Protocole de test pour agents développementaux et intrinsèquement motivés par une approche interactionniste

Olivier Voisin

université de bretagne occidentale

institut de recherche en informatique et systèmes aléatoires

Rapport de stage

Master 2ème année de recherche en informatique

Protocole de test pour agents développementaux et intrinsèquement motivés par une approche interactionniste

Olivier Voisin (MRI parcours SICH)

Août 2011

Laboratoire : *LIRIS (Laboratoire d'InfoRmatique en Image et Systèmes d'information)*
Equipe : *SILEX (Supporting Interaction and Learning through Experience)*
Projet : *IDEAL (Implementation of DEvelopmentAl Learning, http://liris.cnrs.fr/ideal/)*
Encadreurs : *Olivier GEORGEON et Alain MILLE*

LIRIS

Résumé

Dans ce rapport, nous remarquons que l'évaluation des agents développementaux et intrinsèquement motivés est un réel problème. Les environnements de simulation actuels, étant conçu pour des agents dont le but est de résoudre des problèmes, ne sont pas adaptés. L'enjeu est alors de proposer un modèle et une implémentation d'un nouvel environnement de simulation répondant à des critères spécifiques (interactivité, immersivité, en ligne, multi-agent, ergonomie, évolutivité et interactionniste). Ces critères résument la volonté de produire un nouveau protocole de test destiné aux concepteurs d'agent développemental et intrinsèquement motivé, qui reste simple d'utilisation et adaptable aux différents cas d'utilisation.

Nous proposons un modèle qui tient compte de deux niveaux d'utilisateur afin de séparer les concepteurs d'agent, qui préparent les simulations, et les observateurs, qui jouent les simulations. Le partage des résultats évolue des valeurs statistiques traditionnelles vers la possibilité de rejouer les simulations, de modifier leur cours afin d'étudier l'apprentissage des agents, au même titre que les psychologues étudient l'apprentissage chez l'être humain. Nous centrons ensuite notre modèle sur les interactions. D'un côté, les interactions agent-environnement permettent de ne plus faire de distinction entre les processus d'action et de perception. La perception se fait au travers de l'action. D'un autre côté, les interactions observateur-environnement permettent de faire participer l'humain aux simulations, sans qu'il n'agisse directement sur l'agent, comme pour les apprentissages à base de récompenses. Enfin, nous précisons l'importance des traces qui sont des résultats de simulation permettant d'étudier a posteriori les comportements et l'apprentissage des comportements des agents.

La présentation de SECA, implémentation du modèle mis en place, vient ensuite. Nous décrivons son architecture, l'interactivité du point de vue de l'observateur qui devient acteur à part entière dans les simulations et le tableau de bord qui offre une représentation des stimuli reçus par l'agent et de son état interne.

Table des matières

Table des figures

1 Introduction

1.1 Contexte : IA développementale et intrinsèquement motivée

Pendant de longues années, l'IA s'est focalisé sur le développement d'agents[1] capables de résoudre des problèmes spécifiés par des humains. Les agents étaient donc créés dans un contexte précis (le contexte du problème) pour un but précis (résoudre le problème). En IA développementale, l'intérêt est porté sur la reproduction de l'apprentissage observé chez l'enfant. Un tel apprentissage permet non plus de se limiter à la résolution d'un seul problème mais de résoudre toutes sortes de problèmes.

L'apprentissage par renforcement conduit à des agents extrinsèquement motivés. Un agent *extrinsèquement motivé* est un agent qui a un but précis et qui est récompensé lorsqu'il atteint ce but. Ce type d'agent cherche un comportement optimal au travers d'expériences successives. En contraste, un agent *intrinsèquement motivé* est un agent qui n'a pas de but défini par avance et par conséquent, pas de récompense venant de l'extérieur non plus. Il est plus autonome dans la recherche de nouveaux comportements.

Approche interactionniste vs symbolique

En intelligence artificielle traditionnelle, quand un agent perçoit un environnement, celui-ci passe des symboles à l'agent. Cette approche est dite symbolique. Elle demande donc au développeur d'implémenter une sémantique dans l'agent. Le sens donné à ces symboles vient donc du développeur et non de l'agent.

Une deuxième approche se distingue de celle-ci : l'approche interactionniste. L'environnement passe non plus des symboles mais des stimuli à l'agent. En effet, « le sujet ne connait pas de "choses en soi" (hypothèse ontologique) mais il connait l'acte par lequel il perçoit l'interaction entre les choses » (Le Moigne, 1995). L'agent apprend alors des régularités d'interaction. Cette approche est centrée sur l'interaction ce qui conduit à une nouvelle problématique. La frontière entre un agent et un environnement simulé est floue, sa position exacte est donc un choix arbitraire de l'utilisateur. En définissant cette frontière, nous définissons deux flux qui traversent cette frontière. Le flux allant de l'environnement vers l'agent est appelé flux de stimuli. Le flux allant de l'agent vers l'environnement est appelé flux d'actions.

En définissant la position de cette frontière, l'utilisateur définit la nature de ces stimuli et de ces actions. Ainsi, les stimuli et les actions peuvent être plus ou moins abstraits. Un fort niveau d'abstraction réduit la quantité de données transmises et le processus d'interprétation qui est en aval. Un faible niveau d'abstraction permet de simuler des interactions plus proches de la réalité, riche en informations qui n'ont pas toutes un intérêt suivant l'interaction considérée. Il faut donc un processus d'interprétation plus complexe.

[1] Généralement virtuels mais également incarnés au travers de robots.

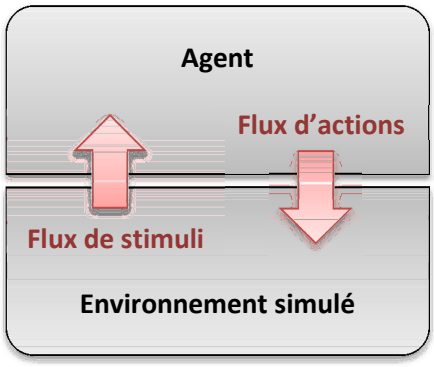

Figure 1.1 : Frontière entre un agent et un environnement simulé

Projet IDEAL

Le projet IDEAL (Implementation of DEvelopmentAl Learning) consiste à implémenter un mécanisme d'apprentissage développemental des premiers stades de l'évolution individuelle (Piaget, 1937) dans un agent artificiel situé dans un environnement simulé. Pour cela, un mécanisme permet à l'agent de rentrer dans un processus de type « bottom-up » pour organiser hiérarchiquement ses schèmes de comportements, au fur et à mesure que l'agent interagit avec l'environnement simulé.

Ce travail cherche à étayer deux hypothèses sur la cognition : l'hypothèse émergentiste et l'hypothèse constructiviste. D'après ces hypothèses, un observateur peut attribuer des phénomènes cognitifs[2] à un agent en observant son activité, à condition que le comportement de l'agent s'auto-organise de manière appropriée. Ces hypothèses sont liées aux épistémologies constructivistes (Le Moigne, 1995), et aux théories de la cognition située (Suchman, 1987) et incarnée (Wilson, 2002).

Ce projet répond au besoin d'une implémentation information des hypothèses émergentistes et constructivistes afin de les valider ou de voir quelles sont leur limites. Il apporte également un nouveau point de vue à l'approche développementale en intelligence artificielle. Si les hypothèses sont confirmées, des démonstrations d'agents capables de se développer dans un environnement simulé seront mises en ligne. Ces démonstrations agrémenteront le débat éthique à propos du statut des futurs agents intrinsèquement motivés, qui feront transparaître une sensibilité[3] et un comportement de plus en plus élaboré.

1.2 Problématique

IDEAL n'est pas le seul projet à développer une IA développementale et intrinsèquement motivée. Un problème commun se pose alors : Comment valider ces IA à partir des protocoles de test mis en place pour tester les IA sensées résoudre des problèmes. En effet, les IA développementale et intrinsèquement motivée n'ont pas de problème à résoudre, de but à atteindre. Les protocoles de test actuels ne sont donc pas adaptés à ces nouvelles IA (Oudeyer, Kaplan, & Hafner, 2007). Nous proposons alors de modéliser et d'implémenter un protocole de test pour les agents

[2] Tels que la connaissance, l'émotion, l'intention ou l'anticipation.
[3] En philosophie, la sensibilité désigne la faculté de percevoir par les sens.

développementaux. Ce protocole de test repose sur un environnement de simulation, spécialement conçu pour les apprentissages comportementaux.

