous Linux gère l'application sous OpenMASK. Cette application reçoit les modifications effectuées par l'intermédiaire de l'interface graphique et met à jour les paramètres associés aux humanoïdes virtuels. Cela permet de tester immédiatement l'impact des changements de paramètres sur la vie de musée. Lors de l'émission d'événements sonores, un message est transmis au serveur sonore qui effectue la restitution.

Deux types d'objets informés sont disponibles : les œuvres du musée (statuettes et tableaux) et la carte de l'environnement. Les statuettes, qui peuvent être volées par la voleuse et touchées par

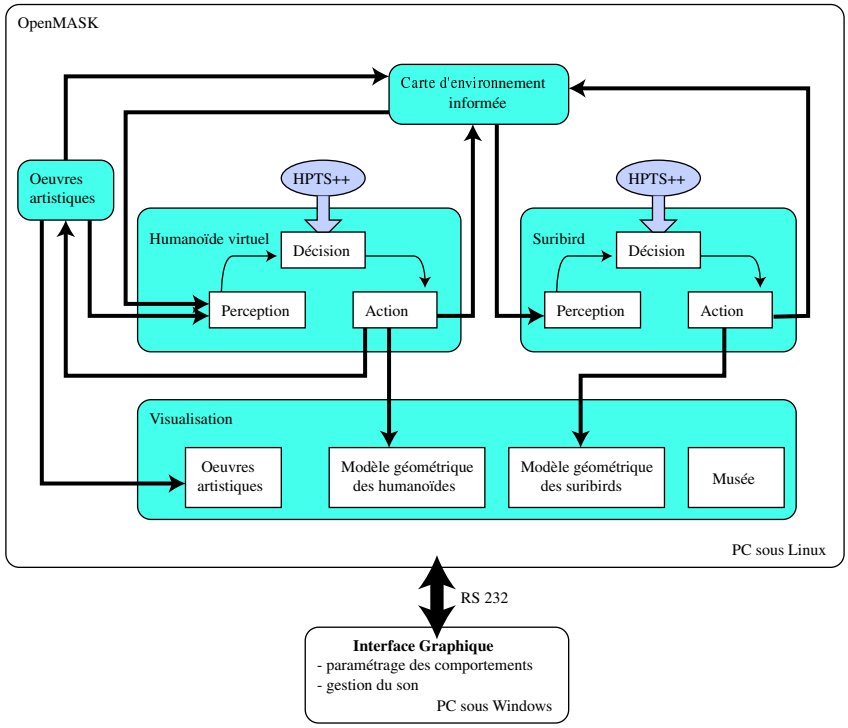


FIG. 5.5 – *Architecture de l'application du musée.*

l'enfant, fournissent les coordonnées de préhension ainsi que des positions idéales de saisie. Les tableaux et les statuettes fournissent aux visiteurs un certain nombre de positions à adopter pour les regarder ainsi qu'un certain nombre de cibles pour la gestion de l'attention visuelle. L'environnement a été représenté par l'intermédiaire d'une grille régulière (le système de subdivision spatiale présenté dans cette thèse n'était pas encore disponible en 2001). Cette grille contient les informations sur les obstacles ainsi qu'une référence sur l'œuvre qui se trouve éventuellement à cette position et une référence sur l'humanoïde peuplant la case si cette case correspond à une zone de navigation. Cette grille est aussi utilisée pour la détection de collision et la perception visuelle.

Le système de ressources d'HPTS++ a été utilisé pour gérer la concurrence des comportements associés aux personnages principaux. Les divers paramètres fournis par l'intermédiaire de l'interface graphique sont pris en compte dans le paramétrage des fonctions de priorité associées aux automates. Cette propriété permet de manipuler les comportements à haut niveau, en laissant le système d'ordonnancement gérer automatiquement la concurrence pour produire un comportement émergent cohérent. La réalisation de cette démonstration a permis de mettre en avant les atouts de ce mode de conception au travers d'un développement incrémental dans lequel l'ajout d'un nouveau comportement ne nécessite pas de prise en compte explicite des comportements déjà décrits.

La figure 5.6 présente des prises de vue de l'application. L'image en haut à gauche montre la guide effectuant la visite et l'image en haut à droite présente le groupe de visiteurs écoutant ses explications devant un tableau. L'image en bas à gauche présente la voleuse à la recherche d'une œuvre à voler. Enfin, l'image en bas à droite présente un vue plus générale du musée et de son

FIG. 5.6 – *Extraits de la démonstration du musée virtuel de la Cité des Sciences et de l'Industrie.*

activité. Le musée est peuplé d'environ trente humanoïdes autonomes et d'une dizaine de suribirds. Cette application fonctionne en temps réel sur une machine équipée de deux processeurs pentium 4 cadencés à 1 GHz et d'une carte graphique de type Quadro 2. Cette démonstration est désormais en exposition à la cité des sciences et fonctionne en continu tous les jours ouvrables.

5.2.2 Fiction interactive : tranche de vie à l'épicerie

Cette section présente notre première mise en œuvre d'une fiction interactive dont la présentation a été effectuée lors du festival Imagina 2002. Dans un premier temps, nous allons définir ce qu'est une fiction interactive. Dans un deuxième temps, nous présenterons le scénario de « Tranche de vie à l'épicerie » en fournissant quelques détails sur l'architecture de l'application et sur son exploitation des travaux réalisés durant cette thèse.

La fiction interactive désigne une forme de narration dans laquelle un utilisateur est amené à participer à l'histoire [MS02, CCM02]. Il peut influer directement ou indirectement sur son déroulement, soit en intervenant de manière ponctuelle, soit en devenant acteur. Elle se déroule sur la base d'un scénario qui décrit une histoire non linéaire dans laquelle des choix sont effectués en fonction des interactions de l'utilisateur et de l'évolution des personnages. Généralement, un scénario se découpe en plusieurs actes qui sont reliés par un nœud dramatique, traduisant un moment clef de l'histoire (voir figure 5.7). Les personnages sont des acteurs semi-autonomes. Ils sont dirigés à haut niveau par le scénario et leur autonomie leur offre une adaptation au contexte et notamment aux interventions de l'utilisateur. Les applications de la fiction interactive couvrent un champ d'applications s'étendant du domaine du jeu vidéo aux simulations immersives.

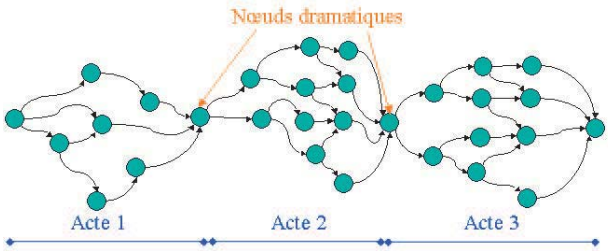


FIG. 5.7 – *Exemple de la structure d'un scénario d'une fiction interactive non linéaire.*

La fiction interactive « Tranche de vie à l'épicerie » se base sur un scénario mettant en jeu deux acteurs semi-autonomes : Jean et Sara (voir figure 5.8). Jean décide de se rendre à l'épicerie pour y faire ses achats. Il y rencontre Sara, une amie d'enfance dont il était amoureux. Il discutent et Jean, pris d'une certaine nostalgie, invite Sara chez lui pour prendre un apéritif et discuter de leur vie actuelle. Ne s'étant jamais remis de la séparation, il tente de la reconquérir.

FIG. 5.8 – *Présentation de la démonstration. De gauche à droite : générique de début, Jean, Sara*

Pour influer sur le déroulement de l'histoire, plusieurs paramètres peuvent être réglés en début de simulation :

- la timidité de Jean et de Sara ;
- le fait que Sara soit ou non enceinte ;
- l'attirance de Sara pour Jean.

Ces différents paramètres conditionnent le déroulement de l'histoire en influant sur des probabilités d'enchaînement des diverses scènes prévues. Par exemple, si Sara est timide, elle n'acceptera pas l'invitation de Jean. Au cours de l'exécution, un utilisateur peut influer sur le scénario en faisant sonner le téléphone de Jean. En fonction de la conversation qui se déroule alors, le scénario change le comportement de Sara. Par exemple, la maîtresse de Jean peut téléphoner ; Sara se met alors à douter de l'intégrité de Jean et ne se laissera pas reconquérir. Par manque de temps lors de la réalisation de la démonstration, seule cette modalité d'interaction a été proposée.

La réalisation de cette démonstration a posé divers problèmes théoriques, qui ont été traités par quatre personnes (sans citer les personnes ayant travaillé sur la plate-forme logicielle OpenMASK qui sert de base à la réalisation des démonstrations) : animation des humanoïdes, acquisition et restitution de mouvements capturés (Stéphane Ménardais), scénario (Stéphane Donikian), perception visuelle, gestion des caméras et idiomes cinématographiques (Nicolas Courty). La majeure partie de mon travail a consisté à modéliser les comportements des acteurs semi-autonomes, à gérer leur navigation à l'intérieur de milieux contraints tels que l'épicerie et l'appartement, à connecter un serveur sonore et à spécifier les modes de communication entre les divers modules.

La figure 5.9 présente les principaux modules utilisés dans la conception de la fiction interactive. L'architecture utilisée pour gérer le comportement des humanoïdes ainsi que leur déplacement est celle présentée section 5.1 (modulo le modèle cognitif) que nous n'avons pas de nouveau détaillé sur ce schéma. Pour la réalisation de cette démonstration, le modèle HPTS++ a été utilisé dans trois domaines d'application :

- gestion du comportement des humanoïdes,
- gestion du système de cinématographie [Cou02],
- gestion du scénario.

Le scénario constitue le cœur de l'application, il cadence les actions des humanoïdes en leur soumettant un certain nombre de commandes relatives au comportement à adopter pour respecter la ligne directrice de l'histoire. Ce scénario a été décrit par l'intermédiaire d'HPTS++ en utilisant une hiérarchie d'automates. La ligne principale a été décrite par l'intermédiaire d'un automate décrivant l'enchaînement des scènes. Lors du commencement de chaque scène, le scénario effectue une requête auprès du serveur sonore pour jouer une musique de circonstance. Chaque sous automate correspond à une scène particulière (synopsis dans l'appartement, rencontre dans l'épicerie et discussion dans l'appartement) qui elle même utilise des sous automates pour gérer des sous parties génériques de scénario. Par exemple, un automate générique a été conçu pour la gestion des discussions. Il est paramétré par les phrases à prononcer et par les acteurs les prononçant. La synchronisation de la conversation est gérée par messages : requête de diction et signalement de fin de phrase. Outre les diverses requêtes effectuées auprès des acteurs, le scénario est aussi en charge de signaler au système de cinématographie la typologie de la scène et la manière de la filmer. Des détails sur cette version du système de cinématographie peuvent être consultés dans [Cou02].

