Un modèle de simulation agent basé sur les interactions

JC. Routier[1], P. Mathieu[1], and P. Urro[2]

[1] Laboratoire d'Informatique Fondamentale de Lille
CNRS upresa 8022
Cité Scientifique Villeneuve d'Ascq, France
routier@lifl.fr, mathieu@lifl.fr
[2] Cryo Interactive Entertainment
p.urro@cryo-interactive.com

Résumé

Le domaine des jeux vidéos est un secteur économique important et, dans la mesure où il concerne de nombreux capitaux, peut être considéré comme un élément moteur pour les technologies informatiques. Après le graphisme, il semble qu'actuellement le prochain challenge est d'introduire plus d'intelligence dans les jeux. Le concept de multi-agents constitue un outil puissant pour y parvenir.

Nous présentons ici le résultat de notre collaboration avec l'éditeur de jeux vidéo Cryo Interactive qui est l'un des leaders français du domaine. L'objectif était de fournir un cadre pour la modélisation de comportements artificiels pour les jeux vidéo et pour leur évolution. Une contrainte importante était que la simulation se comporte de manière rationnelle, c'est-à-dire simule ce qui pourrait se passer dans le monde réel.

Pour réaliser cela, nous avons proposé une description du monde où toute entité est un agent et chaque action qui peut être accomplie est décrite par une *interaction* entre deux agents, l'un qui l'effectue et l'autre qui la subit.

Les interactions sont au centre de notre travail. Elles constituent une manière de représenter la connaissance sur le monde et en même temps elles contiennent la dynamique des simulations. Nous montrons dans cet article qu'elles forment un outil puissant avec peu de limites d'expressivité dans les simulations.

Chaque agent animé possède un moteur de comportement qui s'appuie sur des interactions. L'agent a une mémoire, construit des plans et les modifie dynamiquement. Succinctement, la tâche principale du moteur est de faire correspondre un agent "qui peut effectuer" avec un agent "qui peut subir" une interaction. Ce principe permet de gérer des plans d'actions et d'assurer la résolution des buts des différents agents.

Ce travail a abouti à la réalisation d'une application qui exploite ce modèle. Elle le valide et montre qu'un monde artificiel peut facilement être modélisé à partir de cette approche par interactions. Nous terminons en montrant que des situations complexes d'interactions entre agents sont faciles à construire avec notre modèle.

1 Introduction

Parmi les champs d'application de l'informatique, les jeux ont toujours occupé une place particulière. Dans un premier temps, l'aspect ludique peut lancer un certain discrédit sur les applications et les travaux dans le domaine. Il est clair néanmoins que l'industrie du jeu a été et reste un catalyseur pour le développement de technologies matérielles et logicielles. Le graphisme en est probablement la meilleure illustration. Les sommes importantes impliquées dans l'industrie du jeu laissent présager qu'il n'en sera pas autrement dans les prochaines années.

L'intelligence artificielle a très tôt été attirée par les jeux. Les premiers défis ont concernés les classiques, mais non triviaux, jeux à deux joueurs. Le but est alors de construire un programme qui, pour un jeu particulier, joue le mieux possible et, pour montrer son "intelligence", rivalise avec les meilleurs joueurs humains. Si des solutions de haut niveau ont été trouvées, ce n'est pas encore le cas pour tous les jeux (tel que le Go par exemple).

Mais ce ne sont pas vraiment ces jeux qui sont concernés pas l'industrie du jeu vidéo. Les thématiques de cette industrie sont plutôt la simulation, le sport, la stratégie, le jeu de rôles, l'action, etc. Le point commun de toutes ces applications est un *monde artificiel* avec de nombreux agents qui interagissent à l'écran. Les joueurs, de plus en plus habitués, demandent toujours plus d'intelligence à leurs adversaires ou partenaires virtuels :

* Ce travail est soutenu par un contrat PRIAMM avec la société Cryo Interactive Entertainment et est financé par le Centre National de la Cinématographie et le Secrétariat d'État à l'industrie

– la partie “non – gérée par le – joueur”, c’est-à-dire l’univers du jeu, doit avoir un comportement raisonnable,
– l’adversaire géré par le programme doit être un adversaire coriace mais ne doit pas tricher pour y parvenir,
– les personnages non joueurs (PNJ) doivent avoir un comportement rationnel et intelligents. Cela ne concerne pas seulement les comportements individuels mais aussi les comportements collectifs.

Pour toutes ces raisons il est clair que l’intelligence artificielle peut apporter (et c’est parfois déjà partiellement le cas) sa contribution. Mais il est clair également que cette contribution peut être renforcée par l’utilisation de concepts multi-agents. L’idée d’utiliser les agents pour modéliser un monde artificiel et très naturelle et n’est évidemment pas nouvelle ([4, 1]).