Un tel environnement demande d'être en mesure de :

- Définir la frontière entre l'agent et l'environnement
- Donner à l'utilisateur une marge de manœuvre sur cette frontière
- Permettre à l'utilisateur d'étudier et d'analyser le comportement des agents

2 Etat de l'art sur les paradigmes expérimentaux

En intelligence artificielle, il existe seulement deux méthodes pour tester ses agents autonomes : au travers d'une plateforme robotique ou grâce à une simulation numérique. Les plateformes robotiques sont coûteuses et tributaires des contraintes de la réalité (bruit des mesures, disfonctionnement électriques ou mécaniques, …). Elles sont donc souvent remplacées par des environnements de simulation, qui reproduisent la réalité tout en la simplifiant.

Par conséquent, de nombreux environnements de simulation sont disponibles dans la littérature et nous distinguerons les environnements « génériques », destinés à tester différents agents, des environnements « ad hoc », développés en parallèle des agents testés. Nous nous appuieront sur les critères suivants pour comparer les environnements : l'*interactivité* (avec un utilisateur), l'*immersivité*, la présence des simulations *en ligne* (sur Internet), l'orientation *multi-agent*, l'*ergonomie* (pour un utilisateur), l'*évolutivité* (pour un développeur), l'hypothèse *interactionniste* et l'*analyse des comportements*.

2.1 Environnements génériques pour agents apprenants

2.1.1 oRis : Langage de simulation

oRis (Harrouet, 2000) est un environnement de simulation interactive. C'est à la fois un langage de programmation interprété et un environnement d'exécution (Figure 2.1). Ce langage permet de programmer des simulations et d'intervenir dans le déroulement d'une simulation en modifiant le code à tout instant, sans avoir à recompiler l'ensemble. L'environnement d'exécution dispose d'une interface qui permet de visualiser une simulation et d'interagir avec le code exécuté.

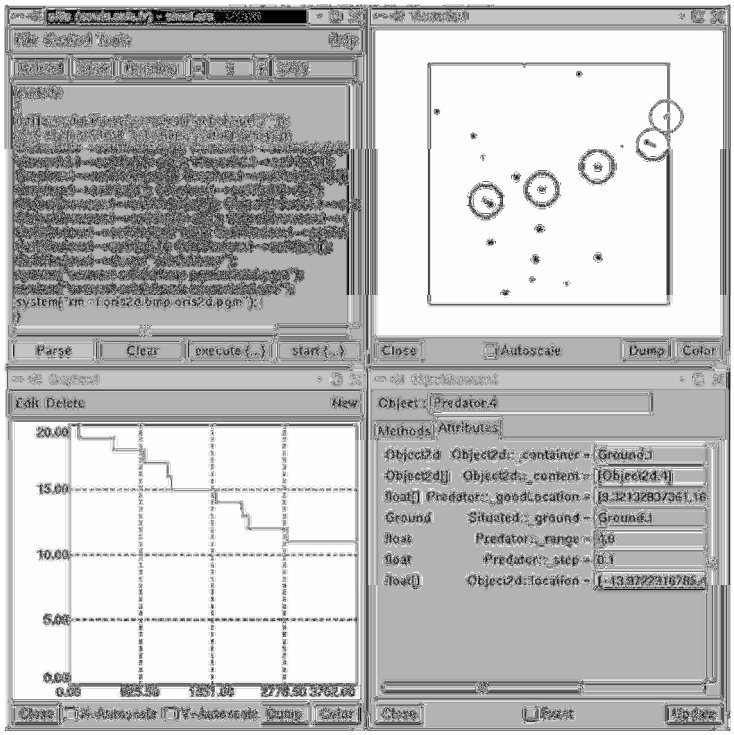

Figure 2.1 : Interface d'oRis

En tant que langage de programmation, oRis est par définition très évolutif. Il est également interactif et orienté multi-agent. En revanche, les simulations ne sont pas rendues disponibles en ligne, l'ergonomie est masquée par les possibilités offertes et l'immersion dans la simulation est difficile. De plus, oRis offre une approche symbolique (et non pas interactionniste) et ne permet pas d'analyser le comportement d'un agent.

2.1.2 RoboCode/dTank : Jeu de simulation

RoboCode (Li, 2002) est un simulateur de combat de robots, représentants des tanks, dans un monde en deux dimensions développé par IBM Alphaworks. Les robots doivent se déplacer en évitant les murs et en évitant de se faire tirer dessus par d'autres robots. Pour remporter une manche, un robot doit localiser ses adversaires et leur tirer dessus. Ce jeu se joue hors-ligne et une interface graphique est disponible pour visualiser les combats (Figure 2.2). Les comportements des robots sont programmés en Java par les joueurs et ce jeu attire tellement de joueurs que près de 10.000 robots sont disponibles en ligne (http://robocoderepository.com). Avec une telle diversité de concurrents, ce jeu attire aussi les développeurs d'agents autonomes. Ainsi, (Eisenstein, 2003) utilise un algorithme génétique pour faire évoluer un contrôleur de robot et battre ses concurrents.

Figure 2.2 : Interface de RoboCode

RoboCode offre un minimum de réalisme avec des effecteurs qui ne sont pas instantanées, des balles qui mettent du temps à atteindre leur cible, un radar unidirectionnel et un système de collision. Etant un jeu vidéo, il a également un rendu sonore en plus du rendu vidéo, ce qui le rend plus immersif que les simulations privées de sons. Bien que les combats ne se déroulent pas en ligne, le jeu et les robots sont téléchargeables gratuitement. C'est un environnement de simulation multi-agent non interactif puisque lors des combats, il est impossible d'interagir avec la scène pour modifier le décor par exemple. De la même manière, l'évolutivité est restreinte car nous ne pouvons pas modifier le code source du jeu simplement, ce qui paraît logique pour un jeu vidéo mais qui freine le test d'agents plus complexes ou qui sortent du contexte du jeu. La perception des autres robots se fait au travers d'un évènement qui renvoie toutes les informations disponibles sur le robot perçu, donc l'approche est symbolique. L'analyse de comportements n'est pas possible non plus dans ce type de simulation car le but du jeu est simple et seul le résultat compte : le vainqueur est le dernier robot survivant.

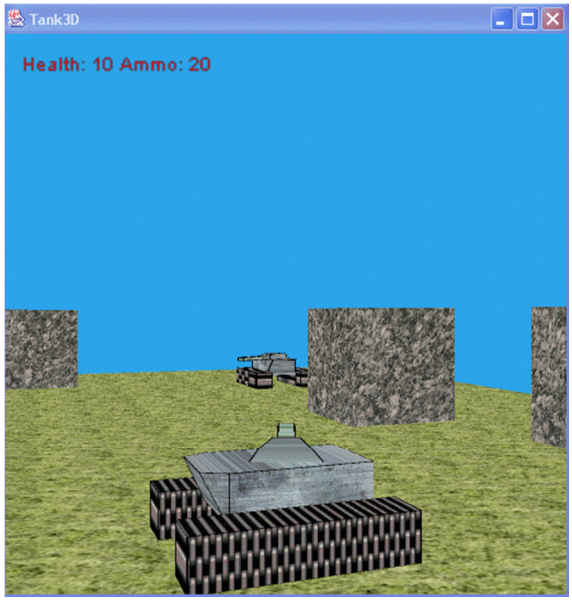

Figure 2.3 : Interface 3D de dTank

dTank (Ritter, Kase, & Bhandarkar, 2007) est un environnement de simulation très similaire à RoboCode. Utilisé principalement dans l'enseignement, il été développé spécialement pour tester des modèles cognitifs d'agents autonomes. Bien que dTank offre le choix entre une interface en deux dimensions et une interface en trois dimensions (Figure 2.3), le monde simulé est toujours en deux dimensions.

2.1.3 Vacuum Cleaner Environment : Un problème de ménage

Le Vacuum Cleaner Environment est un problème qui fait partie des « grands classiques » en IA (Russel & Norvig, 1995). Il a récemment été mis au gout du jour par une implémentation java pour l'enseignement par (Cohen, Ritter, & Haynes, 2005). C'est un environnement de simulation, qui simule des mondes simples sous forme de grilles à deux dimensions (Figure 2.4). Il permet à un agent, représenté par un aspirateur, de se déplacer pour chercher la poussière à aspirer.