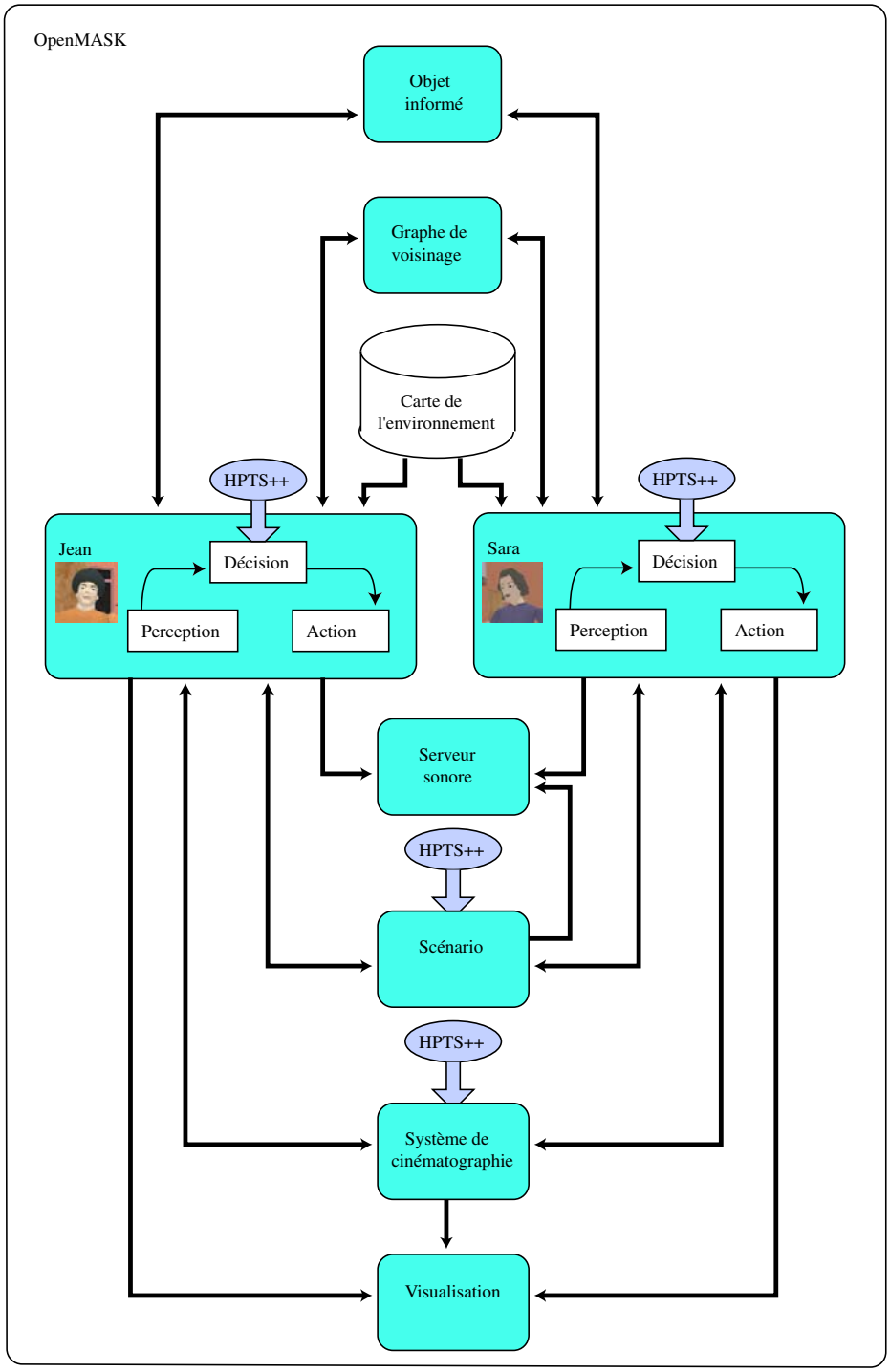


FIG. 5.9 – *Architecture de l'application de fiction interactive.*

En terme de modélisation des comportements, plusieurs facteurs dépendant du scénario ont été pris en compte. Par exemple, le paramètre psychologique lié à la timidité possède une grande influence sur le comportement ; il conditionne les attitudes des personnages lors des conversations. Le personnage timide a tendance à peu bouger, croiser ses mains derrière le dos, alors que le personnage qui n'est pas timide va parler avec les mains, en faisant de grands gestes. Ce paramètre influe aussi sur l'attention visuelle. Dans ce cadre, il est utilisé pour définir les priorités associées aux automates de perception en gérant la concurrence sur la ressource des yeux entre trois comportements :

- le comportement de perception passive (comportement pas défaut de l'humanoïde),
- le comportement consistant à regarder l'interlocuteur (comportement associé à une discussion),
- le comportement consistant à regarder vers le sol (comportement associé à la timidité durant une conversation).

L'une des conséquences de cette concurrence est qu'un personnage timide aura tendance à jeter de brefs coups d'œils vers son interlocuteur alors qu'un personnage qui n'est pas timide le regardera droit dans les yeux. Pour être à même de donner une sensation de vie lors des conversations, les mouvements de la bouche ont été synchronisés sur le son correspondant à la phrase prononcée. Cette synchronisation est effectuée par l'intermédiaire d'une information fournie par le serveur sonore : la moyenne du volume sonore sur environ un cinquantième de seconde. Cette moyenne est alors appliquée (modulo un facteur d'échelle) comme angle d'ouverture de la bouche. Même si cela ne parait pas réaliste à prime abord, le résultat obtenu est proche de celui d'un dessin animé (de bonne qualité) auquel l'œil humain est relativement habitué et s'avère de meilleure qualité qu'un film doublé. Ce détail, couplé à l'attention visuelle, permet de rendre les personnages plus vivants en fournissant une information visuelle supplémentaire au spectateur. En animation comportementale, ces détails sont d'une importance capitale ; ils fournissent au canal de perception humain des informations (qui à prime abord ne sont pas interprétées consciemment) dont la présence dénote la vie.

Lors de l'intervention de l'utilisateur, le scénario envoie un message à Jean provoquant le lancement d'un comportement de réponse au téléphone. Le mécanisme d'ordonnancement d'HPTS++ permet d'automatiquement synchroniser la réponse avec le contexte : Jean ne répond que lorsqu'il n'est plus en train de parler et que Sara a fini sa phrase. Les comportements d'écoute et de diction utilisent deux ressources qui sont nécessaires au comportement de répondre au téléphone. Lorsque le comportement est à même de prendre ces deux ressources, Jean se lève, prononce une phrase de transition et va répondre dans l'entrée de l'appartement. Le comportement de se lever est provoqué par la concurrence entre le comportement de rester assis et le comportement de déplacement associé à la réponse au téléphone (plus prioritaire). Lorsque le comportement consistant à rester assis relâche la ressource des jambes, il provoque un comportement consistant à se lever ; le comportement de réponse au téléphone peut alors générer le déplacement. Lorsqu'il se termine il relâche la ressource des jambes, provoquant la reprise du comportement consistant à rester assis. Jean retourne alors vers le canapé et se rassied en reprenant la conversation là ou elle en était restée. Cette propriété, qui peut être comparée à celle des piles de tâches [Mon02] ou des piles d'automates [NT97], est l'une des conséquences de l'utilisation du mécanisme d'ordonnancement d'HPTS++.

Les figures 5.10 et 5.11 montrent des prises de vue de la démonstration. La série d'images de la figure 5.10 présente la scène de rencontre de Jean et Sara dans l'épicerie. Jean ayant saisi le paquet de gâteaux (qu'il tient dans la main droite) aperçoit Sara. Il décide de lui parler et l'invite chez lui. La figure 5.11 présente la scène suivante dans laquelle les deux personnages conversent dans le salon, assis sur les canapés. Le téléphone sonne alors et Jean se lève pour aller répondre. Cette

FIG. 5.10 – *Scène de rencontre à l'épicerie.*

FIG. 5.11 – *Scène de discussion dans l'appartement.*

démonstration fonctionne en temps réel (25 images secondes) sur un ordinateur équipé de deux processeurs pentium 4 cadencés à 1 GHz et d'une carte graphique de type Quadro 2. En terme de performances, le goulet d'étranglement se situe dans l'affichage des décors et des humanoïdes. Ils sont constitués d'un grand nombre de facettes et de degrés de liberté qui demandent un calcul intensif de la part du processus d'affichage.

La réalisation de cette démonstration a soulevé un grand nombre de questions quand à l'interaction entre les humanoïdes et le scénario. Le système utilisé pour décrire le scénario s'avère très dirigiste et laisse relativement peu de place à l'autonomie. Dans ce cadre, il semblerait plus judicieux d'utiliser un contrôle de plus haut niveau. Le scénario pourrait être décrit sous la forme d'automates décrivant des suites de buts à réaliser. Ces buts seraient alors fournis à des humanoïdes autonomes qui choisiraient leurs actions en fonction de leurs connaissances sur l'état du monde. Cela permet-

trait de générer des situations imprévues, tout en respectant la ligne directrice du scénario. De plus, il deviendrait possible de décrire des scénarios à un haut niveau d'abstraction en ne souciant pas des modalités de contrôle des humanoïdes mais en se concentrant sur la trame de l'histoire [Don03]. Cette technique va être testée dès que la connexion entre BCOOL et HPTS++ sera effective.

5.2.3 Cinématographie virtuelle

Durant la réalisation de la fiction interactive, un système de cinématographie dédié a été utilisé. Il nous a permis de mettre en évidence un certain nombre de problèmes liés à l'adaptation au contexte d'une simulation dynamique dans laquelle tout ne peut être prévu à l'avance. Dans ce cadre, nous avons effectué un travail de spécification d'une architecture pour un système de cinématographie autonome, s'appuyant sur les propriétés d'HPTS++ [CLDM03]. Pour des notions d'adaptation, ce système est orienté agent. Nous allons commencer par présenter certains aspects de la cinématographie, puis nous montrerons comment les propriétés d'HPTS++ peuvent permettre, d'une part, d'adapter les prises de vue des caméras à un contexte dynamique et, d'autre part, de retranscrire des règles cinématographiques.

5.2.3.1 Cinématographie

Le langage cinématographique est une référence internationale commune. Il peut être considéré comme une structure permettant de transmettre une histoire dans un style familier aux spectateurs. Un film peut être vu comme une suite d'images mais est généralement structuré en prises de vue, scènes et périodes. Les périodes sont décomposées en plusieurs scènes. Une scène possède une continuité spatiale et temporelle ; elle est généralement décomposée en plusieurs prises de vue. Une prise de vue représente une suite d'images possédant une continuité temporelle, filmées par une même caméra.

Il existe un certain nombre de règles permettant d'enchaîner plusieurs prises de vues de manière cohérente pour le spectateur [Ari84]. A titre d'exemple, voici quatre règles à respecter pour ne pas perdre le spectateur :

- Si un acteur est présent dans une moitié de l'écran dans une prise de vue, il doit rester dans cette moitié durant un changement de vue sur le même axe visuel.
- Si un acteur se déplace, deux prises de vues successives doivent conserver la direction perceptible du déplacement.
- Si le même objet est vu sous différents angles, l'angle entre les points de vue devrait être supérieur à 30 degrés.
- Dans le cadre des dialogues, l'angle entre deux prises de vues doit être inférieur à 180 degrés car il n'est pas permis de franchir la ligne d'intérêt. Cette ligne imaginaire est définie comme passant par les deux acteurs en interaction.

Même si les films possèdent des scénarios différents, certaines situations sont, la plupart du temps, filmées de la même manière. Cette sorte de stéréotype est appelé un idiome. Il s'agit de règles cinématographiques codifiant la manière d'enchaîner les prises de vue dans certaines situations. Prenons l'exemple d'une conversation entre deux acteurs. Il existe trois grandes formes de plans qui peuvent être adoptées : les plans internes, les plans externes et les plans parallèles (voir figure 5.12). Il est possible de mélanger ces différentes manières de filmer en utilisant la vue globale des deux

acteurs comme transition. L'idiome décrit alors une alternance de prises de vues filmant l'acteur possédant la parole.

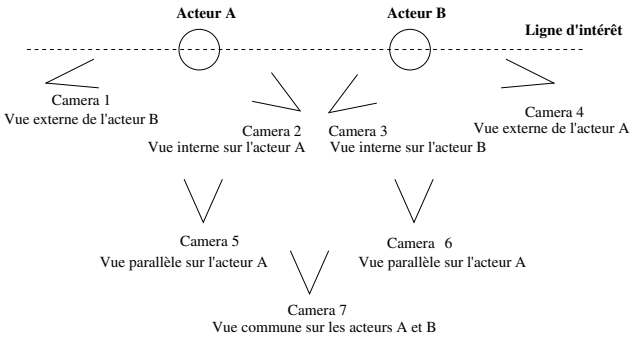


FIG. 5.12 – *Différents placements de caméras pour filmer une conversation entre deux acteurs.*

Dans le monde du cinéma, le scénario est écrit et le placement des acteurs et des caméras est prévu à l'avance en fonction de la scène et de l'action qui se déroule. Dans un monde virtuel, peuplé d'entités autonomes ou semi-autonomes, toutes les situations ne peuvent être prévues à l'avance. Dans ce domaine, il est nécessaire de posséder un système de cinématographie capable de s'adapter au contexte tout en essayant de respecter les règles cinématographiques. Cependant, ces règles peuvent être enfreintes de par la nature dynamique de l'environnement : si la situation ne permet pas d'effectuer un changement de plan tel que défini par ces règles, ces dernières doivent alors être transgressées. Dans le cadre de la cinématographie, différentes approches ont été proposées dans la littérature. Un certain nombre de pointeurs peuvent être trouvés dans la thèse de N. Courty [Cou02].

5.2.3.2 Une approche orientée agent

La notion d'idiome constitue une règle d'enchaînement de points de vue sur une scène. Ces points de vue ont pour rôle de focaliser sur un point d'intérêt propre à la scène et à son déroulement. Comme nous avons pu le voir dans le cas de la conversation entre deux personnes, il existe différentes manières de filmer un point d'intérêt qui sont dépendantes du placement des caméras. Cette redondance est à la base du système de cinématographie autonome proposé. Il exploite les propriétés d'HPTS++ en manipulant trois types d'agents décrits sous la forme d'automates :

- les agents caméra,
- les agents patron d'idiome,
- les agents idiome.

Le système de ressources associé à HPTS++ est alors utilisé pour

- gérer automatiquement la concurrence de plusieurs caméras sur les points d'intérêt qu'elles filment,
- gérer la concurrence de plusieurs idiomes sur la fenêtre graphique associée au rendu.