Nous présentons ici notre point de vue pour la construction concrète de mondes virtuels, qui a été appliqué dans le cadre d’une collaboration avec l’éditeur de jeux vidéo Cryo Interactive Entertainment. Dans ce travail, la notion d’agent opére à deux niveaux différents. D’abord, comme cela peut sembler le plus naturel, les agents sont utilisés pour modéliser les différents personnages de la simulation et leur donner un comportement autonome et raisonnable. Ensuite, la notion d’agent est utilisée pour modéliser l’ensemble de l’univers du jeu.

Le modèle que nous proposons pour la conception de mondes artificiels, est basé sur la notion d’interaction. Chaque entité qui apparaît dans la simulation (un personnage, un chien ou une pomme) est considérée comme un agent. Chaque action qui peut être accomplie dans l’univers de la simulation est décrite par une interaction entre deux agents, l’un qui effectue l’interaction et l’autre qui la subit. Les capacités des agents sont définies par ces interactions. Ensuite, pour chaque agent, un moteur gère son comportement. Le principe de base est de faire coïncider un agent “qui peut effectuer” avec un agent “qui peut subir”. Les interactions contiennent la représentation de la connaissance sur le monde et en même temps influent sur la dynamique des simulations. La puissance d’expression des interactions est élevée. Les interactions sont un moyen de “typer” les agents. En effet vous pouvez caractériser un agent par les interactions qu’il peut effectuer ou subir. L’avantage est que vous pouvez, par exemple, indiquer que vous voulez un agent qui peut être mangé (donc un agent “mangeable”) sans spécifier un agent en particulier. Vous pouvez aussi avoir des interactions de contrôle pour gérer l’état interne de votre agent, comme sa mémoire par exemple. Cette richesse nous permet alors de définir des comportements complexes.

Ce travail a été concrétisé par la réalisation d’une application qui valide ces notions. Elle permet la réalisation de simulations en général et la conception de jeux à destination des game designers. Elle permet de vérifier que les comportement modélisés sont rationnels.

Nous présentons dans un premier temps les buts visés, notamment ce qu’une société de jeux vidéo comme Cryo attend d’une simulation de mondes virtuels. Ensuite, nous présentons le résultat de nos recherches pour répondre à cette attente : les agents, la notion d’interaction et le moteur de comportement.

2 Objectifs

Le but principal est de proposer un modèle générique pour des mondes artificiels où différentes entités peuvent être identifiées. Celles-ci ont des comportements et des capacités qui peuvent être très différentes. Les jeux les plus directement concernés sont ceux de type jeux de rôles, aventure/action et simulation/stratégie. Bien que cela n’exclut pas que d’autres jeux puissent être abordés.

Modéliser le monde virtuel L’objectif principal ici est d’offrir la possibilité de représenter l’univers du jeu ainsi que les actions permises dans cet univers. Cela doit être réalisé pour chaque entité, qu’elle soit animée ou pas.

Pour chacune de ces entités, il est nécessaire de décrire ce qu’elle peut faire et comment on peut l’utiliser. Il n’y a *a priori* pas de limitation sur les actions qui peuvent être conçues.

Cela implique qu’il doit y avoir un moyen pour prendre en compte la connaissance sur l’entité de telle sorte que les autres agents aient un moyen de savoir si ils peuvent ou non l’“utiliser”.

Enfin, dans ce monde, les entités peuvent avoir des comportements et des buts pour éventuellement interagir et coopérer.

Disposer d'un moteur Une fois le monde décrit, il doit être possible de le faire évoluer. Cela signifie que les entités doivent se comporter en fonction de leurs caractéristiques. Les entités qui ont un but doivent travailler à son accomplissement et les autres doivent être "prêtes" à réagir aux sollicitations.
Bien sûr, il est requis que les comportement soient aussi rationnels que possible. Cela signifie qu'ils simulent au mieux ce qui se produirait dans la réalité.
Comme exemple, considérons un personnage dont le but est de prendre un objet dans une pièce derrière une porte cadenassée. Supposons qu'alors qu'il cherche après la clef qui convient, un autre agent ouvre cette porte. Si, au cours de sa recherche, il est amené à voir la porte maintenant ouverte, il semble raisonnable qu'il stoppe cette recherche de la clef pour entrer dans la pièce et prendre directement l'objet visé.
En conséquence, ce personnage doit être capable de construire un plan (tel que "chercher la clef pour ouvrir la porte") mais doit également être capable d'adapter ce plan ("arrêter de chercher et entrer dans la pièce si la porte est maintenant ouverte") (Cf. Figure 1).