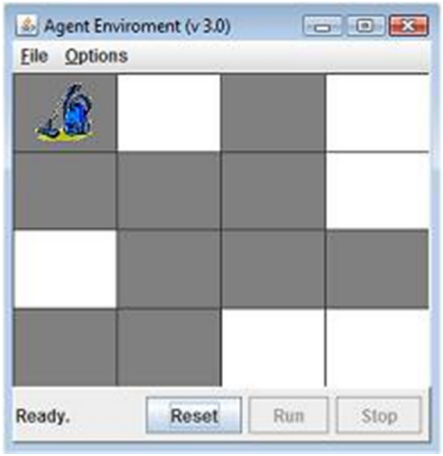

Figure 2.4 : Interface de Vacuum Cleaner Environment

Cet environnement est très simple : il n'est pas en ligne, ni multi-agent mais il est facile à prendre en main et évolutif, du moment qu'on sait coder en Java. De plus, des sons sont joués lors des actions de l'agent et il est possible de d'interagir avec l'environnement pour remettre de la poussière lors du déroulement d'une simulation. Les cellules de la grille sont perçues comme des objets donc l'approche est une fois de plus symbolique. L'analyse de comportements n'est pas possible non plus avec cet environnement.

Grâce à sa grande évolutivité, c'est cet environnement qui a été choisi jusqu'à maintenant pour tester l'agent développé dans le cadre du projet IDEAL. Pour obtenir une approche interactionniste de la perception et rendre possible l'analyse de comportements, l'environnement a été revu et modifié en profondeur. Cependant, il reste limitant de par sa nature discrète et sa structure en deux dimensions.

2.2 Environnements ad hoc en apprentissage constructiviste

Parmi les environnements généralistes, nous avons vu qu'aucun n'adoptait une approche interactionniste pour la perception des agents. Dans le cadre des agents constructivistes, cette approche est essentielle car une approche symbolique suppose que l'agent connaisse a priori l'objet qu'il perçoit. La complexité de l'agent augmente alors avec le nombre d'objets disponibles dans

l'environnement simulé. Pour s'affranchir de cet a priori, les modeleurs d'agents constructivistes utilisent une perception par stimuli, qui est plus réaliste. En conséquent, ils sont souvent obligés de développer leur propre environnement de simulation pour tester leurs agents.

2.2.1 Gridland : Représentation par la géométrie

Gridland (Cañamero, 1997) est un environnement en deux dimensions qui est peuplé de figures géométriques (Figure 2.5). Ces figures se distinguent selon trois catégories principales :

- Les êtres vivants ;
- Les sources d'eau et les sources de nourriture ;
- Les blocs inanimés de différentes tailles et différentes formes.

De plus, il existe deux espèces au sein des êtres vivants : les *Abbotts* et les *Ennemies*. Les Ennemies ont un comportement simple et prédéfini qui est de se déplacer sur la grille, de boire l'eau et de manger indifféremment la nourriture, les Abbotts ou d'autres Ennemies. Les Abbotts sont des créatures autonomes qui ont des capacités d'apprentissage et de résolution de problèmes. Une créature est composée de plusieurs agents spécialisés pour une tâche particulière. Il existe 7 types d'agents différents : les capteurs (*sensors*), les agents de reconnaissance (*recognizers*), les agents directionnels (*direction-nemes*), les cartes (*maps*), les effecteurs (*effectors*), les agents comportementaux (*behaviors*) et les manageurs (*managers*). Pour obtenir des comportements de plus en plus complexes, seuls de nouveaux agents sont ajoutés à la créature, sans modifier les agents déjà présents. C'est le principe sous-jacent à une architecture à subsomptions (Brooks, 1991).

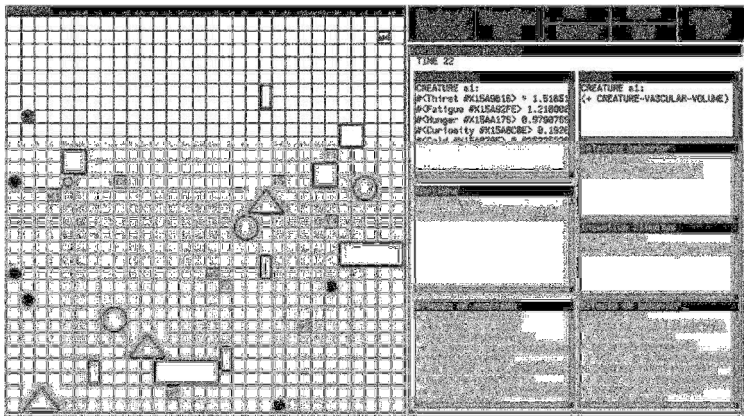

Figure 2.5 : Interface de Gridland

Gridland est un environnement multi-agent et relativement ergonomique si l'on tient compte de son âge. En revanche, lorsqu'une simulation est lancée, l'utilisateur ne peut pas interagir avec l'environnement simulé. Les formes très géométrique et l'absence de sons rend l'immersion difficile. L'environnement n'est donc pas interactif, ni immersif, ni évolutif. L'analyse de comportements n'est pas possible non plus.

Les auteurs de Gridland revendiquent le fait d'avoir des agents intrinsèquement motivés ». La motivation intrinsèque est ici implémentée sous formes d'émotions explicites. Ce sont des variables étiquetées « émotion ». Cet environnement donne donc un protocole expérimental où il n'y a pas d'objectif final à atteindre prédéfini, contrairement aux environnements vus dans la section précédente. En revanche, les agents de Gridland ne sont si développementaux, ni interactionnistes.

En effet, il existe des agents capteurs pour les perceptions et des agents effecteurs pour les actions. Pour une approche interactionniste, il aurait fallu avoir à la place des agents pour les interactions.

2.2.2 Perotto & Alvarez : Apprentissage par induction

Perotto et Alvarez travaillent sur une architecture d'agent apprenant (Perotto & Alvarez, 2006) inspiré par l'approche constructiviste. Cette architecture est à base de schèmes, un schème représentant une unité élémentaire de l'activité intellectuelle. Dans leur implémentation d'un schème, ils le construisent comme la composition d'une perception (appelée *contexte*), d'une action et d'une deuxième perception (appelée *attente*). Cette implémentation est similaire à celle utilisé par Drescher, dont le schème était composé d'un contexte, d'une action et d'un résultat (Drescher, 1991).Le mécanisme d'apprentissage par induction de Perotto et Alvarez permettent ensuite à leur agent de découvrir les régularités de l'environnement en appliquant trois méthodes sur les schèmes : la différentiation, l'ajustement et l'intégration. L'agent est incarné ce qui lui permet d'ajouter des propriétés au corps : la douleur, la fatigue, l'épuisement et le plaisir.

Pour tester leur agent, Perotto et Alvarez ont implémenté un environnement de simulation sous la forme d'une grille à deux dimensions dans laquelle l'agent est sensé apprendre à se déplacer en évitant les collisions et l'épuisement, et en recherchant du plaisir. Les cellules de l'environnement peuvent être des espaces vides ou des obstacles. A chaque tour, l'agent peut faire une des deux actions possibles : avancer d'une cellule ou tourner à 90 degrés (aléatoirement à droite ou à gauche). L'agent peut avancer librement sur les espaces vides mais se cogne contre les obstacles. Il perçoit uniquement la cellule qui est devant lui et les propriétés de son corps (douleur, fatigue, épuisement et plaisir). La douleur s'active quand l'agent se cogne, la fatigue quand il marche longtemps, l'épuisement quand il marche longtemps en état de fatigue et le plaisir quand il marche. La Figure 2.6 représente l'environnement où les cellules noires sont les obstacles, les cellules blanches sont les endroits vides et le cercle représente l'agent.

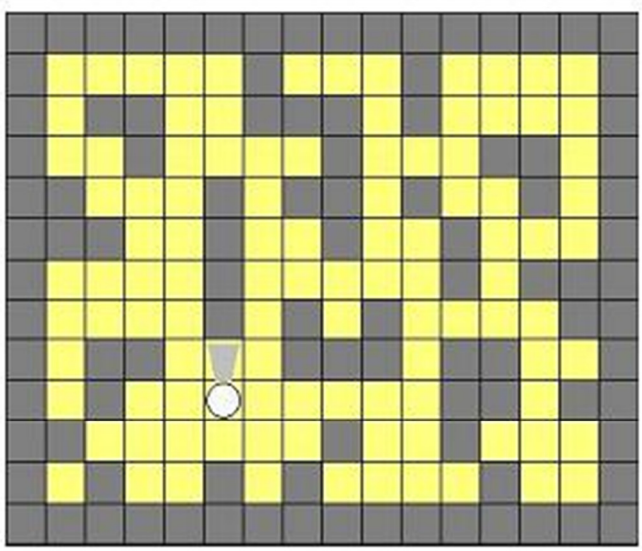

Figure 2.6 : Interface de l'environnement de Perotto et Alvarez

Cet environnement ne permet pas d'interagir avec le monde simulé lors de la simulation : il n'est pas interactif. Son rendu graphique seul fait qu'il n'est pas très immersif non plus. Les

simulations sont effectuées par l'équipe seule et ne sont pas disponibles en ligne, tout comme le code source de l'environnement, ce qui bride son évolutivité. Son interface sobre le rend ergonomique et il adopte une approche interactionniste. En revanche, il n'est pas multi-agent et ne permet pas d'analyser le comportement des agents.