Agents caméra. Chaque caméra est un agent à même de filmer un ou plusieurs points d'intérêt. Cet agent est représenté sous la forme d'un automate HPTS++ supervisant une caméra utilisant le système d'asservissement visuel et de contraintes exprimées dans le plan image développé par N. Courty [Cou02]. Cette notion permet à la caméra de s'adapter à un contexte dynamique, en modifiant le point de vue (évitement d'occultation par exemple) ou en relâchant des contraintes trop fortes qui ne peuvent être satisfaites. Pour gérer la concurrence des agents caméra sur un ou plusieurs points d'intérêts, ces derniers sont symbolisés par des ressources. Plusieurs caméras utilisant des ressources en commun sont alors des caméras permettant de filmer, de manière différente, les mêmes points d'intérêt. La priorité associée à l'automate est définie comme le produit d'une qualité de prise de vue, calculée par l'agent caméra, et d'une préférence, modifiable dynamiquement de l'extérieur. Cette préférence permet de favoriser ou défavoriser le type de prise de vue réalisé par la caméra.

La figure 5.13 montre un schéma type d'automate de gestion de caméra. Cet automate possède deux états, un état consistant à filmer en s'adaptant à l'environnement mais en ne prenant pas la ressource de point d'intérêt (ptI), l'autre prenant cette ressource. La relâche de la ressource est possible dans deux cas : soit l'agent caméra est dans l'impossibilité de filmer correctement, soit la durée pendant laquelle il a filmé la scène excède un temps minimum. Dans ce dernier cas, une possibilité d'adaptation est offerte pour changer de caméra si une caméra plus prioritaire nécessite cette ressource. La caméra possédant la ressource de point de vue est alors la caméra maximisant la préférence et la qualité de la prise de vue.

Un agent caméra, lors de sa création est paramétré avec la ou les ressources correspondant aux points d'intérêt qu'il permet de filmer. Au même titre que pour les comportements, cette approche permet de gérer une cohérence intrinsèque à l'agent qui peut choisir le moment approprié pour accepter d'éventuellement relâcher la ressource. De plus, cette approche autorise l'utilisation de la caméra dans divers contextes, en concurrence avec d'autres caméras, sans devoir prendre en compte explicitement les éventuels conflits.

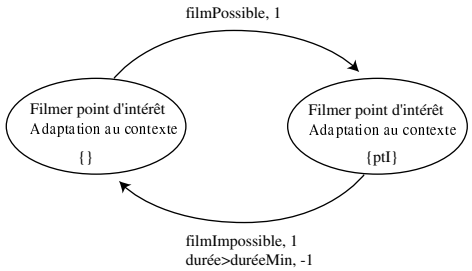


FIG. 5.13 – *Exemple simplifié de l'automate d'un agent caméra.*

Patron d'idiome. Un idiome encode une règle cinématographique permettant d'enchaîner de manière cohérente plusieurs points de vue sur une scène. Cet enchaînement de points de vue vise à corréler la réalisation au scénario, en montrant, successivement, les points d'intérêt de la scène qui peuvent être filmés par diverses caméras. Cette notion permet de créer des patrons d'idiomes qui constituent une abstraction des idiomes. Leur rôle est d'encoder l'enchaînement de prises de vues sur des points d'intérêt, sans se soucier des caméras pouvant les filmer. Ces patrons d'idiomes sont décrits sous la forme d'automates dont chaque état correspond à la focalisation sur un point

d'intérêt. Cet automate est paramétré par :

- un contrôleur local gérant la concurrence de plusieurs caméras sur des points d'intérêt,
- l'ensemble des ressources correspondant aux points d'intérêt.
- la ressource correspondant à la fenêtre de rendu, pour permettre la gestion d'une concurrence entre plusieurs idiomes.

Dans chacun des états de son automate, le patron d'idiome peut accéder à l'automate possédant la ressource correspondant au point d'intérêt à filmer. Cette accès est rendu possible par l'intermédiaire du contrôleur local passé en paramètre (voir section 3.3.2 p. 173). Cette propriété permet de décorréler le patron d'idiome des caméras qu'il manipule. Il peut alors être utilisé dans divers contextes en conjonction avec des caméras effectuant des prises de vue différentes. Chaque état filmant un point d'intérêt de la scène utilise une ressource particulière : la ressource de fenêtre graphique. Elle permet de gérer la concurrence de plusieurs idiomes se déroulant en parallèle. La fonction de priorité associée à un idiome est définie comme le produit de l'importance de la scène à filmer et de la préférence associée à l'idiome. Ces deux facteurs peuvent être modifiés dynamiquement pour s'adapter au contexte et permettent de corréler la réalisation au scénario.

La figure 5.14 présente un patron d'idiome de conversation à deux personnes. Les enchaînements entre les points de vue sur la scène dépendent de la conversation et de la personne qui parle. Lors d'une fin de phrase, une transition d'adaptation est possible pour relâcher la fenêtre graphique et laisser la main à un idiome qui s'avère plus important du point de vue du scénario.

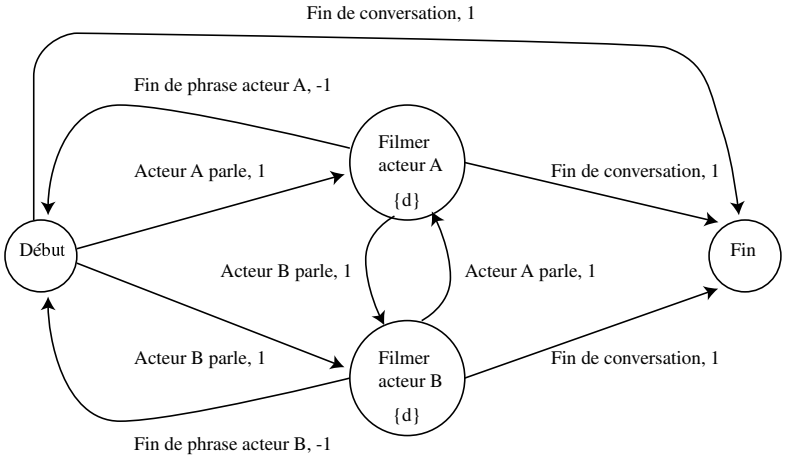


FIG. 5.14 – *Automate associé au patron d'idiome de conversation.*

Idiome. Les patrons d'idiomes, pour être à même de filmer une scène nécessitent un ensemble de caméras effectuant des prises de vues des différents points d'intérêt. L'idiome possède donc le rôle de créer dynamiquement :

- des agents caméra en concurrence sur les points d'intérêt,
- un contrôleur local qui se charge de leur exécution,
- un patron d'idiome en lui fournissant la ressource de fenêtre graphique, le contrôleur de caméras et l'ensemble des ressources correspondant aux points d'intérêts.

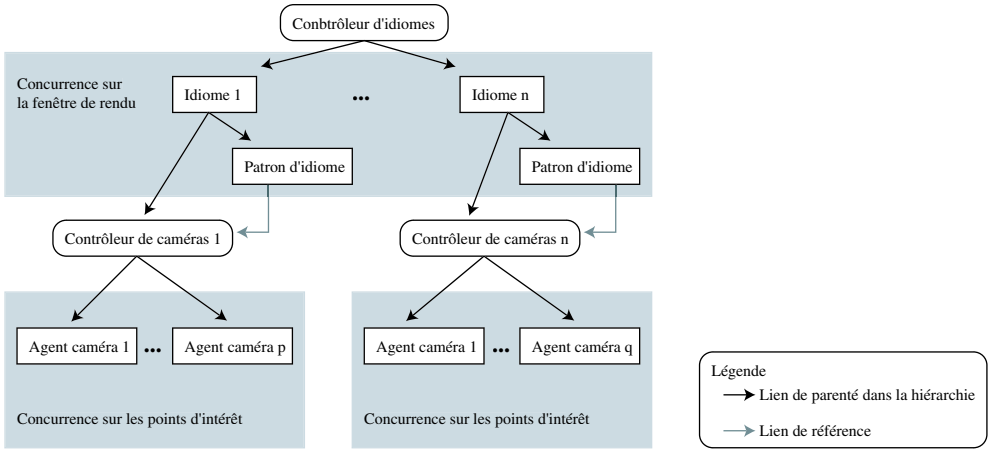


FIG. 5.15 – *Architecture du système de cinématographie.*

Architecture. La figure 5.15 présente un résumé de l'architecture du système de cinématographie autonome. Ce dernier se découpe en plusieurs niveaux de concurrence : concurrence entre les agents caméras et concurrence entre les idiomes. Cette architecture possède plusieurs bonnes propriétés.

1. Elle hérite des bonnes propriétés d'HPTS++ sur la cohérence, l'adaptation et l'exclusion mutuelle. Ces propriétés sont utilisées au niveau des idiomes mais aussi au niveau des caméras pour gérer une cohérence globale.
2. Elle permet de décrire des agents caméras qui peuvent être utilisés dans des contextes dynamiques divers. Une fois ces agents décrits, ils peuvent être utilisés en conjonction de tout patron d'idiome défini par l'utilisateur.
3. Elle gère la concurrence entre plusieurs caméras en tentant de favoriser les meilleures prises de vue. Cette notion permet de tenter de réaliser au mieux, dans un contexte dynamique et imprédictible.
4. Elle gère la concurrence entre plusieurs idiomes, permettant d'interrompre une scène à des moments cohérents pour laisser la main à une scène plus importante. Cette notion s'avère utile dans le cadre d'une fiction temps réel où le contrôle temporel est excessivement différent de celui du cinéma, car tout n'est pas prévu à l'avance et plusieurs scènes peuvent se dérouler simultanément.
5. Elle décorrèle, grâce à la notion de patron d'idiome, les idiomes et les caméras. Ces patrons d'idiomes expriment donc une règle cinématographique abstraite, qui se concrétise en fonction de l'ensemble de caméras qui lui est associé. Les changements de caméras peuvent donc permettre de décrire différents styles de réalisation, sans nécessiter une nouvelle description d'idiome, mais juste en utilisant un nouvel ensemble de caméras.

La figure 5.16 présente la première réalisation utilisant cette architecture. Chaque image se découpe en quatre fenêtres associées à quatre instances différentes de l'idiome de conversation :

- La fenêtre haut gauche présente l'idiome de conversation utilisant des plans parallèles sur les acteurs.
- La fenêtre haut droite présente ce même idiome, mais utilisant des plans externes.

Idiome avec plan parallèle	Idiome avec plan externe
Idiome mélangeant les plans	Idiome avec plan interne

FIG. 5.16 – *Un exemple montrant trois idiomes utilisant le même patron d'idiome de conversation avec des caméras différentes. Le figure du bas présente les configurations.*

- La fenêtre bas droite présente cet idiome en utilisant des plans internes.
- Enfin, la fenêtre en bas à gauche présente cet idiome utilisant une concurrence sur les trois plans précédents. Les caméras possèdent des préférences évoluant dynamiquement et les transitions entre plans s'effectuent en passant par un plan global sur les deux acteurs.

Cette démonstration présente quatre styles différents de réalisation qui peuvent être obtenus en changeant les ensembles de caméras associés au patron d'idiome de conversation.

Ce système de cinématographie constitue une première version et reste expérimental. A plus long terme, le but est de fournir un système de cinématographie autonome paramétrable avec un style de réalisation. Ce style pourrait permettre de choisir automatiquement les typologies de caméras à utiliser et paramétrer la rapidité des enchaînements de plans. Il s'agira alors de faire une étude sur les grand styles cinématographiques (Hitchcock, Lynch...) et de détecter quels sont les paramètres à prendre en compte dans le système. Cependant, il reste cette question ouverte : ces styles sont-ils exprimables sous la forme de règles formelles, nécessaires au système? D'autre part, il faut effectuer une étude approfondie sur le lien entre le scénario (qui peut éventuellement se créer ou se modifier dynamiquement) et le système de cinématographie. Il s'agit dans ce cadre de savoir quelle est l'information minimale à fournir au système dans le but de maîtriser son niveau d'asservissement.