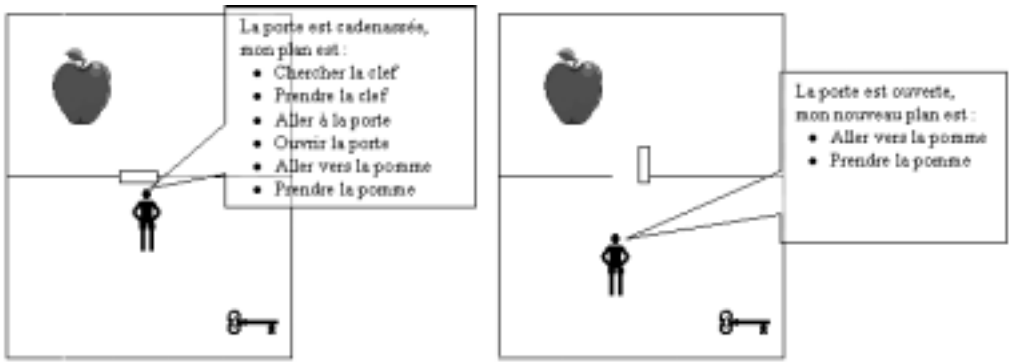

Fig. 1. À gauche, la porte est fermée le plan initial nécessite la recherche de la clef. À droite, alors que l'agent recherche la clef, la porte est ouverte, le plan est adapté.

Notre proposition Nous proposons d'utiliser les concepts d'interaction entre agents pour atteindre ces objectifs. Ce que nous voulons c'est fournir un cadre uniforme pour concevoir le monde artificiel et permettre sa simulation. Ce cadre uniforme aboutit naturellement à un système informatique de description comportementale permettant d'établir des bibliothèques de comportements utilisables pour différents jeux.

Environnement Nous ne parlerons pas longtemps de l'*environnement* ici. Disons simplement que puisque nous voulons modéliser un monde virtuel, nous avons besoin d'un support spatial et temporel. Cela est réalisé à travers l'environnement, qui en plus fournit un outil pour visualiser les agents (et ainsi vérifier le comportement de la simulation). Ainsi l'environnement gère la position des différents agents et définit la carte (ie. "où sont les murs et les pièces").

3 Tout est Agent

Dans un souci d'uniformité de l'ensemble des éléments, nous avons décidé que toute entité qui peut intervenir dans une simulation est un agent : les pommes, les clefs, les voitures, les chiens, les humains, etc. Cela signifie que nous disposons d'une structure uniforme pour représenter toutes ces entités.

3.1 Attributs d'un agent

Un agent est défini non seulement par des *propriétés* mais aussi par des *interactions* (nous définirons plus précisément cette notion par la suite). Ces interactions sont de deux sortes : celles qu'un agent peut accomplir et celles qu'un agent peut subir.

En fait, lorsque vous construisez un agent, vous devez définir une classe d'agent, et comme classiquement en programmation objet, vous pouvez ensuite construire de nouvelles classes à partir de celles existantes grâce à un mécanisme d'héritage. Ainsi un agent est une instance d'une classe. Lorsque vous définissez une classe, vous faites des distinctions parmi les propriétés et les interactions, certaines sont dites *générales* et d'autres *spécifiques*.

Les propriétés et interactions générales sont en quelque sorte comme les attributs `static` en JAVA ou en C++ : toutes les instances d'une classe utilisent les mêmes. Vous pouvez décider lors de la conception de votre monde que toutes les pommes sont mangées de la même manière, ou que pour toutes les voitures la propriété `couleur` doit valoir rouge.

Les interactions spécifiques permettent de distinguer les agents (instances). Lorsque vous créez un agent vous pouvez (et non "devez" puisque des valeurs par défaut peuvent être fournies) les spécifier en précisant les propriétés et interactions spécifiques. Vous pouvez indiquer par exemple que la propriété `nom` pour ce Hobbit vaut `Merry` et vaut `Sam` pour cet autre, ou que vous devez utiliser cette clef rouge ci pour appliquer l'interaction `décadenasser` sur cette porte.

Nous avons indiqué que tout est agent mais nous devons faire une distinction entre deux classes principales : les agents animés et les autres. Les agents non animés ne possèdent pas d'autres caractéristiques que celles décrites précédemment. Nous allons présenter maintenant les agents animés.

3.2 Agents animés et buts

Il existe une classe particulière d'agents : les agents *animés*. Cela ne signifie en fait pas nécessairement qu'ils bougent, un arbre pourrait être un agent animé. Ces agents sont caractérisés par le fait qu'ils ont un comportement qui est défini par l'existence de buts que ces agents doivent satisfaire. En conséquence, ce sont ces agents qui contribuent à la dynamicité du monde et de la simulation. Dans la mesure où ils ont des buts, ils vont agir (ou "interagir") pour accomplir ces buts. Le but d'un agent `arbre` pourrait ainsi être de faire pousser des pommes et de les laisser tomber sur le sol.

Ces agents animés ont des attributs supplémentaires appelées *priorité*. Ceux-ci sont utilisés pour personnaliser les agents et peuvent être vus comme des préférences servant à influencer le moteur dans le choix de la prochaine action à prendre en compte. Ceci est très pratique pour obtenir des comportement différents entre des agents et leur donner une forme de personnalité.

Un agent (animé) peut avoir plusieurs buts. C'est le rôle du moteur (il y en a un pour chaque agent (animé[1])) de décider quel est le but courant qui doit être sélectionné et quelle est l'action suivante à accomplir pour le résoudre. Il est possible d'attribuer des priorités intrinsèques aux buts pour indiquer certaines préférences de l'agent.