2.2.3 Guerin & McKenzie : Un bébé au bras articulé

La théorie de Piaget a inspiré beaucoup de travaux en intelligence artificiel. Guerin et McKenzie soulignent le fait que beaucoup de ces travaux ne tiennent pas compte des étapes de développement décrites par Piaget. Ainsi, ils affirment qu'avec des simulations dans des environnements où l'agent se déplace en cherchant à éviter des murs ou des obstacles, les résultats ne peuvent pas être comparés aux développements infantiles (Guerin & McKenzie, 2008). Pour leur modèle d'agent, ils reprennent le principe d'un schème composé d'une perception (état initial du monde perçu avant l'exécution du schème), d'une action et d'une autre perception (prédiction du monde perçu après l'exécution du schème). Ils ajoutent également un quatrième terme qui représente une liste de couples de valeurs associant une valeur à chacun des schèmes existants. Cette valeur représente l'utilité d'un schème dans l'accomplissement d'un autre schème (Par exemple, déplacer un bras avant d'attraper un objet).

Le bébé simulé vit dans un monde continu[4] à 2 dimensions avec des blocs carrés (Figure 2.7). Un moteur physique simule les collisions et les frictions entre les blocs. Le bébé a un seul bras (au sens courant : membre supérieur) composé de deux blocs rectangulaires et d'un bloc carré : un bras (au sens anatomique : segment composant le membre supérieur), un avant-bras et une main. Le bras et l'avant-bras peuvent tourner respectivement autour de l'épaule et du coude. La main renvoie un stimulus tactile au contact d'un bloc et peut attraper les blocs qui sont en contact. Le bébé dispose également d'une bouche, d'un champ de vision et d'une fovéa. Le champ de vision est défini par deux droites et est centré sur un angle, le long duquel la fovéa peut se déplacer (pour s'éloigner ou se rapprocher de l'œil).

Le bébé dispose d'un total de dix capteurs :

- Cinq capteurs renvoient un stimulus lors d'une interaction avec un bloc : toucher ou attraper avec la main, toucher ou sucer avec la bouche et voir avec la fovéa.
- Quatre capteurs renvoient les positions du bras, de l'avant-bras, du champ de vision et de la fovéa.
- Le dernier capteur est particulier puisqu'il renvoie la liste des blocs qui sont dans le champ de vision.

Remarquons que les capteurs de position renvoient des valeurs qui peuvent s'apparenter à des tensions musculaires, mais que le dernier capteur renvoie une perception symbolique, et non plus interactionniste. Chaque élément de cette liste d'objets est composé de trois informations : le type d'objet, l'angle entre l'objet et le centre du champ de vision et de la distance à la fovéa. Pour interpréter l'information envoyer par ce capteur, le bébé est donc obligé de connaître a priori les objets qui existent dans son environnement.

[4] Aussi continu que les mondes simulés le permettent.

Rappelons que pour Piaget, le nouveau-né vit dans un environnement qu'il ne connait pas et qu'il construit ses connaissances à partir de cet environnement inconnu, au travers d'interactions successives. De même pour un agent artificiel, le point principal n'est pas de le faire évoluer dans un contexte défini à l'avance afin de comparer son comportement à un être-humain, mais de voir émerger des comportements dans un environnement inconnu de l'agent. Si l'agent est tributaire de l'environnement, il ne construit plus des connaissances. Il organise ses comportements.

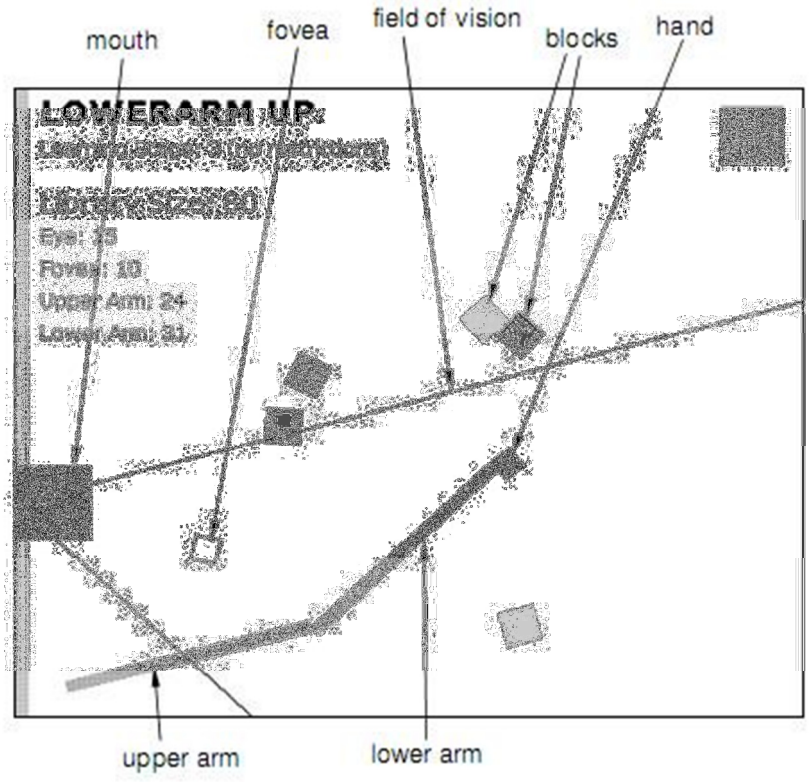

Figure 2.7 : Interface de l'environnement de Guerin et McKenzie

L'environnement de Guerin et McKenzie rejoint sur bien des points l'environnement de Perotto et Alvarez. Il ne permet pas d'analyser les comportements des agents et n'est pas évolutif à cause de son code source non disponible. Son interface simple résume bien l'évolution de la simulation et est ergonomique. Les simulations sont exécutées par Guerin et McKenzie seulement, pour un seul agent et ne sont pas interactive. Pour cause, l'environnement de simulation n'est pas en ligne et n'est pas multi-agent. La grosse différence vient du fait qu'il n'adopte pas une approche interactionniste mais symbolique.

2.3 Comparatif des environnements

Nous avons vu trois environnements « généralistes », qui permettent de tester différents agents de différentes équipes de recherche, et trois environnements « ad hoc », développés par les équipes de recherche en intelligence artificielle pour tester leur agent constructiviste. Afin de comparer plus simplement les différents environnements vus précédemment, le tableau ci-dessous (Tableau 2.1) récapitule les caractéristiques pour chacun des critères cités au début de l'état de l'art.

Critères	Environnements « généralistes »			Environnements « ad hoc »		
	oRis	RoboCode/dTank	Vacuum	Gridland	Environnement de Perotto et al	Environnement de Guerin et al
Interactivité	+	-	+	-	-	-
Immersivité	-	+	+	-	-	-
En ligne	-	+	-	-	-	-
Multi-agent	+	+	-	+	-	-
Ergonomie	--	+	+	+	+	+
Evolutivité	++	-	+	-	-	-
Interactionniste	-	-	-	+	+	-
Analyse de comportements	-	-	-	-	-	-

Tableau 2.1: Comparatif des environnements de simulation

Nous avons vu que les environnements dits généralistes avaient une approche symbolique, influencés par les langages informatique orientés objets. En regardant du point de vue des environnements qui ont une approche interactionniste, nous remarquons bien que le domaine de recherche des agents constructivistes est encore relativement jeune et ne dispose pas d'outils de simulation généralistes. Les équipes de recherche sont obligés de développer leur propre environnement de simulation en parallèle de leur agent. Il est important de voir également qu'aucun des environnements étudiés n'offre pas d'outil d'analyse de comportements d'agents. Afin de détecter et comprendre l'apparition de nouveaux comportements, il semble important que les environnements laissent une trace des interactions entre l'agent et l'environnement.

3 Propositions

L'architecture d'environnement de simulation proposée est conçue pour les problèmes à caractère situé[5]. Ce type de problèmes fait références aux agents qui existe au sein d'un environnement et qui voient leur comportement affecté par ce même environnement.