Conclusion

L'architecture que nous avons présenté se veut ouverte. Elle met l'accent sur la modularité de description et la réutilisation des composants logiciels. Au travers d'HPTS++, elle permet de décorréler les différents niveaux de contrôle en offrant une ouverture, qui permet l'exécution cohérente et le mélange de comportements provenant de sources variées : modèle cognitif, comportement par défaut, scénario... Son intégration dans la plate-forme OpenMASK permet d'exploiter les propriétés sur les fréquences de calcul dans le but d'alléger le coût global de calcul de la simulation en conservant des temps de réaction réalistes. Les diverses démonstrations réalisées montrent sa capacité à être utilisée dans le cadre d'applications complexes, en temps réel. L'outil HPTS++, qui constitue le noyau exécutif du modèle, laisse entrevoir de nombreux domaines d'application comme le montre son utilisation dans le cadre de la description de scénarios ou dans la spécification d'un système de cinématographie autonome. Son couplage avec BCOOL permettra d'explorer de nombreux champs d'application (agents cognitifs, scénarisation par but...) dans lesquels la sélection d'actions offrira un niveau de contrôle abstrait laissant une grande place à l'autonomie et à la réactivité des systèmes décrits.

Conclusion

Nous avons abordé, tout au long de cette étude, les problèmes relatifs à la description du comportement des humanoïdes autonomes et leur interaction avec l'environnement. Dans ce cadre, nous avons proposé une architecture ouverte, tentant d'automatiser un certain nombre de traitement et d'offrir des outils et des concepts de haut niveau, visant à simplifier la description de leurs comportements.

Synthèse des travaux effectués

L'architecture que nous avons proposé, se scinde en quatre grands systèmes, visant à permettre la description de quatre formes de connaissances et de comportements relatifs à l'être humain. Deux modèles, accompagnés de langages, permettent de décrire les parties réactive et cognitive du comportement. Pour effectuer le lien avec l'environnement, nous avons proposé un modèle de navigation réactive configurable. Ce modèle s'appuie sur une subdivision de l'espace calculée à partir du modèle géométrique de l'environnement, permettant aux entités, après un simple traitement, de naviguer de manière cohérente.

Représentation de l'espace

Nous sommes partis du constat que les environnements virtuels sont le plus souvent fournis sous la forme de fichiers de géométrie, pour proposer un modèle de subdivision spatiale offrant une représentation adéquate pour la navigation des humanoïdes. Ce modèle permet de représenter précisément des environnements intérieurs et/ou extérieurs complexes et de grande taille. La chaîne de traitement proposée automatise l'extraction d'une subdivision spatiale en cellules convexes et détecte les goulets d'étranglements. Cette propriété assure que les chemins planifiés peuvent être empruntés par les humanoïdes, sans traitement supplémentaire relatifs à leur envergure. De plus, les cellules générées disposent de bonnes propriétés topologiques. Cette particularité peut ensuite être exploitée pour créer des niveaux d'abstraction topologiques, qui permettent de réduire l'ordre de grandeur de la complexité de planification d'un chemin. Il est alors possible de planifier des chemins, en temps réel, dans des environnements complexes de grande taille. Enfin, la structure de la subdivision autorise des calculs rapides de visibilité par lancer de rayon, qui sont à la base du filtrage du graphe de voisinage utilisé dans le processus de navigation.

Navigation

Le modèle de navigation proposé s'inspire des études sur le comportement piétonnier et prend en compte, par défaut, ses grandes caractéristiques. Sa partie configurable, par l'intermédiaire des

modules de réactions aux collisions, permet de reproduire une grande variété de comportements inspirés des diverses analyses du comportement humain. Il permet de tester l'impact de ces règles lors de la simulation de foules composées de plusieurs centaines d'entités. Dans ce domaine, l'utilisation du graphe de voisinage lui confère de bonnes propriétés. Il permet de maîtriser la complexité du calcul de voisinage, en effectuant un calcul global, sans nuire à l'autonomie des entités. Le résultat de ce calcul permet alors d'accéder directement aux entités voisines lors des tests de collisions, sans traitement supplémentaire. Enfin, la distance de prise en compte du voisinage est dépendante de la densité des piétons mais pas d'une zone de rayon fixe autour de l'entité. Les algorithmes adaptent alors automatiquement leur distance de prévision en fonction de ce facteur. La propriété du voisinage direct, et du nombre linéaire des relations de voisinage, en fonction du nombre d'entités, permet d'assurer la stabilité du coût de calcul global de la simulation. Ce modèle a été testé avec succès, en environnement intérieur et extérieur. La simulation effectuée dans le centre ville de Rennes montre ses possibilités en permettant de simuler le comportement de navigation autonome de 2400 piétons en temps réel.

HPTS++

HPTS++ permet la description des comportements réactifs qui peuvent être adoptés par l'humanoïde et offre des mécanismes automatisant la gestion de leur déroulement en parallèle. La gestion par ressources permet la description d'un comportement de manière atomique, en stipulant ses possibilités d'adaptation au travers de leurs effets mais sans en décrire les causes. Le système d'ordonnancement permet alors de gérer une exécution cohérente, sans conflits de ressources ni problèmes d'inter-blocages. Il mélange automatiquement les comportements, au mieux, en fonction de leur importance respective et de leurs possibilités d'adaptations. Cette particularité constitue l'originalité de ce système qui s'avère être le premier modèle réactif à aborder cette problématique. Grâce à ce système, les comportements peuvent être manipulés sans aucune connaissance sur leur déroulement, tout en étant assuré d'une réalisation cohérente. Cette propriété offre un niveau d'abstraction à la fois nécessaire et suffisant pour autoriser l'utilisation d'un mécanisme de raisonnement travaillant sur les conséquences des comportements, sans se soucier de leur réalisation.

Le langage proposé s'insère dans cette optique, en intégrant des concepts de modélisation objet pour la description des comportements. Cette approche permet d'alléger le travail de description, tout en permettant de créer des catégories de comportements, manipulables à différents niveaux d'abstraction. L'exemple du lecteur, buveur, fumeur, met en évidence l'intérêt de cette approche, en reproduisant automatiquement un comportement émergeant cohérent, impossible à reproduire avec les autres systèmes sans une description exhaustive.

BCOOL

Le langage BCOOL est dédié à la description de la composante cognitive des humanoïdes. Il s'inspire de la théorie écologique de Gibson [Gib86], en permettant de décrire le monde sous la forme de ce qu'il offre à un humanoïde. Il s'agit d'un langage de haut niveau, utilisant des concepts issus de la programmation objet. En cela, il met l'accent sur la modélisation et permet la description de composants cognitifs réutilisables. Les notions de modélisation objet permettent d'introduire, automatiquement, des hiérarchies de concepts dans le monde cognitif de l'agent, directement corrélés à la modélisation du concepteur. Les actions (ou interactions), offertes par les objets, prennent en compte les faits décrivant le monde mais aussi la connaissance de leur valeur de vérité. Il devient alors

possible d'exprimer, suivant le même formalisme, des actions perceptives et des actions effectives. Une fois les classes d'objets spécifiées, la description du monde cognitif de l'agent s'effectue par l'intermédiaire d'une description des instances. Cette description est ensuite utilisée pour restreindre l'agent aux interactions offertes par l'environnement.

Le mécanisme de sélection d'actions peut alors prendre le relais, en lui fournissant la possibilité d'atteindre un but fixé. Le système admet différentes formes de buts, certains sont générateurs de comportement, d'autres inhibiteurs. Ils permettent de décrire diverses formes d'influences en indiquant ce que l'agent doit faire mais aussi ne pas faire. Le mécanisme tient compte de la dynamique du monde, en acceptant toute forme de mise à jour des connaissances provenant d'un système extérieur (perception par exemple). Il permet aussi d'introduire des mécanismes d'évolution de la confiance dans la connaissance. Les suites d'actions générées deviennent alors perceptives, quand une connaissance est requise, et effectives lors de la modification de l'état du monde (ou les deux). Enfin, par l'intermédiaire du langage, le lien est effectué avec le modèle réactif, pour réaliser les comportements sélectionnés. Ce dernier, grâce à ses bonnes propriétés, décharge le modèle cognitif de la gestion de la cohérence des actions.

Validation

Plusieurs expériences ont validé les travaux que nous avons présenté : tests de rapidité de la planification, simulation de foules dans le centre ville de Rennes, démonstration du lecteur, buveur fumeur. Ils ont, d'autre part, été utilisés dans deux démonstrations grand public : un musée virtuel peuplé d'humanoïdes autonomes, en exposition permanente à la cité des sciences et de l'industrie de la Villette et une démonstration de fiction interactive, présentée au festival Imagina 2002. Le logiciel HPTS++ est désormais déposé à l'agence de protection des programmes (APP) et est utilisé dans le cadre d'un projet RIAM[2] : AVA motion. Ce projet regroupe des partenaires de la recherche tels que l'IRISA[3] et le LPBEM[4] de l'université de Rennes II, mais aussi les sociétés Kineo Cam et Daesign. Ce projet a pour but de fournir un environnement évolué pour la conception d'humanoïdes virtuels, regroupant la gestion de leurs animations, la représentation de l'environnement, la planification de chemin et la navigation, ainsi que la définition de leurs comportements.

Les modèles proposés doivent cependant, encore être soumis à des tests. Le mécanisme de sélection d'actions de BCOOL doit être testé dans le cadre d'environnements dynamiques et connecté à l'animation de l'humanoïde, pour vérifier la cohérence des comportements générés. Le modèle de foules, pour sa part, doit être validé par l'intermédiaire de tests permettant de comparer les simulations à des situations réelles ; une thèse vient de débuter sur ce sujet, en collaboration avec l'AREP[5].

Perspectives

Au cours de nos recherches, plusieurs perspectives de travail ont émergé des divers travaux que nous avons mené.

2. Réseau Recherche et Innovation en Audiovisuel et Multimedia.
3. Institut de Recherche en Informatique et Systèmes Aléatoires.
4. Laboratoire de Physiologie et Biomécanique de l'Exercice Musculaire.
5. Aménagement Recherche Pôle d'Échange

Modèle de croyances et mémoire

Le mécanisme de sélection d'action de BCOOL prend en compte l'évolution de la confiance dans les croyances, cependant il n'offre pas de mécanisme permettant de la gérer. Il est intéressant, dans ce cadre de travailler sur un modèle de croyances permettant de prendre en compte les probabilités de changement du monde en fonction du temps. Ce modèle, indirectement, est aussi lié à la notion de mémoire. Un humanoïde virtuel, pour être réaliste doit pouvoir oublier ou remettre en cause sa connaissance au bout d'un certain temps. Ces notions sont d'une part intéressantes pour le réalisme mais peuvent aussi être génératrices de comportements tels que la discussion (échange de connaissances) ou la surveillance (un comportement qui se traduit par un but de connaissance de l'état d'un lieu) par exemple.

Personnalisation des humanoïdes

Le modèle de sélection d'actions présenté, permet d'influer sur les actions de l'agent, par l'intermédiaire des différentes formes de buts. Comme nous le disions, les buts de maintien peuvent traduire une personnalité maniaque, voulant constamment maintenir un état de l'environnement. Dans un grand nombre de cas, l'influence de la personnalité est traduite par l'intermédiaire d'un profil psychologique utilisé comme pré-condition d'une action [Mon02]. Une approche différente pourrait consister à traduire la personnalité sous une forme factuelle, autrement dit en terme d'activation ou d'inhibition d'un certain nombre de faits décrivant l'environnement. Cette méthode, suivant l'approche adoptée dans BCOOL, permettrait d'activer ou d'inhiber les actions possédant ces faits pour conséquence.

L'atout se situe alors dans le gain en terme de description. Cela permettrait d'automatiser la gestion de la dépendance entre les actions et la personnalité au travers des faits, moins nombreux que les actions possibles des humanoïdes. Par ce biais, toute nouvelle action décrite, subirait automatiquement l'influence de la personnalité. Cependant, cette approche nécessite une étude approfondie sur la notion de personnalité et sur son influence directe sur le comportement pour répertorier les traits de personnalité qui peuvent être traduits sous cette forme.

Scénarisation par buts

La description du comportement cognitif des entités, par l'intermédiaire de BCOOL, laisse entrevoir des applications en terme de scénarisation. Le mécanisme de sélection d'actions permet à des agents d'adopter une suite de comportements leur permettant d'atteindre un but fixé.

En terme de scénarisation, il est possible de créer un agent scénario capable de raisonner sur la situation, rejoignant en cela l'approche de Cavazza sur la fiction interactive [CCM02]. Ses interactions avec l'environnement seraient alors des buts fournis à des agents cognitifs sous ses ordres. Le modèle HPTS++ pourrait être utilisé pour décrire la trame de l'histoire sous la forme d'un enchaînement logique de buts partiels. Le scénario pourrait alors sélectionner des actions adaptées à la situation en vue de fournir aux agents qu'il contrôle des sous-buts à réaliser.

Cette technique permettrait de générer des histoires respectant la trame d'un scénario mais dont le déroulement pourrait changer en fonction les actions d'un utilisateur plongé dans le monde virtuel. Le système deviendrait à la fois réactif et inventif face aux conflits générés. D'autre part, la spécification d'un scénario s'effectuerait à un haut niveau d'abstraction, rendant cette technique accessible à des non-informaticiens.