Il existe deux sortes de buts : les exceptionnels et les récurrents. Les buts exceptionnels sont accomplis une seule fois par les agents et les récurrents peuvent apparaître plus ou moins régulièrement. Un but exceptionnel pourrait être "aller chercher cette clef". Un but récurrent pourrait être "quant la propriété faim est suffisamment élevée, chercher quelque chose à manger", ou "toutes les n unités de temps, produire une nouvelle pomme et la faire tomber sur le sol".

3.3 Mémoire

Dans la mesure où ils "agissent", les agents animés doivent avoir une mémoire. Ceci est particulièrement crucial pour les agents qui modélisent des entités mobiles.

Il est essentiel qu'un tel agent ait sa propre représentation de l'environnement. Cela signifie qu'il dit mémoriser sa propre carte de l'environnement, c'est-à-dire la portion du monde qu'il connaît (car il l'a explorée ou parce qu'un autre agent lui a fourni certaines informations). Mais il doit également mémoriser des informations sur les autres agents : leurs positions et états la dernière fois qu'il a obtenu des informations à leur propos (parce qu'il les as vus ou a obtenu des informations d'un tiers). Au moment où il construit son plan, le moteur de l'agent appuie son raisonnement sur ces informations. Bien sûr, ces informations peuvent être corrompues, dans la mesure où les agents peuvent avoir bougé ou changé d'état : une porte fermée a

[1] Dans la suite, nous omettrons le terme *animé* lorsque le contexte suffira pour induire que c'est ce genre d'agent qui est concerné

pu être ouverte ou cassée, une pomme a pu être ramassée. En conséquence, il est nécessaire que l'agent soit capable de mettre à jour sa mémoire lors de l'obtention de nouvelles informations.

De plus, pour accroître le réalisme de la simulation, il est requis que les capacités de mémorisation soient limitées. Donc un agent peut oublier quelque chose. Mais cet oubli ne peut pas être réalisé par une simple gestion FIFO de liste mémoire. Comme dans la réalité, il est clair que les choses mémorisées n'ont pas la même importance pour les agents. Il apparaît que les choses importantes sont oubliées plus facilement. Il doit en être de même pour nos agents. Par exemple, même si notre agent a une capacité mémoire très limitée, il ne semble pas réaliste qu'alors qu'il cherche une clef pour ouvrir une porte donnée, il oublie où se trouve cette porte... En conséquence, une gestion raisonnable de la mémoire est mise en place pour les agents afin d'éviter de telles étrangetés.

Nous avons décrits comment les agents de notre monde artificiel sont définis à l'aide de propriétés et d'interactions. Les agents animés ont en plus des buts, une mémoire et des priorités. Mais nous devons maintenant expliquer comment ceux-ci peuvent agir (ou "vivre"), c'est-à-dire comment la simulation est possible. En conséquence, nous devons d'abord préciser la notion d'interaction, que nous avons déjà mentionnée, et ensuite indiquer comment le moteur de chaque agent l'utilise.

4 Connaissance et comportement : les interactions

Dans la section précédente, nous avons simplement décrit les constituants du monde, mais nous n'avons rien dit des actions qui peuvent être accomplies sur ceux-ci. Nous avons mentionné que les agents sont caractérisés par deux sorte d'interactions. Celles qu'il peut effectuer et celle qu'il peut subir. Nous allons préciser ici cette notion d'interaction.

Les interactions constituent une notion fondamentale. Elles influencent fortement la dynamique de la simulation et contiennent une grande partie de la représentation des connaissances sur le monde simulé (un peu comme les clauses de Horn en programmation logique).

4.1 Définition

Définition. Une *interaction* est une action entre deux agents : l'un qui effectue l'action, et est appelé *acteur*, et l'autre qui la subit, et est appelé *cible*.

Une interaction est possible, si et seulement si, elle est déclarée en tant que "peut effectuer" par l'acteur et en tant que "peut subir" par la cible. Ainsi, un agent `animé` particulier peut interagir sur une `porte` particulière pour l'ouvrir, si et seulement si l'interaction `ouvrir` est "effectuable" par l'agent `animé` et "subissable" par la `porte`. Bien sûr, dans une interaction, l'acteur et la cible peuvent désigner le même agent. Les interactions sont identifiées par leur nom.

Vous pouvez créer des interactions qui nécessitent que l'acteur se trouve à côté de la cible, et d'autres pour laquelle une certaine distance entre les deux agents est tolérée.

4.2 Écrire une interaction

La définition d'une interaction est déclarative : vous donnez la connaissance sur l'action et pas réellement comment la faire.

Le code d'une interaction a quelque chose d'une règle de système expert, avec éventuellement des paramètres, des variables et des attributs spécifiques (pour une interaction spécifique). Vous pouvez y trouver une partie condition et une autre conséquence, mais vous pouvez trouver également une boucle dans la partie conséquence.