Nous avons vu qu'il est parfois difficile de situer la frontière entre un agent et l'environnement qui le contient. Le concepteur doit pouvoir déplacer cette frontière. Dans l'architecture de notre environnement de simulation, nous faisons intervenir une entité nommé corps de l'agent (Figure 3.1). Ce *corps* est l'interface entre l'environnement et l'algorithme de l'agent. C'est l'incarnation de l'agent dans l'environnement. Le corps constitue un environnement interne et définit la liste des capteurs et des effecteurs de l'agent. L'environnement interne se compose de propriétés propres et de processus qui régulent ces propriétés. Les capteurs permettent à l'agent de sentir

[5] Traduit de l'anglais « situatedness problem ».

l'environnement (capteurs externes) et son corps (capteurs internes). Nous parlons bien de sensation et non de perception car la perception est un processus actif de l'agent qui prend en entrée les stimuli envoyés par les capteurs. Les effecteurs permettent à l'agent d'agir sur l'environnement (effecteurs externes) et sur son corps (effecteurs internes). Dans notre modèle, l'agent est représenté dans l'environnement pour souligner le fait qu'il est autonome et qu'il n'interagit qu'avec son corps et l'environnement simulé. Ainsi, il n'est pas possible de récompenser l'agent, car cette pratique s'apparente à de l'apprentissage par renforcement.

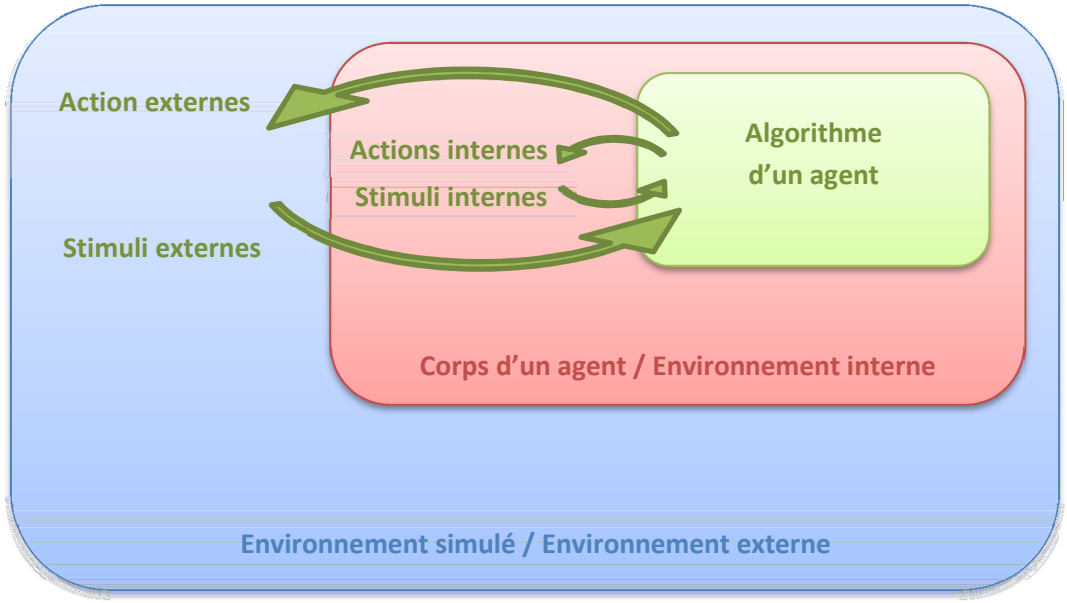

Figure 3.1 : Séparation entre le corps et l'algorithme d'un agent

3.1 Deux niveaux d'utilisateur

Dans le cas des agents extrinsèquement motivés, les tests d'un agent au travers d'une simulation sont effectués par l'équipe de recherche qui développe l'agent. En réalité, ce n'est pas une mais plusieurs simulations qui sont faites. Les résultats obtenus sont ensuite publiés dans un article sous la forme de valeurs synthétiques, qui représente la performance de l'agent pour réaliser la tâche ou atteindre le but. Ces résultats peuvent être de deux types différents :

- Une moyenne sur plusieurs simulations
- Exposition de la meilleure simulation obtenue

La moyenne a l'avantage de prendre en compte l'ensemble des résultats obtenus par les simulations, les bons et les moins. En revanche, une moyenne ne renseigne pas sur la disparité des résultats obtenus. Il faudra alors calculer un écart type. Un autre désavantage de la moyenne est le nombre de simulations effectuées pour obtenir le panel de résultats. Il est fréquent d'avoir des moyennes sur 100 à 1000 simulations, ce qui nécessite un temps de calcul conséquent. Lorsque seuls les résultats de la meilleure simulation sont retenus, ce sont les capacités optimales de l'algorithme qui sont mises en avant, sans en montrer les points faibles.

De plus, les statistiques ne sont pas toujours explicites. Dans le cas d'agents développementaux, ce n'est pas le nombre de comportements qui importe. Il nécessaire d'avoir une augmentation du nombre de comportements connus sans avoir une explosion combinatoire. L'utilité ou la pertinence des comportements sont plus révélateurs de la qualité d'une simulation ou d'un agent, mais ce sont des notions subjectives qu'un ordinateur ne peut pas quantifier.

Dans ce contexte, nous proposons de dissocier les utilisateurs d'un environnement de simulation en deux groupes (Figure 3.2):

- Les observateurs
- Les concepteurs d'agents

Ainsi, les observateurs ne sont plus forcément les concepteurs d'agents eux-mêmes. Un lecteur intéressé par les résultats d'une ou plusieurs simulations peut alors devenir observateur et se faire une meilleure idée de la qualité de l'agent.

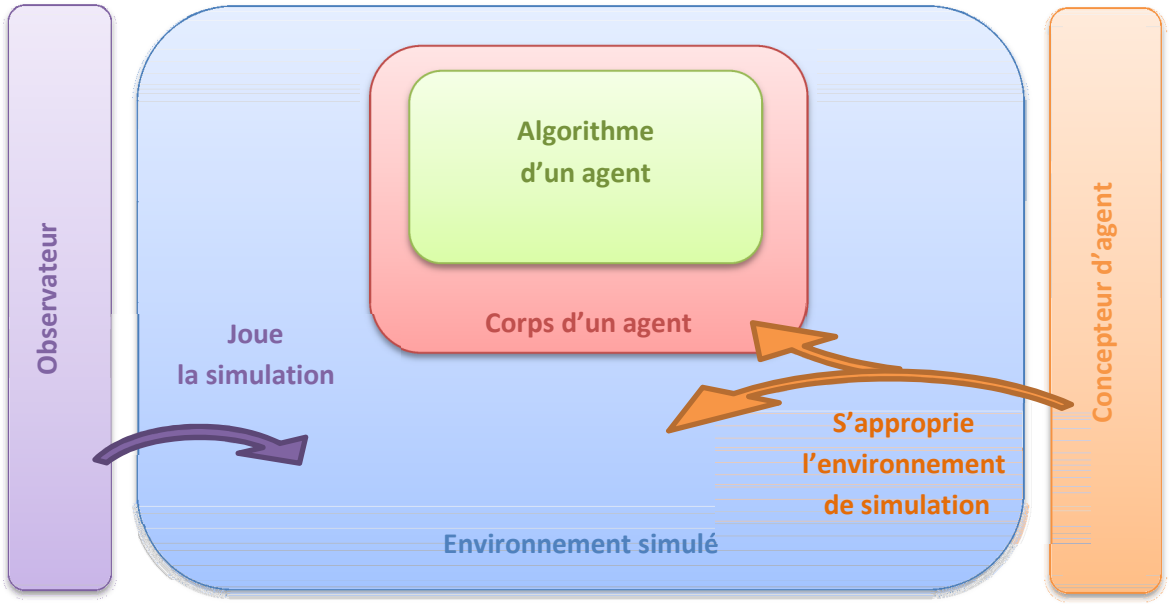

Figure 3.2 : Deux niveaux d'utilisateur : l'observateur et le concepteur d'agent

3.1.1 Les observateurs

Les *observateurs* peuvent être des chercheurs ou des personnes passionnées d'intelligence artificielle. Le but de cette catégorie d'utilisateur et d'élargir le champ des personnes ciblées par une simulation. Ainsi, les observateurs doivent pouvoir faire des actions simples telles que :

- Visualiser une simulation
- Modifier l'environnement simulé
- Voir des informations sur un agent
- Paramétrer un agent
- Visualiser des traces

Pour visualiser une simulation, les contrôles essentiels sont de pouvoir démarrer la simulation, la mettre en pause, la faire avancer pas à pas et la redémarrer. Avec ces contrôles, l'observateur peut observer une simulation mise à sa disposition grâce à un rendu graphique. Le rendu graphique n'apporte rien en soit à une simulation, car celle-ci est une suite de calculs sur un modèle programmé. Mais il est important car il aide l'observateur à comprendre ce qu'il se passe dans l'environnement simulé.