Système de cinématographie autonome

Sur la base du modèle HPTS++, nous avons proposé une version préliminaire d'un système de cinématographie autonome [CLDM03]. A plus long terme, les perspectives sont de modéliser les grands « styles » cinématographiques (Hitchcock, Lynch...), en autorisant leur utilisation dans un contexte dynamique. Dans le domaine de la fiction interactive, cette propriété pourrait permettre de personnaliser la réalisation en fonction du scénario (histoire fantastique, scènes d'actions...). Cela soulève cependant la question de savoir, si ces styles peuvent être exprimés sous la forme de règles formelles, nécessaires au système. Une autre question qu'il faut alors se poser, réside dans le lien entre le scénario et le système de cinématographie. Quelle est l'information minimale à fournir, y a-t-il besoin d'une communication explicite, le système cinématographique doit-il être asservi?

Validation d'environnements

Comme nous le précisions lors de l'introduction, la simulation de piétons peut être utilisée pour vérifier que des environnements respectent certaines contraintes en terme d'évacuation en situation d'urgence par exemple. Dans ce cadre, les résultats de nos travaux sur la navigation et la simulation de foules peuvent être utilisés. Le modèle que nous proposons permet de faire des simulations à un niveau microscopique avec une grande quantité de piétons, en temps réel. Par le biais des configurations des règles de navigation et des divers paramétrages, un utilisateur pourrait configurer, de manière interactive, les comportements individuels. Le graphe de voisinage pourrait alors être utilisé en temps réel, pour calculer diverses statistiques telles que les densités localisées de piétons par exemple. Il peut aussi servir pour la détection, automatique et rapide, de groupes en utilisant des méthodes de classifications exploitant ses propriétés sur le voisinage. Pour sa part, le mode de subdivision spatiale fournit une information très utile : les goulets d'étranglements. Dans le cadre de situations de panique, ils constituent les passages critiques, dans lesquels les personnes se ruent. Des statistiques pourraient alors être calculées, sur le nombre de personnes qui franchissent la zone pour en déduire une information sur le temps d'évacuation moyen en fonction de la taille des sorties.

Annexe A

Nomenclature des grammaires

Deux grammaires sont fournies dans ces annexes : la grammaire de HPTS++ et la grammaire de BCOOL. Elles utilisent toutes deux la nomenclature suivante :

- A^* est une répétition entre 0 et n fois ;
- A^+ est une répétition entre 1 et n fois ;
- { } qualifie un bloc optionnel ;
- ( ) qualifie un groupement ;
- | est un "ou". ;
- les terminaux sont en gras ;
- les non terminaux sont en style normal.

Deux *tokens* sont utilisés :

- ID est un *token* reconnaissant une lettre, suivie d'une suite de caractères alphanumériques. Il est utilisé pour lire des identifiants.
- gobeCpp est un *token* particulier qui lit un suite de caractères quelconques mais s'arrête lorsqu'il trouve « }} ». Il est utilisé pour lire du code C++ natif, sans l'analyser.

Enfin, la règle permettant de lire un code C++ est la suivante :

cpp ::= **{{** gobeCpp **}}**

Annexe B

Grammaire de HPTS++

La description de la grammaire d'HPTS++ utilise la nomenclature définie en annexe A p. 247.

stateMachineDescription	::=	fileHeader stateMachineHeader **{** { variables } { priority } { **before step** cpp } { **after step** cpp } { signalKill } { signalSuspend } { signalResume } initialState (state)* (transition)* **}**
fileHeader	::=	**header** cpp
stateMachineHeader	::=	stateMachineD { **:** (stateMachineD | classD) (**,** (stateMachineD | classD))* }
stateMachineD	::=	**state machine** ID
classD	::=	**class** ID
priority	::=	**priority** cpp
variables	::=	**variables** cpp
initialState	::=	**initial state** cpp
state	::=	**state** ID **{** { **uses** cpp (**,** cpp)* **;** } { **entry** cpp } { **during** cpp } { **exit** cpp } { signalKill } { signalSuspend } { signalResume } **}**

transition	::=	**transition** **{** **origin** cpp **extremity** cpp **preference** cpp **condition** cpp **action** cpp **}**
signalKill	::=	**kill** cpp
signalSuspend	::=	**suspend** cpp
signalResume	::=	**resume** cpp

Annexe C

Grammaire de BCOOL

La description des grammaires de BCOOL utilise la nomenclature définie en annexe A p. 247. Ce langage est composé de deux grammaires distinctes : la grammaire de description du monde conceptuel et la grammaire de description du monde réel.

C.1 Grammaire de description du monde conceptuel

BCOOL	::=	(object | relation)+
object	::=	**object** ID { inherits } definition
inherits		**inherits** ID (**,** ID)*
definition	::=	**{** (member)* (action)* **}**
member	::=	(**property** | ID) ID **;**
action	::=	**action** ID (parameterList)
		{
		constraint exprConstraint **;**
		precondition factList **;**
		effect factList **;**
		target extentedID **;**
		cost cpp
		action cpp
		}
relation	::=	**relation** ID **(** parameterList **)**
	::=	(**;** | **{ constraint** exprConstraint **}**)
extendedID	::=	(**this** | ID) (**.** ID)*
parameterList	::=	ID ID (**,** ID ID)*
factList	::=	fact (**,** fact)*
fact	::=	{ **!** } ((relationUse | extendedID) |
		K ((relationUse | extendedID) **)**)
relationUse	::=	ID **(** extendedId (**,** extendedId)* **)**

exprConstraint ::= xorCT
xorCt ::= orCt (**xor** orCt)*
orCt ::= andCt (**or** andCt)*
andCt ::= atomicCt (**and** atomicCt)*
atomicCt ::= (eqCt | fromCt | notCt | **(** exprConstraint **)**)
eqCt ::= exprConstraint (== | **!**=) exprConstraint
fromCt ::= **from (** extendedID **)**
notCt ::= **not (** exprConstraint **)**

C.2 Grammaire de description du monde réel

world ::= (object | relation)*
object ::= ID instance
instance ::= ID { true | false | **{** (Instance)+ } **}**
relation ::= ID **(** extendedId (**,** extendedId)* **)**
extendedId ::= ID (**.** ID)*

Annexe D

Exemple de monde BCOOL

Cette annexe contient la description du monde utilisé dans l'exemple de la section 4.4.2 p. 211 associé à BCOOL. Cette annexe contient :

- la description du monde conceptuel,
- la description du monde réel,
- les traces détaillées du premier scénario,
- les traces détaillées du second scénario.

D.1 Description du monde conceptuel

```
// Relation de localisation de l'humanoïde
relation location(Human h, Location l) ;

// Relation de localisation d'un objet manipulable
relation location(Takable o, Location l) ;

// Relation stipulant la prise d'un objet manipulable
relation takenBy(Takable t, Human h, Hand hand)
{
  constraint from(hand)=h ;
}

// Classe de base des objets manipulables
object Takable
{
  // L'objet est-il pris ?
  property taken = false ;

  // Prise de l'objet par l'humanoïde
  action take(Human human, Hand hand, Location l)
    {
      constraint from(hand)=human ;
```

```
      precondition K(location(this,l)), !this.taken, location(this, l),
    hand.empty, location(human, l) ;
      effect this.taken, !hand.empty, takenBy(this, human, hand), !location(this, l) ;
      target human ;
    }

  // Dépôt de l'objet par l'humanoïde
  action put(Human human, Hand hand, Location l)
    {
      constraint from(hand)=human ;
      precondition takenBy(this, human, hand), location(human, l) ;
      effect !this.taken, hand.empty, !takenBy(this, human, hand), location(this,l) ;
      target human ;
    }

  // Permet de regarder l'objet pour vérifier qu'il se trouve
  // bien dans une pièce
  action watch(Human human, Location l)
    {
      precondition location(human, l) ;
      effect K(location(this,l)) ;
      target human ;
    }
}

// Classe de récipients pouvant être remplis et manipulés
object Recipe inherits Takable
{
  property empty = true ;
}

// Classe des Lavabos, ces objets ne peuvent être manipulés
object Lavabo
{
  // Permet de remplir un récipient à l'aide du robinet du lavabo
  action fill(Human human, Hand hand, Recipe recipe, Location l)
    {
      constraint from(this)=l ;
      precondition takenBy(recipe, human, hand), recipe.empty, location(human, l) ;
      effect !recipe.empty ;
      target human ;
    }
}

// Un lieu
```

```
object Location
{}

// La cuisine, un lieu contenant un lavabo
object Kitchen inherits Location
{
  Lavabo lavabo ;
}

// Une main
object Hand
{
  property empty = true ;
}

// Un humanoïde
object Human
{
  // Main gauche de l'humanoïde
  Hand leftHand ;
  // Main droite de l'humanoïde
  Hand rightHand ;

  // Déplacement depuis un lieu, ver un autre lieu
  action gotoTo(Location l1, Location l2)
    {
      precondition connexe(l1,l2), location(this, l1) ;
      effect !location(this, l1), location(this, l2) ;
      target this ;
    }
}
```

D.2 Description du monde réel

Le monde réel de l'exemple est constitué d'une maison possédant quatre pièces : un salon, une cuisine, un couloir et une chambre. Un agent est présent dans le salon et peut manipuler un verre, se trouvant dans la chambre. L'agent, pour sa part, se trouve dans le salon.

```
// L'humanoïde
Human jean

// Le verre
Recipe verre

// Les lieux de l'environnement
```

```
Location salon
Location couloir
Location chambre
Kitchen cuisine

// Positionnement de l'humanoïde et du verre dans
// l'environnement
location(jean, salon)
location(verre, chambre)

// Relations d'accessibilité entre les lieux
connexe(cuisine, couloir)
connexe(salon, couloir)
connexe(chambre, couloir)
connexe(couloir, cuisine)
connexe(couloir, salon)
connexe(couloir, chambre)
```

D.3 Traces détaillées des deux scénarios

D.3.1 Premier scénario

Dans ce premier scénario, l'humanoïde connaît la localisation du verre et doit le remplir.

```
possible actions
(2) action(jean.gotoTo, salon, couloir)
(30) action(verre.watch, jean, salon)
---------------------------
select goal : 69 activ 0.3 rank 20
Action selected : 2
Execute action 2 action(jean.gotoTo, salon, couloir)
---------------------------
possible actions
(9) action(jean.gotoTo, couloir, salon)
(10) action(jean.gotoTo, couloir, cuisine)
(11) action(jean.gotoTo, couloir, chambre)
(33) action(verre.watch, jean, couloir)
---------------------------
select goal : 69 activ 0.3 rank 13.3333
Action selected : 11
Execute action 11 action(jean.gotoTo, couloir, chambre)
---------------------------
possible actions
(8) action(jean.gotoTo, chambre, couloir)
```

```
(24) action(verre.take, jean, jean.leftHand, chambre)
(28) action(verre.take, jean, jean.rightHand, chambre)
(32) action(verre.watch, jean, chambre)
---------------------------
select goal : 69 activ 0.3 rank 13.3333
Action selected : 24
Execute action 24 action(verre.take, jean, jean.leftHand, chambre)
---------------------------
possible actions
(8) action(jean.gotoTo, chambre, couloir)
(16) action(verre.put, jean, jean.leftHand, chambre)
(32) action(verre.watch, jean, chambre)
---------------------------
select goal : 69 activ 0.3 rank 20
Action selected : 8
Execute action 8 action(jean.gotoTo, chambre, couloir)
---------------------------
possible actions
(9) action(jean.gotoTo, couloir, salon)
(10) action(jean.gotoTo, couloir, cuisine)
(11) action(jean.gotoTo, couloir, chambre)
(17) action(verre.put, jean, jean.leftHand, couloir)
(33) action(verre.watch, jean, couloir)
---------------------------
select goal : 69 activ 0.3 rank 6.66667
Action selected : 10
Execute action 10 action(jean.gotoTo, couloir, cuisine)
---------------------------
possible actions
(5) action(jean.gotoTo, cuisine, couloir)
(12) action(cuisine.lavabo.fill, jean, jean.leftHand, verre, cuisine)
(15) action(verre.put, jean, jean.leftHand, cuisine)
(31) action(verre.watch, jean, cuisine)
---------------------------
select goal : 69 activ 0.3 rank 3.33333
Action selected : 12
Execute action 12 action(cuisine.lavabo.fill, jean, jean.leftHand, verre, cuisine)
```