Un langage spécifique a été créé pour permettre d'écrire ces interactions. Des fonctions primitives ont été définies pour certaines actions élémentaires (tel que tester si un agent possède un objet donné). Tout le reste (gérer la coopération entre agents, définir ce qu'est ouvrir une porte, etc.) peut être accompli grâce aux interactions. Vous pouvez "invoquer" une interaction sur la cible ou l'acteur (ou un paramètre, si c'est un agent) dans la description d'une interaction.

Vous pouvez avoir, et c'est souvent le cas, de nombreuses "règles" pour une interaction. Cela signifie simplement qu'il existe des alternatives pour l'accomplir.

4.3 Une manière de typer les agents

Dans la mesure où les agents sont essentiellement définis à partir des interactions, le fait qu'un agent puisse effectuer ou subir une interaction donnée est une caractérisation de cet agent.

Ainsi tous les agents qui peuvent subir l'interaction `manger` peuvent être considérés comme les agents "mangeables", ceux qui peuvent effectuer l'interaction `casser` peuvent être considérés comme les "casseurs", et ce même si ces deux notions (mangeable et casseur) n'existent pas explicitement.

Ceci peut être très utile lorsque vous écrivez des interactions. En effet, si vous devez créer une interaction `manger`, vous ne voulez probablement pas préciser que l'acteur doit manger cette pomme en particulier, ou même une pomme quelconque. N'importe quelle poire ferait l'affaire également, ou même tout objet qui peut être une cible pour l'interaction `manger` que vous pourriez imaginer et ajouter dans votre monde dans le futur. Une version générale et déclarative de l'interaction `manger` serait quelque chose comme `trouver un objet` *`mangeable`* `et le manger`. Cela peut être fait grâce aux interactions dans la mesure où le comportement de l'acteur sera de chercher un agent "qui peut subir `manger`". Il est clair que cette possibilité accroît la puissance d'expression des interactions ainsi que la capacité à les réutiliser.

Mais ceci peut être utile également pour obtenir un bon comportement des agents animés. Imaginons que vous ayez un agent qui doive, comme but, casser une porte (pour toute raison que vous pourrez imaginer), mais qu'il ne puisse pas effectuer l'interaction `casser`. Son nouveau (sous-)but sera de trouver un objet, oups... agent, qu'il pourrait obtenir et qui puisse effectuer l'interaction `casser` (une hache ou un marteau ou quoique ce soit d'autre...). Ainsi, par extension, il devient un agent capable de `casser` cette porte. Notre modèle considère qu'un agent peut exploiter les interactions effectuables des objets qu'il possède.

Les avantages sont assez évidents. D'abord, vous obtenez une puissance d'expression plus élevée pour les interactions. Ensuite, les interactions, et donc la connaissance, sont plus robustes : vous pouvez ajouter une nouvelle classe d'agent mangeable sans avoir à changer rien d'autre. Ainsi l'évolution et la réutilisabilité dans différents contextes sont renforcées.

Cette classification des agents est très utile puisque ce mécanisme contribue à donner la possibilité de placer une grande partie de la connaissance du monde dans les interactions et non dans le moteur lui-même ou dans toute base de connaissance supplémentaire.

4.4 Un exemple

L'exemple de la Figure 2 montre comment une interaction est écrite. Il y a une interaction effectuable et deux subissables associées. La deuxième subissable a un paramètre spécifique `?key`. La valeur de ce paramètre est spécifiée quand l'interaction a été attribuée à l'agent qui peut la subir.

```
  effectuable open  |                subissable open
IF                  |IF
     target.open()  |     target.isOpen == true
THEN                |THEN
     actor.success()|     target.succes()
                    |IF
                    |     target.isOpen == false
                    |     actor.own(?key)
                    |THEN
                    |     target.isOpen = true,
                    |     target.tellEnvironment(desactive, target)
```

Fig. 2. Un exemple : l'interaction `open`, effectuable et subissable

L'interaction effectuable dit que "*quand la cible est ouverte, alors vous avez un succ ès*". Ceci est écrit grâce à une invocation de l'interaction subissable de la cible comme condition. Donc, dans le détail, cette interaction sur "*si l'interaction* `open` *de la* `target` *est satisfaite alors l'interaction* `open` *de l'acteur est un succès*".

Il y a deux interactions subissables. La première dit "*si la cible est ouverte alors c'est bon*. La seconde signifie "*si la cible n'est pas ouverte, mais que l'acteur possède une clef particulière, alors la cible devient ouverte*"[2]

Ici on trouve un point important. Si l'acteur veut ouvrir la cible mais ne possède pas la clef spécifiée, il doit satisfaire `actor.own(?key)`, et l'interaction `own` devient alors un nouveau sous-but pour l'agent (ici `own` est une fonction primitive prédéfinie, puisqu'elle apparaît fréquemment, mais elle pourrait avoir été décrite comme une interaction). L'acteur va maintenant agir pour satisfaire ce nouveau but.

Nous voyons que l'interaction décrit déclarativement la connaissance sur "ouvrir quelque chose" et simultanément contient la dynamique nécessaire à cette ouverture. Nous préciserons la dynamique dans la section sur le moteur.