Avec seulement les contrôles précédents, la simulation ressemblerait à une vidéo et pourrait même être remplacée par vidéo, afin de pouvoir profiter des moyens actuels de partage de vidéo sur Internet. Pour couper à cette monotonie, nous avons décidé de faire de l'observateur un acteur de la simulation. Le fait de pouvoir interagir avec l'agent au travers de l'environnement est très utile, voire indispensable pour permettre à l'observateur de s'apercevoir du développement de l'agent (De Loor, Manac'h, & Tisseau, 2010).

La simulation peut se dérouler sans intervention de l'observateur, mais ce dernier peut aussi modifier l'environnement simulé. Par exemple, il peut ajouter des obstacles, de la nourriture, supprimer un mur, etc.

Visualiser la scène simulée aide à comprendre l'évolution de la simulation de manière globale, mais pas l'évolution de chaque agent. Pour cela, l'observateur doit voir plus d'information sur l'agent en question. Il pourrait observer ses stimuli, l'état interne de son corps ou de son esprit. Ainsi, il comprendrait mieux les comportements propres à l'agent et l'émergence de nouveaux comportements.

Sur un agent, les capteurs et les effecteurs peuvent être paramétrés de deux manières. Soit les paramètres sont inscrits dans le code, mais le changement d'un des paramètres demande de recompiler le code et de relancer la simulation. Soit les paramètres sont modifiables directement lors de la simulation et il devient plus simple de comparer des agents qui ont des champs de vision plus ou moins large par exemple. Un exemple sur un effecteur peut être le paramétrage du pas de rotation pour un agent qui peut tourner.

Les différentes visualisations introduites précédemment sont toutes pour un instant donné. En enregistrant les traces d'un agent, nous obtenons des données sur plusieurs cycles. En exploitant ces données, il est possible de rendre des vues extratemporelles. Ainsi, nous pourrions visualiser l'ensemble des points de passage d'un agent et faire ressortir des chemins ou l'agent passe régulièrement. Inversement, il serait possible de détecter des zones d'ombres où l'agent ne va jamais.

Avec les actions présentées, l'observateur n'a plus le rôle d'un observateur traditionnel. Il devient acteur de la simulation. Plutôt que d'observer une simulation, il *joue* cette simulation (Figure 3.2).

3.1.2 Les concepteurs d'agents

Les *concepteurs* d'agent sont des chercheurs en architecture d'agents développementaux. Ils se focalisent sur la modélisation de ce que nous avons appelé l'esprit de l'agent, communément appelé « agent » tout simplement. Afin de tester leur agent dans un environnement, les concepteurs

doivent travailler au niveau du code de l'environnement afin de l'adapter à leurs besoins. Dans ce code, ils doivent pouvoir :

- Ajouter leur agent
- Modifier les objets de l'environnement simulé
- Créer des capteurs et des effecteurs

Pour ajouter leur agent, les concepteurs doivent programmer l'interface entre leur agent et l'environnement. Rappelons que les agents modélisés sont la partie que nous avons appelé esprit de l'agent. L'interface est donc le corps de l'agent.

Du point de vue de l'esprit de l'agent, le corps est constitué de capteurs et d'effecteurs. Le concepteur peut alors réutiliser des capteurs et des effecteurs existant ou définir les siens. Pour chaque effecteur, le concepteur doit associer un son et une animation. Les sons sont des représentations sonores symboliques qui aident l'observateur à s'immerger dans la simulation et à reconnaitre auditivement les actions effectuées. Ils proviennent soit de la liste de sons déjà disponibles, soit de l'adresse url d'un autre son. Les animations sont des représentations visuelles symboliques ou explicites qui ont également le rôle d'aider l'observateur à pleinement percevoir le déroulement de la simulation. C'est également à ce niveau que sont définis les paramètres des effecteurs et des capteurs, si le concepteur veut les rendre paramétrables.

Du point de vue de l'environnement, le corps est une forme géométrique simple (sphère ou pavé pour des simulations en trois dimensions) à laquelle est attachée une représentation complexe, animée et en trois dimensions. La forme simple est utilisée pour la physique de l'environnement, telle que les collisions. La représentation sert au rendu graphique. Pour animer une représentation, le concepteur décompose le corps en plusieurs parties et définit celles qui peuvent être animées.

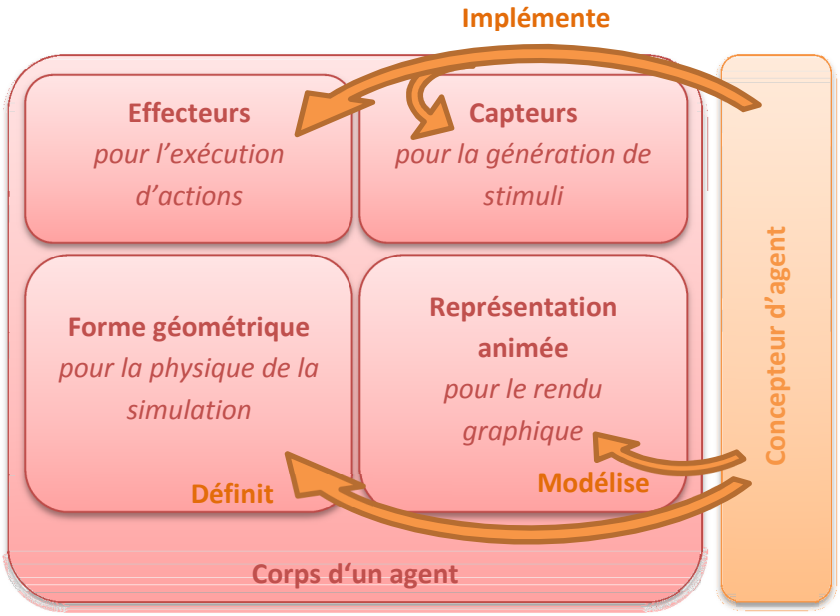

Figure 3.3 : Composition du corps d'un agent

Afin de varier les environnements simulés ou de les adapter à un contexte spécifique (environnement marin pour un poisson par exemple), le concepteur peut modifier les objets de l'environnement. Il peut modifier leur forme, leur texture, leur couleur et leurs propriétés physiques. Le concepteur peut également ajouter de nouveaux objets sans modifier ceux qui existent.

Pour diversifier les actions et les sensations possibles, le concepteur peut également implémenter de nouveaux effecteurs et de nouveaux capteurs. Le type d'effecteurs et de capteurs nécessaires à l'agent dépendent principalement de la position de la frontière entre l'agent et l'environnement que le concepteur a choisi.

3.2 Environnement centré sur les interactions

Pour montrons l'importance donnée aux interactions, nous voulons faire évoluer notre modèle (Figure 3.4). Nous allons voir dans les sections suivantes comment cela est rendu possible.

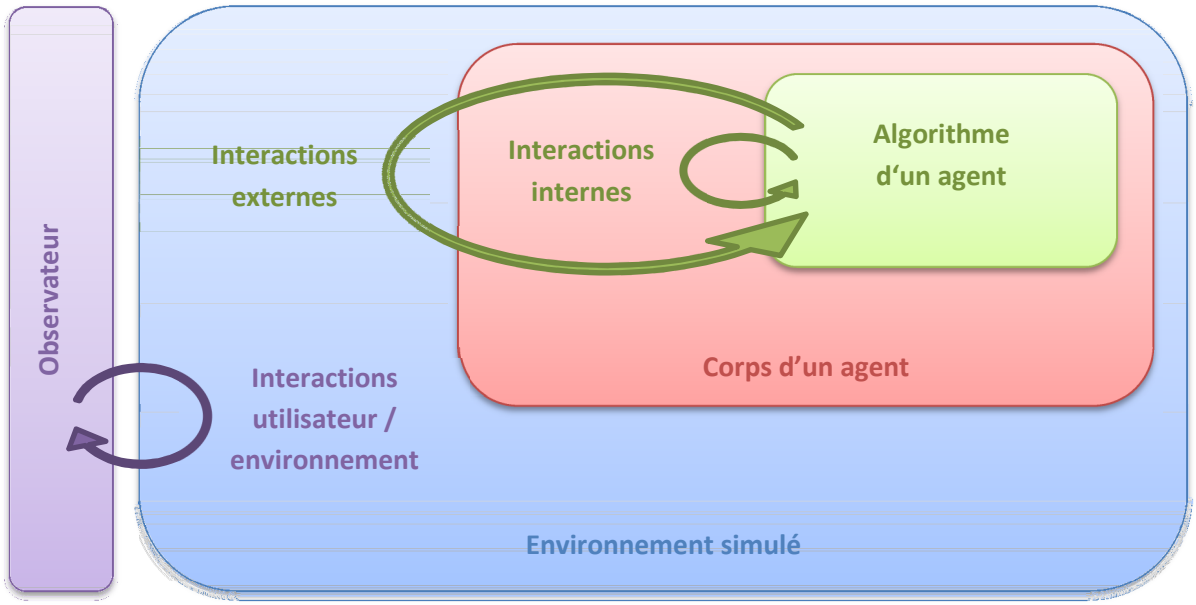

Figure 3.4 : L'interaction comme vecteur d'échanges

3.2.1 Interactions Agent-Environnement

Jusqu'à présent, notre modèle présentait les actions et les stimuli comme des flux indépendants. Cela sous-entend qu'agir et percevoir sont deux processus distincts. Apparaissent alors deux questions :

- Peut-on percevoir sans agir ?
- Peut-on agir sans percevoir ?