D.3.2 Second scénario

Dans ce second scénario, l'humanoïde ne connaît pas la localisation du verre et doit le remplir.

```
---------------------------
possible actions
```

```
(2) action(jean.gotoTo, salon, couloir)
(30) action(verre.watch, jean, salon)
---------------------------
select goal : 69 activ 0.5 rank 10
Action selected : 30
Execute action 30 action(verre.watch, jean, salon)
---------------------------
possible actions
(2) action(jean.gotoTo, salon, couloir)
(30) action(verre.watch, jean, salon)
---------------------------
select goal : 69 activ 0.5 rank 14
Action selected : 2
Execute action 2 action(jean.gotoTo, salon, couloir)
---------------------------
possible actions
(9) action(jean.gotoTo, couloir, salon)
(10) action(jean.gotoTo, couloir, cuisine)
(11) action(jean.gotoTo, couloir, chambre)
(33) action(verre.watch, jean, couloir)
---------------------------
select goal : 69 activ 0.5 rank 8
Action selected : 33
Execute action 33 action(verre.watch, jean, couloir)
---------------------------
possible actions
(9) action(jean.gotoTo, couloir, salon)
(10) action(jean.gotoTo, couloir, cuisine)
(11) action(jean.gotoTo, couloir, chambre)
(33) action(verre.watch, jean, couloir)
---------------------------
select goal : 69 activ 0.5 rank 12
Action selected : 10
Execute action 10 action(jean.gotoTo, couloir, cuisine)
---------------------------
possible actions
(5) action(jean.gotoTo, cuisine, couloir)
(31) action(verre.watch, jean, cuisine)
---------------------------
select goal : 69 activ 0.5 rank 6
Action selected : 31
Execute action 31 action(verre.watch, jean, cuisine)
---------------------------
possible actions
(5) action(jean.gotoTo, cuisine, couloir)
```

```
(31) action(verre.watch, jean, cuisine)
---------------------------
select goal : 69 activ 0.5 rank 14
Action selected : 5
Execute action 5 action(jean.gotoTo, cuisine, couloir)
---------------------------
possible actions
(9) action(jean.gotoTo, couloir, salon)
(10) action(jean.gotoTo, couloir, cuisine)
(11) action(jean.gotoTo, couloir, chambre)
(33) action(verre.watch, jean, couloir)
---------------------------
select goal : 69 activ 0.5 rank 12
Action selected : 11
Execute action 11 action(jean.gotoTo, couloir, chambre)
---------------------------
possible actions
(8) action(jean.gotoTo, chambre, couloir)
(32) action(verre.watch, jean, chambre)
---------------------------
select goal : 69 activ 0.5 rank 10
Action selected : 32
Execute action 32 action(verre.watch, jean, chambre)
---------------------------
possible actions
(8) action(jean.gotoTo, chambre, couloir)
(24) action(verre.take, jean, jean.leftHand, chambre)
(28) action(verre.take, jean, jean.rightHand, chambre)
(32) action(verre.watch, jean, chambre)
---------------------------
select goal : 69 activ 0.5 rank 8
Action selected : 24
Execute action 24 action(verre.take, jean, jean.leftHand, chambre)
---------------------------
possible actions
(8) action(jean.gotoTo, chambre, couloir)
(16) action(verre.put, jean, jean.leftHand, chambre)
(32) action(verre.watch, jean, chambre)
---------------------------
select goal : 69 activ 0.5 rank 12
Action selected : 8
Execute action 8 action(jean.gotoTo, chambre, couloir)
---------------------------
possible actions
(9) action(jean.gotoTo, couloir, salon)
```

```
(10) action(jean.gotoTo, couloir, cuisine)
(11) action(jean.gotoTo, couloir, chambre)
(17) action(verre.put, jean, jean.leftHand, couloir)
(33) action(verre.watch, jean, couloir)
---------------------------
select goal : 69 activ 0.5 rank 4
Action selected : 10
Execute action 10 action(jean.gotoTo, couloir, cuisine)
---------------------------
possible actions
(5) action(jean.gotoTo, cuisine, couloir)
(12) action(cuisine.lavabo.fill, jean, jean.leftHand, verre, cuisine)
(15) action(verre.put, jean, jean.leftHand, cuisine)
(31) action(verre.watch, jean, cuisine)
---------------------------
select goal : 69 activ 0.5 rank 2
Action selected : 12
Execute action 12 action(cuisine.lavabo.fill, jean, jean.leftHand, verre, cuisine)
```

Table des figures

Bibliographie

[AB02] Allbeck (J.) et Badler (N.). – Toward representing agent behaviors modified by personality and emotion. *Dans : Workshop Embodied conversational agents - let's specify and evaluate them!* – Bologne, Italie, 2002.

[ACF01] Arikan (O.), Chenney (S.) et Forsyth (D. A.). – Efficient multi-agent path planning. *Dans : Computer Animation and Simulation '01*, éd. par Magnenat-Thalmann (Nadia) et Thalmann (Daniel). pp. 151–162. – Springer-Verlag Wien New York, 2001.

[All81] Allen (J. F.). – An interval based representation of temporal knowledge. *Dans : the seventh Internationnal Joint Conference on Artificial Intelligence*, pp. 221–226. – Août 1981.

[Ari84] Arijon (D.). – *Grammaire du langage filmé.* – Paris, Editions Dujarric, 1984.

[Arn94] Arnaldi (B.). – Modèles physiques pour l'animation. – Habilitation à dirriger des recherches, Université de Rennes I, 1994.

[BB98] Baerlocher (P.) et Boulic (R.). – Task-priority formulations for the kinematic control of highly redundant articulated structures. *Dans : IROS*, pp. 323–329. – Victoria, Canada, 1998.

[BBET97] Boulic (R.), Bécheiraz (P.), Emering (L.) et Thalmann (D.). – Integration of motion control techniques for virtual human and avatar real-time animation. *Dans : VRST'97*, éd. par Press (ACM), pp. 111–118. – Septembre 1997.

[BBT99] Bordeux (C.), Boulic (R.) et Thalmann (D.). – An efficient and flexible perception pipeline for autnonomous agents. *Dans : Eurographics'99*, pp. 20–30. – Milano, Italy, 1999.

[BCN97] Bouvier (E.), Cohen (E.) et Najman (L.). – From crowd simulation to airbag deployement : particle systems, a new paradigm of simulation. *Journal of Electronic Imaging*, vol. 6, n1, Janvier 1997, pp. 94–107.

[BF97] Blum (A. L.) et Furst (M. L.). – Fast planning through planning graph analysis. *Artificial Intelligence*, vol. 90, n1/2, Février 1997, pp. 281–300.

[BG01] Bonet (B.) et Geffner (H.). – Planning as heuristic search. *Artificial Intelligence*, vol. 129, n1/2, 2001, pp. 5–33.

[BH97] Brogan (D. C.) et Hodgins (J. K.). – Group behaviors for systems with significant dynamics. *Autonomous Robots*, vol. 4, 1997, pp. 137–153.

[BHPR97] Ballard (D. H.), Hayhoe (M. M.), Pook (P. K.) et Rao (Rajesh P. N.). – Deictic codes for the embodiment of cognition. *Behaviourial and brain sciences*, vol. 20, n4, 1997.

[BJ03] Brogan (D. C.) et Johnson (N. L.). – Realistic human walking paths. *Dans : Computer Animation and Social Agents (CASA'03).* – IEEE Computer Society Press, 2003.

[BL91] Barraquand (J.) et Latombe (J.-C.). – Robot motion planning : a distributed representation approach. *Internation Journal of Robotics Research*, vol. 10, n 6, 1991, pp. 628–649.

[Blu97] Blumberg (B. M.). – *Olds Tricks, New Dogs : Ethology and Interactive Creatures.* – Thèse de Doctorat, Massachussetts Institute of Technology, Février 1997.

[BMdOB03] Braun (A.), Musse (S. Raupp), de Oliveira (L. P. L.) et Bodmann (B. E. J.). – Modeling individual behaviors in crowd simulation. *Dans : Computer Animation and Social Agents (CASA'03).* – IEEE Computer Society Press, 2003.

[BMH98] Brogan (D. C.), Metoyer (R. A.) et Hodgins (J. K.). – Dynamically simulated characters in virtual environments. *IEEE Computer Graphics and Applications*, vol. 18, n 5, 1998, pp. 58–69.

[BMT96] Boulic (R.), Mas (R.) et Thalmann (D.). – A robust approach for the control of the center of mass with inverse kinetics. *Computer and Graphics*, vol. 20, n 5, septembre-octobre 1996, pp. 693–701.

[BPB99] Badler (N. I.), Palmer (M. S.) et Bindiganavale (R.). – Animation control for real-time virtual humans. *Communication of the ACM*, vol. 42, n 8, Août 1999.

[Bra87] Bratman (M. E.). – *Intentions, Plans, and Practical Reason.* – Harvard University Press, 1987.

[Bro91] Brooks (R. A.). – Intelligence without reason. *Dans : Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, éd. par Myopoulos (John) et Reiter (Ray). pp. 569–595. – Sydney, Australia, 1991.

[BSA+00] Bindiganavale (R.), Schuler (W.), Allbeck (J. M.), Badler (N. I.), Joshi (A. K.) et Palmer (M.). – Dynamically altering agent behaviors using natural language instructions. *Dans : Proceedings of the 4th International Conference on Autonomous Agents (AGENTS-00)*, éd. par Sierra (Carles), Maria (Gini) et Rosenschein (Jeffrey S.). pp. 293–300. – NY, Juin 2000.

[BT98] Bandi (S.) et Thalmann (D.). – Space discretization for efficient human navigation. *Computer Graphics Forum*, vol. 17, n 3, Septembre 1998, pp. 195–206.

[BW95] Badler (N. I.) et Webber (B. L.). – Planning and parallel transition networks: Animation's new frontiers. *Dans : Pacific Graphics '95.* – 1995.

[BY95] Boisonnat (J.-D.) et Yvinec (M.). – *Géométrie algorithmique.* – EDISCIENCE international, 1995, *Collection Informatique.*

[CCM02] Cavazza (M.), Charles (F.) et Mead (S. J.). – Planning characters' behaviour in interactive storytelling. *The Journal of Visualization and Computer Animation*, vol. 13, 2002, pp. 121–131.

[CCZB00] Chi (Diane), Costa (Monica), Zhao (Liwei) et Badler (Norm). – The emote model for effort and shape. *Dans : SIGGRAPH 2000*, pp. 173–182. – New Orleans, Louisiana, Juillet 2000.

[CD97] Chauffaut (A.) et Donikian (S.). – Gasp: a general animation and simulation platform. *Dans : 14th IMEKO World Congress (ISM-CR'97).* – Tampere, Finland, Juin 1997.

[CDD+02] Courty (N.), Devillers (F.), Donikian (S.), Lamarche (F.), Margery (D.) et Menardais (S.). – Towards believable autonomous actors in real-time applications. *Dans : IMAGINA'02.* – Monaco, 12-14 Février 2002.

[Che87] Chew (L. P.). – Constrained delaunay triangulations. *Dans : Proceedings of the third annual symposium on Computational geometry*. pp. 215–222. – ACM Press, 1987.

[CKP95] Cremer (J.), Kearney (J.) et Papelis (Y. E.). – HCSM: A framework for behavior and scenario in virtual environments. *Modeling and Computer Simulation*, vol. 5, n3, 1995, pp. 242–267.

[CLDM03] Courty (N.), Lamarche (F.), Donikian (S.) et Marchand (E.). – A cinematography system for virtual story telling. *Dans : 2nd International Conference on Virtual Storytelling*. – Toulouse, Novembre 2003.

[CLM+03] Charles (F.), Lozano (M.), Mead (S. J.), Bisquerra (A. F.) et Cavazza (M.). – Planning formalisms and authoring in interactive storytelling. *Dans : First International Conference on Technologies for Interactive Digital Storytelling and Entertainment*. – mai 2003.

[CLS03] Choi (M. G.), Lee (J.) et Shin (S. Y.). – Planning biped locomotion using motion capture data and probabilistic roadmaps. *ACM Transactions on Graphics*, vol. 22, n2, 2003, pp. 182–203.

[Cod89] Coderre (B.). – Modeling behavior in petworld. *Dans : Proceedings of the Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems (ALIFE '87)*, éd. par Langton (Christopher G.). pp. 407–420. – Redwood City, CA, USA, Septembre 1989.

[Cou02] Courty (N.). – *Animation référencée vision : de la tâche au comportement*. – Thèse de Doctorat, INSA de Rennes, Novembre 2002.

[CRR01] Caselli (S.), Reggiani (M.) et Rocchi (R.). – Heuristic methods for randomized path planning in potential fields, 2001.