4.5 Illustration de la puissance des interactions

Présentons maintenant brièvement quelques autres possibilités offertes par les interactions.

Coopération Il est possible dans les interactions de spécifier si une interaction doit ou peut être accomplie par plusieurs agents. Par exemple, imaginez que pour ouvrir une porte, il soit nécessaire d'appuyer simultanément sur deux boutons distants, vous pouvez alors spécifier dans l'interaction que deux acteurs différents sont nécessaires pour les deux interactions "appuyer" (sans préciser ni qui doit être choisi ni comment répartir le travail, ceci est réalisé par le moteur de l'agent). Ce sera à l'acteur principal (celui qui initie l'interaction) de trouver un partenaire.

Dans un autre cas, pour ouvrir une porte vous avez besoin de deux clefs. L'agent qui doit ouvrir cette porte peut, soit trouver les deux clefs tout seul soit demander à un autre agent de trouver l'une de ces clefs pour lui (voire même les deux). C'est son choix et cela peut être précisé (facilement et naturellement) dans l'interaction.

En conséquence les interactions permettent également de mettre en place une forme de coopération entre agents.

Dynamicité Avec les interactions, il est très facile d'ajouter ou de retirer des capacités à un agent. Imaginez que la serrure d'une porte ait été endommagée, vous pouvez alors empêcher que quelqu'un la décadenasse simplement en retirant l'interaction correspondante de cette instance de porte.

Ainsi vous pouvez gérer l'évolution dynamique des agents. Un agent peut apprendre d'un autre une des ses interactions. Cela peut être accompli par enseignement explicite, mais peut également être obtenu par observation : un agent en voit un autre casser une porte et ainsi apprend comment casser.

Interactions de contrôle Il est également possible d'avoir des interactions qui permettent de changer le comportement du moteur de l'agent lui-même. Par exemple, la gestion de la mémoire peut être décrite par des interactions, ou encore la stratégie utilisée pour explorer les parties inconnues du monde. Cela peut être utile pour individualiser le comportement des agents.

Vous pouvez même imaginer une interaction qui contrôle la gestion des interactions, une espèce de méta-interaction, par exemple pour gérer l'apprentissage de nouvelles interactions comme cela a été mentionné précédemment.

En conclusion, les interactions sont le point central de notre travail. Elles contiennent la représentation des connaissances sur le monde simulé. Le pouvoir d'expression est suffisamment élevé pour constituer un cadre facile pour modéliser le monde et sa dynamique.

5 Planification : le Moteur

Chaque agent animé possède son propre moteur de raisonnement. Par défaut le moteur est le même pour tous les agents de notre simulation, mais nous avons vu précédemment que l'utilisation d'interactions de contrôle permet de les individualiser.

Le moteur utilise la connaissance contenue dans les interactions pour construire des plans pour résoudre les différents buts de l'agent en concurrence. La planification n'est pas un problème nouveau et de nombreuses approches ont déjà été utilisées, y compris dans des applications multi-agents [2, 3].

[2] Nous avons ignoré la dernière ligne de la seconde interaction. Elle signifie "*et l'environnement est informé que l'état a changé*", ceci peut être utile si vous souhaitez avoir un effet de bord tel qu'une animation qui montre la cible en train d'être ouverte, même si il aurait été possible d'utiliser un principe d'événement/listener pour gérer cela.

Notre moteur de planification est bien évidemment basé sur les interactions et nous allons le présenter brièvement ici. De nombreux facteurs influencent la sélection du but courant. Pour deux agents différents avec les mêmes buts, le choix du prochain but courant peut être différent en fonction de leurs connaissances et caractéristiques.

5.1 Capacités du moteur

Le rôle du moteur est de construire des plans pour résoudre les buts de l'agent. Dans notre système **un but est une interaction** que l'agent doit accomplir. Un agent peut avoir plusieurs buts concurrents. La cible de ce but peut ou non être connue. Si ce n'est pas le cas, alors le moteur doit trouver une cible adaptée pour accomplir ce but.

Même si le moteur fait un bon choix pour le prochain but courant, cela n'est pas suffisant pour avoir un comportement rationnel. Le moteur doit prendre en compte d'autres considérations. Imaginons par exemple qu'un agent ait deux buts : l'un est de manger et l'autre est de décadenasser une porte avec une certaine clef rouge. Supposons que le but `manger` ait une priorité plus élevée. Il est donc le but en cours de traitement par l'agent (ie. le moteur). Alors qu'il cherche après un objet mangeable, l'agent passe au voisinage de la clef rouge. Un observateur trouverait stupide qu'il ne ramasse pas cette clef dans la mesure où elle sert au second but, et ce même si elle n'est pas directement utile pour son but courant. Le moteur de notre agent est opportuniste (comme cela peut être raisonnablement exigée) et détecte que la clef peut être utile dans le futur (c'est-à-dire pour résoudre un autre but). L'agent ramassera donc la clef.