La théorie de la perception active répond à la première question. Selon cette théorie, il n'existe pas de perception sans action concrète de la part du sujet percevant. Ainsi, les stimuli qui conduisent à la perception sont tributaires d'une action. Pour la deuxième question, il est difficile d'imaginer une action qui n'engendre pas de perception. En effet, même si la perception n'est pas externe, il existe toujours une perception interne. S'il était possible de faire une action sans en

percevoir ses effets, comment pourrions-nous apprendre à maitriser cette action ? Les deux processus semblent donc intimement liés. Nous définissons alors une interaction comme un couple d'une action et d'un stimulus. Les échanges entre l'agent et l'environnement sont donc réduits à un flux d'interactions (Figure 3.5). Avec cette architecture, l'agent est capable de succéder une interaction à une autre et de former ainsi des chaines d'interactions.

Nous avons vu qu'en plaçant la frontière entre l'agent et l'environnement, le concepteur définit un niveau d'abstraction pour les actions et les stimuli. Ce niveau se reporte donc sur les interactions. Un fort niveau d'abstraction permet une communication plus simple entre l'agent et l'environnement. Suite à une action, l'environnement ne transmet que l'information utile : un stimulus particulier.

Prenons l'exemple d'un agent simple qui peut se déplacer vers l'avant et manger de la viande et de l'herbe. Les actions sont « avancer » et « manger ». Pour l'action « manger », définissons les stimuli « a avancé » si l'agent a correctement avancé et « s'est cogné » si l'agent est bloqué par un obstacle. Pour l'action « manger », prenons les stimuli « a mangé de la viande » « a mangé de l'herbe » et « n'a rien à manger ». Les interactions possibles sont alors :

- (« avancer », « a avancé »)
- (« avancer », « s'est cogné »)
- (« manger », « a mangé de la viande »)
- (« manger », « a mangé de l'herbe »)
- (« manger », « n'a rien à manger »)

Nous remarquons bien que les stimuli sont propres à une action avec des interactions à fort niveau d'abstraction. En plus du niveau d'abstraction, le concepteur peut influer sur le niveau de détails des interactions. Quatre processus permettent de modifier ce niveau de détails en modifiant l'action : par fusion, par différenciation, par spécification et par généralisation.

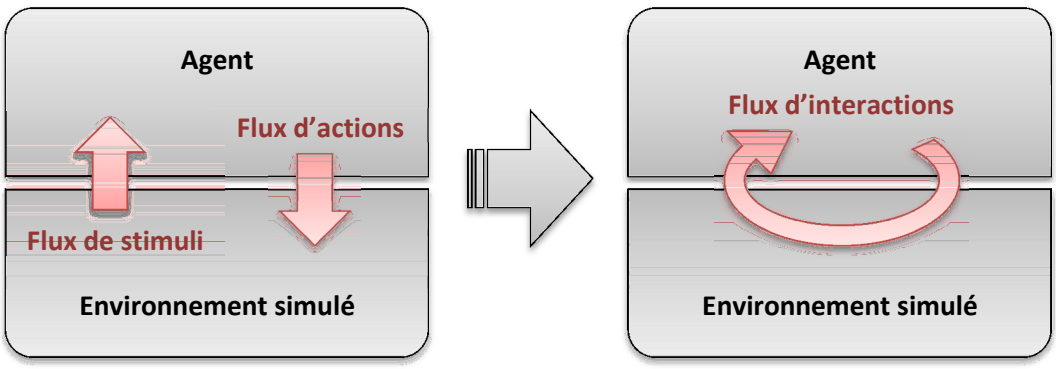

Figure 3.5 : L'action et la perception au travers d'interactions

La fusion consiste à rassembler deux actions pour n'en former plus d'une. Les stimuli possibles deviennent alors l'ensemble des combinaisons entre les stimuli associés à la première action avec les stimuli associé à la seconde action.

La différenciation est l'opération inverse. Elle consiste à séparer une action en deux nouvelles actions. Les stimuli possibles pour chacune des deux actions est une simplification des stimuli

associés à l'action de départ pour ne garder que le sens en rapport avec l'action considérée. Après simplification du sens, plusieurs stimuli peuvent être identiques. Il ne représente alors qu'un seul et même stimulus. Ce processus est utile lorsqu'une action a beaucoup de stimuli qui lui sont rattaché.

La spécification consiste à détailler une action en plusieurs sous actions afin lui donner plus de sens. Seul le nom de l'action change donc il n'y pas de changement au niveau des stimuli. En revanche, ce processus précède souvent un processus de différenciation.

La généralisation est l'opération inverse de la spécification. Elle consiste à simplifier le nom d'une action pour ne pas faire apparaitre les sous actions. Les noms des stimuli peuvent être simplifiés également.

Les processus de fusion et de généralisation permettent de diminuer le niveau de détails alors que les processus de différenciation et de spécification permettent d'augmenter le niveau de détails. Il existe également deux processus qui permettent de modifier le niveau de détails en modifiant les stimuli : par diversification et par simplification.

La diversification consiste à scinder un stimulus en deux nouveaux stimuli. Ce processus permet de détailler, complexifier la perception.

La simplification est l'opération inverse. Elle consiste à fusionner deux stimuli entre eux, ce qui permet de simplifier la perception.

Reprenons l'exemple introduit précédemment afin d'illustrer ces six processus qui permettent au concepteur de changer de niveau de détails :

- Fusion :
  - « avancer » + « manger » => « avancer et manger »
  - Les stimuli possibles sont au nombre de 6 :
    - « a avancé et a mangé de la viande »
    - « a avancé et a mangé de l'herbe »
    - « a avancé et n'a rien à manger »
    - « s'est cogné et a mangé de la viande »
    - « s'est cogné et a mangé de l'herbe »
    - « s'est cogné et n'a rien à manger »
- Généralisation :
  - « avancer et manger » => « avancer »
  - Les stimuli se simplifient également :
    - « a avancé et a mangé de la viande »
    - « a avancé et a mangé de l'herbe »
    - « a avancé »
    - « s'est cogné et a mangé de la viande »
    - « s'est cogné et a mangé de l'herbe »
    - « s'est cogné »
- Spécification :
  - « manger » => « mâcher et avaler »
  - Les stimuli peuvent être renommés également :

            - « a mâché et avalé de la viande »
            - « a mâché et avalé de l'herbe »
            - « n'a rien à mâcher et avaler »
    - Différenciation :
        - « mâcher et avaler » => « mâcher » + « avaler »
        - Les stimuli pour l'action « mâcher » deviennent :
            - « a mâché de la viande »
            - « a mâché de l'herbe »
            - « n'a rien à mâcher »
        - Les stimuli pour l'action « avaler » deviennent :
            - « a avalé de la viande »
            - « a avalé de l'herbe »
            - « n'a rien à avaler »
    - Diversification :
        - « a avalé de la viande » => « a avalé de la viande (mâchée) » + « ne peut pas avaler de la viande non mâchée »
        - « a avalé de l'herbe » => « a avalé de l'herbe mâchée » + « a avalé de l'herbe non mâchée »
    - Simplification :
        - « a mangé de la viande » + « a mangé de l'herbe » => « a mangé »

Les interactions à fort niveau d'abstraction vues précédemment sont appelées interactions primitives. Pour augmenter le réalisme et reproduire les sens observés dans la nature, il faut se placer à faible niveau d'abstraction. En effet, lorsque l'homme effectue une action, c'est un ensemble de stimuli qu'il perçoit en retour au sein desquels certains feront sens pour l'action considérée. L'information utile est donc noyée au milieu de l'information reçue. A un faible niveau d'abstraction, les interactions au niveau de l'interface entre l'environnement et l'agent sont appelées interactions élémentaires. Pour traiter les stimuli sensoriels reçus, un agent a besoin d'un système sensorimoteur qui permet d'abstraire une interaction élémentaire en une interaction primitive.