[CT00] Caicedo (A.) et Thalmann (D.). – Virtual humanoids: Let them be autonomous without losing control. *Dans : Proc. 3IA 2000*. – Limoges, France, 2000.

[DCDK98] Donikian (S.), Chauffaut (A.), Duval (T.) et Kulpa (R.). – Gasp: from modular programming to distributed execution. *Dans : Computer Animation '98*. – Philadelphie, USA, Juin 1998.

[DD03] Devillers (F.) et Donikian (S.). – A scenario language to orchestrate virtual world evolution. *Dans : Proc. of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA'03)*, pp. 265–275. – 2003.

[DDLT02] Devillers (F.), Donikian (S.), Lamarche (F.) et Taille (J.F.). – A programming environment for behavioural animation. *The Journal of Visualization and Computer Animation*, vol. 13, 2002, pp. 263–274.

[Dev01] Devillers (F.). – *Langage de scénario pour les acteurs semi-autonomes*. – Thèse de Doctorat, Université de Rennes I, Septembre 2001.

[DF98] Decugis (V.) et Ferber (J.). – Action selection in an autonomous agent with a hierarchical distributed reactive planning architecture. *Dans : Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98)*, éd. par Sycara (Katia P.) et Wooldridge (Michael). pp. 354–361. – New York, Mai 1998.

[Don97] Donikian (S.). – Vuems : a virtual environment modeling system. *Dans : Computer Graphics Internationnal'97*. pp. 84–92. – Hasselt-Diepenbeek, Belgium, Juin 1997.

[Don01] Donikian (S.). – Hpts: a behaviour modelling language for autonomous agents. *Dans : Proceedings of the Fifth International Conference on Autonomous Agents*, éd. par Mül-

ler (Jörg P.), Andre (Elisabeth), Sen (Sandip) et Frasson (Claude). pp. 401–408. – Montreal, Canada, Mai 2001.

[Don03] Donikian (S.). – Dramachina: an authoring tool to write interactive fictions. *Dans : TIDSE'03, first International Conference on Technologies for Interactive Digital Storytelling and Entertainment*, éd. par Gobel (S.), Braun (N.) et Spierling (U.). – Darmstadt, Germany, Mars 2003.

[DTJ00] Dikjstra (J.), Timmermans (H. J. P.) et Jessurun (A. J.). – A multi-agent cellular automata system for visualising simulated pedestrian activity. *Dans : Theoreteical and Practical Issues on Cellular Automata - Proceedings on the 4th International Conference on Cellular Automata for research and Industry.* pp. 29–36. – Karlsruhe, 2000.

[EHN94] Erol (K.), Hendler (J. A.) et Nau (D. S.). – UMCP: A sound and complete procedure for hierarchical task-network planning. *Dans : Artificial Intelligence Planning Systems*, pp. 249–254. – 1994.

[FBT99] Farenc (N.), Boulic (R.) et Thalmann (D.). – An informed environment dedicated to the simulation of virtual humans in urban context. *Dans : Computer Graphics Forum (Eurographics '99)*, éd. par Brunet (P.) et Scopigno (R.). pp. 309–318. – The Eurographics Association and Blackwell Publishers, 1999.

[Feu00] Feurtey (F.). – *Simulating the Collision Avoidance Behavior of Pedestrians.* – Thèse de Master, The University of Tokyo, School of Engineering, 2000.

[FN71] Fikes (R. E.) et Nilsson (N. J.). – Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, vol. 2, n3/4, 1971.

[Fru71] Fruin (J. J.). – *Predestrian Planning and Deasign.* – Metropolitan Association of Urban Designers and Environmental Planners, 1971.

[FTT99] Funge (J.), Tu (X.) et Terzopoulos (D.). – Cognitive modeling: Knowledge, reasoning and planning for intelligent characters. *Dans : SIGGRAPH 99.* – Los Angeles, Août 1999.

[Fun98] Funge (J.). – *Making Them Behave: Cognitive Models for Computer Animation.* – Thèse de Doctorat, University of Toronto, 1998.

[Fun99] Funge (J.). – Representing knowledge within the situation calculus using interval-valued epitemic fluents. *Journal of Reliable ComputingJournal of Reliable Computing*, vol. 5, n1, 1999.

[GBR+95] Granieri (J. P.), Becket (W.), Reich (B. D.), Crabtree (J.) et Badler (N. I.). – Behaviroal control for real-time simulated human agents. *Dans : Symposium on Interactive 3D graphics*, pp. 173–180. – 1995.

[Gib86] Gibson (J.). – *The ecological approach to visual perception.* – Hillsdale, USA, Lawrence Erlbaum Associates, Inc, 1986.

[Gib97] Gibson (J.). – The theory of affordances. *Dans : Perceiving, acting and knowing - Towards an ecological psychology*, éd. par et J. Bransford (R. Shaw). pp. 67–82. – Houghton-Mifflin, 1997.

[Gof71] Goffman (E.). – *Relations in public : microstudies of the public order.* – New York : Basic books, 1971.

[GTH98] Grzeszczuk (R.), Terzopoulos (D.) et Hinton (G.). – NeuroAnimator: Fast neural network emulation and control of physics-based models. *Dans : SIGGRAPH 98 Conference*

Proceedings, éd. par Cohen (Michael). ACM SIGGRAPH, pp. 9–20. – Addison Wesley, Juillet 1998.

[Hel01] Helbing (D.). – Traffic and related self-driven many-particle systems. *Reviews of Modern Physics*, no73, 2001, pp. 1067–1141.

[HFV00] Helbing (D.), Farkas (I.) et Vicsek (T.). – Simulating dynamical features of escape panic. *Nature*, vol. 407, 2000, pp. 487–490.

[HK02] Hostetler (T. R.) et Kearney (J. K.). – Strolling down the avenue with a few close friends. *Dans : Third Irish Workshop on Computer graphics*, pp. 7–14. – 2002.

[HKL+99] Hoff III (K. E.), Keyser (J.), Lin (M.), Manocha (D.) et Culver (T.). – Fast computation of generalized Voronoi diagrams using graphics hardware. *Computer Graphics*, vol. 33, 1999, pp. 277–286.

[KAB+98] Koga (Y.), Annesley (G.), Becker (C.), Svihura (M.) et Zhu (D.). – On intelligent digital actors. – Imagina'98 White papers, 1998.

[Kal01] Kallmann (M.). – *Object Interaction in Real-Time Virtual Environments.* – Lausanne, Thèse de Doctorat, Ecole Polytechnique Fédérale de Lausanne, 2001.

[KBT03] Kallmann (M.), Bieri (H.) et Thalmann (D.). – Fully dynamic constrained delaunay triangulations. *Geometric Modelling for Scientific Visualization*, 2003.

[KKWH01] Kulkla (R.), Kerridge (J.), Willis (A.) et Hine (J.). – *PEDFLOW: Development of an autonomous Agent Model of Pedestrian Flow.* – Rapport technique, Redwood House, 66 Spylaw Road, Edimburgh EH10 5BR, United Kingdoms., Transport Research Institute, Napier University, 2001.

[KMCT00] Kallmann (M.), Monzani (J.-S.), Caicedo (A.) et Thalmann (D.). – ACE: A platform for the real time simulation of virtual human agents. *Dans : Computer Animation and Simulation '00*, éd. par Magnenat-Thalmann (Nadia), Thalmann (Daniel) et Arnaldi (Bruno). pp. 73–84. – Springer-Verlag Wien New York, 2000.

[Kor85] Korpf (R. E.). – Depth-first iterative-deepening : An optimal admissible tree search. *Artificial intelligence*, vol. 27, n1, 1985, pp. 97–109.

[KT02] Kallmann (M.) et Thalmann (D.). – Modeling behaviors of interactive objects for real time virtual environments. *Journal of Visual Languages and Computing*, vol. 13, 2002, pp. 177–195.

[Kuf98] Kuffner (J. J.). – Goal-directed navigation for animated characters using real-time path planning and control. *Lecture Notes in Computer Science*, vol. 1537, 1998, pp. 171–179.

[Kuf99] Kuffner (J. J.). – *Autonomous agents for real-time animation.* – Thèse de Doctorat, Staford University, Décembre 1999.

[KVD03] Klinger (E.) et Viaud-Delmon (I.). – *Le traité de la réalité virtuelle, 2^e édition*, chap. Réalité virtuelle et psychatrie, pp. 297–324. – Ecole des mines de Paris - Les Presses, 2003, volume 2.

[LA98] Logan (B.) et Alechina (N.). – A* with bounded costs. *Dans : Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pp. 444–449. – 1998.

[Lai01] Laird (J. E.). – It knows what you're going to do: Adding anticipation to a quakebot. *Dans : Fith International Conference on Autonomous Agents.* – Montreal, Canada, Mai 2001.

[Lat91] Latombe (J.-C.). – *Robot Motion Planning.* – Boston, Boston: Kluwer Academic Publishers, 1991.

[Lau83] Laumond (J.P.). – odel structuring and concept recognition: Two aspects of learning for a mobile robot. *Dans : Intenational Joint Conference on Artificial Intelligence (JCAI)*, pp. 839–841. – Aout 1983.

[Lau89] Laumond (J.P.). – A learning system for the understanding of a mobile robot environment. *Dans : Machine and Human Learning*, éd. par Kodratoff (Y.), pp. 161–171. – 1989.

[LD01] Lamarche (F.) et Donikian (S.). – The orchestration of behaviours using resources and priority levels. *Dans : Computer Animation and Simulation '01*, éd. par Magnenat-Thalmann (Nadia) et Thalmann (Daniel). pp. 171–182. – Springer-Verlag, 2-3 Septembre 2001.

[LD02] Lamarche (F.) et Donikian (S.). – Automatic orchestration of behaviours through the management of resources and priority levels. *Dans : Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'02)*, éd. par Gini (Maria), Ishida (Toru), Castelfranchi (Cristiano) et Johnson (W. Lewis). pp. 1309–1316. – ACM Press, Juillet 2002.

[LMM03] Loscos (C.), Marchal (D.) et Meyer (A.). – Intuitive behavior in dense urban environments using local laws. *Dans : Theory and Pratice of Computer Graphics (TPCG'03).* – Birmingham, UK, Juin 2003.

[LNR87] Laird (J. E.), Newell (A.) et Rosenbloom (P. S.). – Soar: An architecture for general intelligence. *Artificial intelligence*, vol. 33, n3, 1987, pp. 1–64.

[LRL+97] Levesque (H.), Reiter (R.), Lesperance (Y.), Lin (F.) et Scherl (R.). – GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, vol. 31, 1997, pp. 59–84.

[LTC+00] Lester (J.), Towns (S. G.), Callaway (C. B.), Voerman (J. L.) et FitzGerald (P. J.). – *Embodied conversational agents*, chap. Deictic and Emotive Communication in Animated Pedagogical Agents, pp. 123–154. – The MIT Press, 2000.

[LW92] Lee (J. R. E.) et Watson (R.). – *Regards et habitudes des passants.* – 1992, *Les anales de la recherche urbaine*, volume 57-58.

[MAC+02] Margery (D.), Arnaldi (B.), Chauffaut (A.), Donikian (S.) et Duval (T.). – Openmask: Multi-threaded or modular animation and simulation kernel or kit : a general introduction. *Dans : VRIC 2002*, éd. par Richir (S.), Richard (P.) et Taravel (B.), pp. 101–110. – Juin 2002.

[Mae90] Maes (P.). – Situated agents can have goals. *Dans : Designing Autonomous Agents*, éd. par Maes (Patti). pp. 49–70. – MIT Press, 1990.

[Mal97] Mallot (H. A.). – Behavior-oriented approaches to cognition : theoritical perspectives. *Theory in biosciences*, vol. 116, 1997, pp. 196–200.

[MCT01] Monzani (J.-S.), Caicedo (A.) et Thalmann (D.). – Integrating behavioural animation techniques. *Dans : Eurographics 2001.* – 2001.

[MD98] Moreau (G.) et Donikian (S.). – From psychological an real-time interaction requirements to behavioural simulation. *Dans : Computer Animation and Simulation '98*, éd. par Arnaldi (B.) et Hegron (G.). pp. 29–44. – Springer-Verlag Wien New York, 1998. Proceedings of the Eurographics Workshop in Lisbon, Portugal, August 31-September 1, 1998.

[Men03] Menardais (S.). – *Fusion et adaptation en temps réel de mouvements acquis pour l'animation d'humanoïdes synthétiques.* – Thèse de Doctorat, Université de Rennes I, Janvier 2003.

[MGD99] Musse (S.Raupp), Garat (F.) et D.Thalmann. – Guiding and interacting with virtual crowds in real-time. *Dans : Eurographics Workshop on Animation and Simulation '99 (CAS '99).* pp. 23–34. – Springer, 1999.