La mémoire est également gérée par l'agent, y compris les mises à jour et les oublis. Il en est de même pour les déplacement de l'agent, le moteur utilise un algorithme classique du type A$*$ pour optimiser ses déplacements. Une stratégie d'exploration pour découvrir les parties inconnues de l'environnement ou pour chercher un agent requis est également fournie.

5.2 Comment le moteur fonctionne

Le moteur doit faire des choix et des plans sur les nombreux buts concurrents de l'agent. Il est essentiellement basé sur une recherche classique du type chaînage arrière. Nous allons présenter ici les grandes lignes de son fonctionnement. Il est possible d'en dégager quatre grandes étapes :

- mise à jour de la mémoire
- remise en question des buts
- détection du prochain but courant
- résolution d'une des actions du but courant

Mise à jour de la mémoire De nouveaux agents peuvent être apparu ou avoir changé d'état dans le champ de perception de l'agent. Les informations relatives doivent donc être mises à jour.

Remise en question des buts En accord avec les nouvelles informations contenues dans la mémoire, certains buts peuvent devenir sans fondement. C'est le cas par exemple quand une porte précédemment cadenassée a été ouverte, ou quand un objet a disparu, etc.

Sélection du prochain but courant Le principe du moteur est similaire à un chaînage arrière. Il gère un arbre de recherche avec les différents buts concurrents ainsi que la progression dans la résolution de ces buts.

Le chaînage est réalisé sur les interactions, et plus particulièrement sur les conditions des interactions. Par exemple dans l'interaction `open` décrite à la figure 2, la deuxième interaction subissable a une condition `actor.own(?key)`. Celle-ci signifie que l'interaction `own` doit donner un `success()`, pour que la condition soit satisfaite. Donc cette interaction `own` va devenir le prochain (sous-)but courant pour permettre la résolution de but principal `open`.

Les priorités des différents buts (et sous-buts) sont évaluées et le but avec la plus forte priorité est sélectionné.

Résolution d'une action du but Dans certains cas la résolution d'une action peut nécessiter un déplacement de l'agent (pour aller chercher un objet par exemple), ensuite si la condition de l'interaction courante est satisfaite, alors la conséquence est appliquée.

Mais un but peut également échouer. Quand un but a échoué, deux stratégies peuvent être choisies. Soit le but est supprimé et définitivement oublié, soit une pénalité est appliquée à ce but. Ainsi, cette pénalité étant prise en compte pour le calcul de la priorité du but, ce but sera ignoré pendant un certain temps. Dans ce cas, le but pourra à nouveau être pris en compte dans différents cas de figures :

– tous les buts plus prioritaires on été résolus
– l'agent reçoit (ou perçoit) des informations qui entraînent une augmentation suffisante de la priorité du but,
– la pénalité est réduite peu à peu avec le temps. Ainsi après un certain temps, cette pénalité est annulée et la priorité du but peut devenir à nouveau suffisamment élevée (cela simule le fait que vous attendrez probablement un certain temps avant de réessayer de rencontrer un collègue dans son bureau si il n'y était pas la dernière fois que vous avez essayé de l'y voir. Vous ne réessayez pas toutes les deux minutes).

Enfin, nous avons mentionné que certains buts pouvaient être récurrents. Cela signifie qu'une fois qu'ils ont abouti à un succès ou un échec, ces buts ne vont pas disparaître mais rester dans la "liste des buts à résoudre". Nous donnerons par la suite un exemple d'exploitation de cette récurrence.

5.3 Influences sur le choix du but courant

Rappelons que pour un agent donné, de nombreux buts sont en compétition. En conséquence il est nécessaire de faire un choix parmi les possibilités qu'ils offrent. Une fonction d'évaluation est utilisée pour déterminer l'importance de chacun de ces buts. Cette évaluation est influencée par différents facteurs.

Priorités des agents pour les buts Lorsque vous créez un agent animé, des priorités sont attribuées pour ses buts. Cette priorité influence l'importance du but et de tous les sous-buts nécessaires pour le résoudre.

Propriétés des agents et interactions Les agents ont des priorités qui peuvent être associées aux interactions. Quand vous définissez une interaction, vous pouvez dire que telle priorité favorise ou pénalise un but correspondant à cette interaction. Par exemple, si pour ouvrir une porte cadenassée vous disposez de deux (sous-)interactions possibles, qui sont soit la `décadenasser` (ie. trouver la bonne clef et l'utiliser), soit la `casser`. L'interaction `décadenasser` est positivement influencée par la propriété `discret`, et l'interaction `casser` par la propriété `violent`. Si un agent possède la priorité `violent`, plus cette priorité est haute, plus il y aura de chances pour qu'il choisisse l'interaction `casser` au détriment de l'interaction `décadenasser`. Ainsi deux agents avec les mêmes buts peuvent avoir des comportements différents.

L'utilisation de ces propriétés permet une gestion facile de certains comportements utiles aux simulations. C'est le cas par exemple de tous les comportements récurrents qui se produisent sous certaines conditions. Par exemple, supposons que l'on souhaite qu'un agent puisse être sensible à la faim. Vous pouvez disposer d'une propriété qui représente le degré de faim de l'agent, et cette propriété est régulièrement augmentée avec le temps. L'interaction `manger` est, logiquement, influencée positivement par cette propriété et l'agent peut effectuer cette interaction. Ainsi, de manière naturelle, l'importance du but `manger` va augmenter régulièrement. Après un certain temps elle deviendra le but le plus important au fur et à mesure que l'agent devient affamé. Alors, le moteur va automatiquement proposer l'interaction `manger` comme nouveau but courant et notre agent va alors chercher après un agent mangeable. Lorsqu'il en aura trouvé et mangé un, sa faim va décroître et avec elle l'importance du but (de l'interaction) `manger`. Ce but est donc mis de côté jusqu'à ce que l'agent se sente de nouveau affamé, et les autres buts peuvent à nouveau être sélectionnés.

Évaluation des coûts Dans le moteur, une évaluation des sous-buts qui doivent être accomplis pour résoudre le but est effectuée pour faciliter la différenciation entre les buts concurrents.

Comme exemple, une évaluation du temps, basée sur l'éloignement, peut être prise en compte. Imaginons l'agent qui peut `décadenasser` ou `casser` la porte devant laquelle il se trouve. Il sait où se trouvent la hache et la clef. La clef est proche de la porte mais la hache est plus loin, alors le moteur sera plus enclin à choisir l'interaction `décadenasser`.

Cette fonction d'évaluation du coût peut facilement être particularisée et adaptée en fonction de la simulation.

6 Un framework ouvert

Un prototype a été développé pour valider le moteur et le principe des interactions. Il fournit le moyen de créer les classes d'agents, les interactions, et de produire un environnement et des instances d'agents (en leur attribuant des valeurs de propriétés, des buts, etc.).

Dans cette phase de description, une interface graphique permet de définir facilement les classes d'agents, les interactions, les priorités et l'environnement (voir figures 3 et 4. La structure des données de type orienté objet facilite la réutilisabilité des éléments d'une simulation à une autre (une fois que vous avez dit comment ouvrir quelque chose, cela sera probablement réutilisable dans différentes simulations et pour différentes choses ouvrables).

Fig. 3. La fenêtre de création d'interactions

Une interface permet la construction facile d'un environnement 2D ainsi que le placement des agents définis. La simulation peut ensuite être exécutée et évaluée.

Les simulations testées montrent que le moteur des agents présente tous les comportements rationnels que nous avons précédemment présentés et cités comme nécessaires. Elles montrent que notre travail n'est évidemment pas limité aux simulations pour des jeux. En effet, les interactions constituent un outil suffisamment expressif pour exprimer une large variété de problèmes de simulation. L'utilisation de buts récurrents permet par exemple d'avoir des phénomènes de type "cycle de vie".

De plus la dynamicité fournie par la possibilité d'ajouter de nouvelles interactions à un agent ouvre la voie à des scénarii de simulations complexes et variés.

7 Conclusion

Nous avons présenté un cadre et une application pour la simulation de mondes artificiels. Son aspect novateur est contenu dans la notion d'interaction. Les agents sont décrits à partir des interactions qu'ils peuvent effectuer et subir. Parmi les agents, certains sont particuliers et sont appelés "agents animés". ces agents ont des buts à accomplir. Les interactions constituent à la fois la représentation des connaissances du monde et la base de la dynamique du moteur de raisonnement des agents animés. Ce moteur se comporte de manière rationnelle et gère plusieurs buts en concurrence en sélectionnant le but courant en fonction de critère d'évaluation et en étant capable de remettre en cause ses choix ultérieurs.

Même si la motivation première de ce travail était une application aux jeux vidéo, notre notion d'interactions offre un contexte ouvert à d'autres types de simulations.

Les principes présentés ici ont été testés dans un prototype qui valide la représentation des connaissances via les interactions et le moteur des agents basés sur ces mêmes interactions.

Fig. 4. La fenêtre de création d'interactions

L'intégration de ce travail est actuellement en cours dans la société Cryo Interactive Entertainment.

Nous remercions Wilfried Hinault et Emmanuel Bernard, qui ont fourni un gros travail pour produire un prototype très réussi permettant de tester les simulations basées sur cette idée d'interactions décrite dans cet article.

References

1. AICenter. Excalibur: Adaptive constraint-based agents in artificial environments. `http://www.ai-center.com/projects/excalibur`.
2. P. Maes. How to do the right thing. Technical Report AI memo 1180, MIT, 1989.
3. A. Nareyek. Specification and development of reactive systems. In *1998 AIPS Workshop on Integrating Planning, Scheduling and Execution in Dynamic and UncertainInformation Processing 86*, pages 7–14, Menlo Park California, 1998. AAAI Press.
4. M. Travers. *Programming with Agents: New metaphors for thinking about computation*. PhD thesis, MIT, 1996.