[MGT99] Musse (S. Raupp), Garat (F.) et Thalmann (D.). – Guiding and interacting with virtual crowds in real-time. *Dans : Eurographics Workshop on Animation and Simulation*, pp. 23–34. – Milan, Italy, 1999.

[MH69] McCarthy (J.) et Hayes (P. J.). – Some philosophical problems from the standpoint of artificial intelligence. *Dans : Machine Intelligence 4*, éd. par Meltzer (B.) et Michie (D.), pp. 463–502. – Edinburgh University Press, 1969. reprinted in McC90.

[MH03] Metoyer (R. A.) et Hodgins (J. K.). – Reactive pedestrian path following from examples. *Dans : Computer Animation and Social Agents (CASA'03).* – IEEE Computer Society Press, 2003.

[Mon02] Monzani (J.S.). – *An architecture for the Behavioral Animation of Virtual Humans.* – Thèse de Doctorat, Ecole polytechnique de Lausanne, Juillet 2002.

[Mor98] Moreau (G.). – *Modélisation du comportement pour la simulation interactive : application au trafic routier multimodal.* – IRISA, Thèse de Doctorat, Université de Rennes I, Novembre 1998.

[MRHA00] Milazzo (J. S.), Rouphail (N. M.), Hummer (J. E.) et Allen (D. P.). – Quality of service for uninterrupted pedestrian facilities. *Dans : Highway Capacity Manual.* – Washington, DC, 2000.

[MS02] Mateas (M.) et Stern (A.). – *Architecture, Authorial Idioms and Early Observations of the Interactive Drama Facade.* – Rapport technique n CMU-CS-02-198, Carnegie Mellon University, Pittsburgh, School of Computer Science, Décembre 2002.

[MT97] Musse (S.Raupp) et Thalmann (D.). – A model of human crowd behavior : Group inter-relationship and collision detection analysis. *Dans : Computer Animation and Simulation '97.* pp. 39–51. – Springer Verlag, 1997.

[MT01] Musse (S. Raupp) et Thalmann (D.). – Hierachical model for real time simulation of virtual huamn crowds. *IEEE Transaction on Visualization and Computer Graphics*, vol. 7, n2, Avril-Juin 2001.

[New90] Newell (A.). – *Unified theories of cognition.* – Harward University Press, 1990.

[NT97] Noser (H.) et Thalmann (D.). – Sensor based synthetic actors in a tennis game simulation. *Dans : Computer Graphics International '97.* pp. 189–198. – IEEE Computer Society Press, 1997.

[O'H00] O'Hara (N.). – K-d tree neighbour finding for the flocking algorithm. *Dans : Pacific Graphics.* – 2000.

[O'H02] O'Hara (N.). – Hierarchical impostors for the flocking algorithm in three dimensional space. *Dans : Third Irish Workshop on Computer Graphics.* – 2002.

[Ove02] Overmars (M. H.). – Recent developments in motion planning. *Dans : International Conference on Computational Science (3)*, pp. 3–13. – 2002.

[PLS03] Pettré (J.), Laumond (J.-P.) et Siméon (T.). – A 2-stages locomotion planner for

digital actors. *Dans : Proc. of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA'03)*, pp. 258–264. – 2003.

[Rey87] Reynolds (C. W.). – Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, vol. 21, n 4, 1987, pp. 25–34.

[Rey99] Reynolds (C. W.). – Steering behaviors for autonomous characters. *Dans : Game Developers Conference 1999*. – 1999.

[Rey00] Reynolds (C. W.). – Interaction with groups of autonomous characters. *Dans : Game Developers Conference 2000*. – 2000.

[RG91] Rao (A. S.) et Georgeff (M. P.). – Modeling rational agents within a BDI-architecture. *Dans : Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, éd. par Allen (James), Fikes (Richard) et Sandewall (Erik). pp. 473–484. – Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, 1991.

[RG95] Rao (A. S.) et Georgeff (M. P.). – BDI-agents: from theory to practice. *Dans : Proceedings of the First Intl. Conference on Multiagent Systems*. – San Francisco, 1995.

[Rho96] Rhodes (B. J.). – *PHISH-Nets : Planning heuristically In Situated Hybrid Networks*. – Thèse de Master, Massachusets Institute of Technology, 1996.

[RKBB94] Reich (B.), Ko (H.), Becket (W.) et Badler (N. I.). – Terrain reasoning for human locomotion. *Dans : Computer Animation '94*. pp. 996–1005. – IEEE Computer Society Press, 1994.

[RM94] Reinefeld (A.) et Marsland (T. A.). – Enhanced iterative-deepening search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, n 7, 1994, pp. 701–710.

[RQ98] Relieu (M.) et Quéré (L.). – *Les risques urbains : Acteurs, systèmes de prévention*, chap. Mobilité, perception et sécurité dans les espaces urbains. – 1998, *Anthropos : Collection ville*.

[Sch99] Schuler (W.). – Preserving semantic dependencies in synchronous tree adjoining grammar. *Dans : The 37th Annual Meeting of the Association for Computational Linguistics*, pp. 88–95. – 1999.

[SL03] Scherl (R.) et Levesque (H. J.). – Knowledge, action, and the frame problem. *Artificial Intelligence*, no144, 2003, pp. 1–39.

[TB96] Thrun (S.) et Bücken (A.). – Integrating grid-based and topological maps for mobile robot navigation. *Dans : Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*. pp. 944–951. – Menlo Park, Août 1996.

[TC00] Tecchia (F.) et Chrysanthou (Y.). – Real time rendering of densely populated urban environments. *Dans : RenderingTechniques '00 (10th EurographicsWorkshop on Rendering)*. pp. 45–56. – Brno, Czech Republic, 2000.

[TCR+96] Trias (T.), Chopra (S.), Reich (B.), Moore (M.), Badler (N.), Webber (B.) et Geib (C.). – Decision networks for integrating the behaviors of virtual agents and avatars. *Dans : IEEE Virtual Reality Annual International Symposium (VRAIS '96)*. pp. 156–162. – Los Alamitos, CA, 1996.

[TD00a] Thomas (G.) et Donikian (S.). – Modelling virtual cities dedicated to behavioral

animation. *Dans : EUROGRAPHICS'2000*, éd. par Gross (M.) et Hopgood (F.). – Interlaken, Switzerland, Août 2000.

[TD00b] Thomas (G.) et Donikian (S.). – Virtual humans animation in informed urban environments. *Dans : Proceedings of the Conference on Computer Animation (CA-00)*. pp. 112–121. – Los Alamitos, CA, Mai 2000.

[Tho99] Thomas (G.). – *Environnements virtuels urbains : modélisation des informations nécessaires à la simulation de piétons*. – Thèse de Doctorat, Université de Rennes I, 1999.

[TIJ+98] Terzopoulos (D.), In (X.), Joshi (K.), Reynolds (C. W.) et Simpson (T.). – Behavioral modeling and animation (panel): past, present, and future. *Dans : Conference abstracts and applications: SIGGRAPH 98*, pp. 209–211. – Orlando, USA, Juillet 1998.

[TPH02] Thanagarajah (J.), Padgham (L.) et Harland (J.). – Representation and reasoning for goals in bdi agents. *Dans : Australasian Conference on Computer Science*. – Melbourne, Australie, Janvier 2002.

[tRGM03] traun (D.), Rickel (J.), Gratch (J.) et Marsella (S.). – Negociation over tasks in hybrid human-agent teams for simulation-based training. *Dans : Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'03)*. pp. 441–456. – Melbourne, Australia, Juillet 2003.

[Tyr94] Tyrell (T.). – An evaluation of maes's bottom-up méchanism for action selection. *Adaptative Behavior*, vol. 2, n4, 1994, pp. 307–348.

[vF93] van de Panne (M.) et Fiume (E.). – Sensor-actuator networks. *Dans : Computer Graphics (SIGGRAPH '93 Proceedings)*, éd. par Kajiya (James T.), pp. 335–342. – Août 1993.

[WM03] Wiener (J. M.) et Mallot (H. A.). – *Fine-to-Coarse Route Planning and Navigation in Regionalized Environments*. – Rapport technique n115, Juillet 2003.

[YSSB98] Yahja (A.), Stentz (A.), Singh (S.) et Brumitt (B. L.). – Framed-quadtree path planning for mobile robots operating in sparse environnements. *Dans : IEEE Conference on Robotics and Automation (ICRA)*. – Leuven, Belgium, Mai 1998.

[Zie98] Ziemke (T.). – Adaptative behavior in autonomous agents. *Presence*, vol. 7, n6, Décembre 1998, pp. 564–587.

Publications

Revue

Devillers (F.), Donikian (S.), Lamarche (F.) et Taille (J.F.). – A programming environment for behavioural animation. *The Journal of Visualization and Computer Animation*, vol. 13, 2002, pp. 263–274.

Conférences internationales

Lamarche (F.) et Donikian (S.). – Automatic orchestration of behaviours through the management of resources and priority levels. *Dans : Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'02)*. pp. 1309–1316. – ACM Press, Juillet 2002.

Courty (N.), Lamarche (F.), Donikian (S.) et Marchand (E.). – A cinematography system for virtual story telling. *Dans : 2nd International Conference on Virtual Storytelling.* – Toulouse, Novembre 2003.

Workshop international

Lamarche (F.) et Donikian (S.). – The orchestration of behaviours using resources and priority levels. *Dans : Computer Animation and Simulation '01*, éd. par Magnenat-Thalmann (Nadia) et Thalmann (Daniel). pp. 171–182. – Springer-Verlag, 2-3 Septembre 2001.

Séminaire

Courty (N.), Devillers (F.), Donikian (S.), Lamarche (F.), Margery (D.) et Menardais (S.). – Towards believable autonomous actors in real-time applications. *Dans : IMAGINA'02.* – Monaco, 12-14 Février 2002.

Humanoïdes virtuels, réaction et cognition : une architecture pour leur autonomie

Les êtres vivants sont caractérisés par leur rapport avec l'environnement dans lequel ils évoluent. Ils sont dotés de capacités de perception, d'action et de décision. L'animation comportementale s'inspire de cette architecture pour peupler des environnements virtuels avec des entités autonomes à l'image des organismes vivants. Dans ce domaine, il est un être vivant qui retient particulièrement l'attention : l'être humain. Dans cette thèse, nous proposons une architecture pour la description du comportement des humanoïdes virtuels. Dans un premier temps, nous étudions la relation entre l'humanoïde et son environnement dans le cadre de la navigation. Nous proposons un système de subdivision spatiale permettant d'extraire des propriétés topologiques à partir d'un environnement géométrique. Ces informations sont ensuite utilisées pour créer un algorithme de planification de chemin hiérarchique. Nous proposons, dans la continuité, un algorithme de navigation s'inspirant d'études sur le comportement piétonnier et permettant de simuler, en temps réel, des foules composées de plusieurs centaines de piétons. Dans un second temps, nous étudions les composantes réactives et cognitives du comportement. Nous proposons un modèle réactif dont la propriété est de synchroniser et d'adapter automatiquement le déroulement simultané de plusieurs comportements. Le modèle cognitif peut alors exploiter cette propriété pour choisir des actions permettant à l'humanoïde de réaliser un but. Ce modèle se base sur un langage orienté objet permettant de décrire le monde sous la forme des interactions qu'il offre aux humanoïdes. Différentes applications ont validé les travaux réalisés dans cette thèse : simulation de foule, fiction interactive et cinématographie virtuelle.

mots clefs : réalité virtuelle, animation comportementale, modèles réactifs, modèles cognitifs, subdivision spatiale, planification de chemin, navigation.

Virtual humans, reaction and cognition : an architecture for their autonomy

Living beings are characterized by their relationship with their environment. They can perceive, act and decide. Behavioral animation uses this architecture to model autonomous entities, similar to living beings, populating virtual environments. In this thesis, we propose an architecture to describe virtual humans behavior. First, we study the relationship between virtual humans and their environment through the navigation process. We propose a spatial subdivision algorithm extracting topological properties from the geometry of the environment. This information is used to create a hierarchical path-planning algorithm. Then, a navigation algorithm, inspired by studies on pedestrian behavior, is proposed to simulate in real time crowds composed of several hundreds of autonomous entities. Secondly, we study reactive and cognitive behaviors. We propose a reactive model, which is able to handle automatically the synchronization and adaptation of several parallel behaviors. The cognitive model is therefore able to select actions to achieve a given goal. This model is based on an object oriented language used to describe the interactions offered by the environment. Different applications have been realized to validate the models in the field of crowd simulation, interactive fiction and virtual cinematography.

keywords: virtual reality, behavioral animation, reactive model, cognitive model, spatial subdivision, path-planning, navigation.