Une interaction élémentaire a une représentation différente d'une interaction primitive : c'est toujours un couple mais il est composé d'une action et d'une matrice de stimuli. Cette matrice de stimuli a la même forme pour toutes les actions mais les stimuli qui la composent varient suivant l'état de l'environnement simulé (environnement externe) et du corps de l'agent (environnement interne). Une matrice de stimuli apporte une explosion combinatoire des possibilités de stimulation. Le système sensori-moteur permet de gérer cette explosion combinatoire en synthétisant les stimuli sensoriels en un stimulus abstrait.

Les stimuli sensoriels qui composent la matrice sensorielle peuvent provenir de l'environnement simulé et/ou du corps de l'agent. Les cinq sens courant (vision, audition, touché, olfaction et la gustation) nécessitent des stimuli provenant de l'environnement simulé. Pour qu'un agent perçoive la position de ses membres ou qu'il perçoive des propriétés physiologiques telles que la faim, la soif ou la douleur, il est nécessaire d'avoir des stimuli provenant du corps de l'agent. Ainsi ce sont bien l'environnement externe (environnement simulé) et l'environnement interne (corps de l'agent) qui génèrent la matrice sensorielle.

Nous retrouvons également les trois catégories des systèmes sensoriels décrits en psychologie :

- Extéroception : perception de l'environnement (au travers des cinq sens courants)
- Proprioception : perception du corps (des tensions musculaires)
- Intéroception : perception de la condition physiologique du corps (faim, douleur, etc.)

3.2.2 Interactions Observateur-Environnement

Nous proposons de ne pas limiter la possibilité d'interactions aux agents seulement, mais de l'étendre à l'observateur. En devenant acteur, l'observateur s'immerge au cœur de la simulation. Il peut interagir avec l'agent de manière indirecte, c'est-à-dire sans agir directement sur lui, pour vérifier l'apprentissage autonome de l'agent et mieux interpréter cet apprentissage.

D'une part, l'observateur peut vérifier la validité d'un apprentissage. En modifiant les conditions initiales ou l'évolution d'une simulation, l'observateur peut remarquer la présence ou l'absence d'apprentissage. Des résultats synthétiques ou une vidéo du comportement d'un agent ne peuvent pas garantir de la validité d'un apprentissage. L'observateur avoir le contrôle sur la simulation. Afin d'éviter les pièges de la simulation, nous voyons de plus en plus de projet d'apprentissage sur des robots. Dans la réalité, la frontière entre le robot (ou l'agent) et l'environnement est clairement définie et ne peut pas être modifiée, ni contestée. En informatique, c'est le concepteur d'agent qui définit la frontière entre l'environnement simulé et l'agent. Pour pouvoir vérifier la validité des apprentissages sous environnement simulé, nous avons choisi de diffuser les simulations pour chacun puisse faire ses propres observations.

D'autre part, l'observateur peur vérifier la fiabilité d'un apprentissage. Après s'être assuré de la validité d'un apprentissage, un observateur peut tester les capacités et les limites de l'apprentissage. Nous proposons de calquer les méthodes utilisées en psychologie pour l'apprentissage infantile, qui consiste à observer les stades d'évolution. De la même manière, il devient possible d'observer les agents, avec encore plus de précisions puisque nous avons accès à toutes les variables définissant l'état interne d'un agent. De plus, la différence entre la création de sens pour les machines et pour les humains pose un sérieux problème dans l'évaluation des agents. Afin de permettre à un humain de guider l'agent dans son apprentissage, il a déjà été proposé d'intégrer l'humain dans le processus (De Loor, Manac'h, & Tisseau, 2010). Au travers des interactions Observateur-Environnement, c'est ce qui est rendu possible.

3.3 Environnement utilisant des traces

Dans sa définition générale[6], « une trace est une empreinte ou suite d'empreintes, de marques que laisse le passage d'un être ou d'un objet ; une marque laissée par une action quelconque ; ce à quoi on reconnaît que quelque chose a existé, ce qui subsiste d'une chose passée ». Dans notre contexte, une trace numérique est issue de l'observation d'une activité, elle présente une marque laissée lors d'une interaction entre deux parties. Une trace numérique est composée d'objets qui sont situés temporellement les uns par rapport aux autres.

[6] Dictionnaire Petit Robert. Version électronique du nouveau Petit Robert de la langue française, 2011.

Les traces sont très utilisées dans le domaine des Environnements Informatiques pour l'Apprentissage Humain (EIAH), notamment pour l'évaluation des sujets apprenants. Nous avons choisi d'utiliser des traces afin d'évaluer les agents. Une trace enregistre les interactions d'un agent avec l'environnement simulé lors d'une simulation. Dans le monde des machines, il est possible d'étendre ces possibilités et d'enregistrer l'évolution de l'état interne d'un agent. En effet, pour une être humain, on ne peut enregistrer que ce qui observable, hors seules les interactions sont observables simplement. De manière générale, l'enregistreur de traces n'a pas accès aux données physiologiques du sujet. En informatique, nous avons accès à toutes les variables. Les traces sont donc plus denses.

L'implémentation d'un enregistreur de traces au travers d'un service web a fait l'objet d'un autre stage de master de recherche en informatique. Nous ne rentrerons donc pas les détails de l'outil appelé Abstract-Lite.

Dans notre contexte, les agents ne sont pas les seuls à interagir avec l'environnement simulé. L'observateur a son propre rôle dans l'évolution de la simulation. L'étude d'une trace seule d'un agent n'explique pas le déroulement d'une simulation, étant donné que les actions de l'observateur n'y sont pas enregistrées. L'environnement simulé doit donc avoir sa propre trace.

4 Implémentation : l'environnement SECA

4.1 Introduction

Pour développer un outil qui implémente les propositions précédemment décrites, je me suis initialement tourné vers deux technologies bien différentes :

- Blender
- WebGL

Blender est un logiciel libre et gratuit d'animation, de modélisation et de rendu 3D. Grâce à son moteur de jeu intégré, Blender peut être utilisé comme moteur 3D gérant le rendu et la logique d'un jeu ou d'une simulation interactive. De plus, c'est un programme extensible (ajout de scripts) à l'aide du langage Python.

WebGL est une spécification d'affichage pour les navigateurs web. Elle permet d'utiliser le standard OpenGL depuis le code JavaScript d'une page web. C'est une technologie permettant l'affichage de contenus 3D sur les pages web sans avoir à installer de plugins.

L'avantage de Blender est la gestion complète de la simulation et du rendu grâce au Blender Game Engine, le moteur de jeu intégré. Il propose un ensemble de capteurs et d'effecteurs et il est possible d'implémenter ses propres scripts en Python. En revanche, Blender ne satisfait pas deux critères de l'implémentation voulue : la simplicité d'utilisation et le partage des simulations sur Internet. En effet, l'utilisation de Blender propose de très nombreuses fonctionnalités et son apprentissage demande du temps. Il n'est pas dans l'idée de ce projet de transformer les

concepteurs d'agent en professionnels de la modélisation 3D. De plus, diffuser les simulations sur Internet demanderait soit aux observateurs de télécharger Blender pour pouvoir jouer les simulations, soit un puissant serveur qui ferait tourner les simulations et qui n'enverrait que les images aux observateurs. Dans ce dernier cas, quand plusieurs observateurs demandent à voir des simulations en même temps, la charge du serveur augmente fortement.

WebGL permet un rendu 3D directement au sein des pages web ce qui facilite la diffusion des simulations. La charge de calcul demandée par le rendu est alors à la solde de l'observateur et l'accélération matérielle, offerte par OpenGL, permet d'utiliser la puissance de calcul du processeur des cartes graphiques, optimisé pour les rendus 3D. En revanche, WebGL ne gère pas la simulation. Il faut donc développer un moteur de simulation et un moteur de rendu 3D en JavaScript.

C'est cette dernière technologie qui a été retenu pour implémenter notre environnement de simulation interactive. Mais les moteurs de de simulation et de rendu n'ont pas été codés en JavaScript mais en Java. En effet, grâce au Google Web Toolkit (GWT), il est possible de compiler du code Java en code JavaScript.

GWT est un ensemble d'outils logiciels développé par Google, permettant de créer des applications web dynamiques mettant en œuvre JavaScript, en utilisant le langage et les outils Java. GWT met l'accent sur des solutions efficaces et réutilisables aux problèmes rencontrés habituellement par le développement AJAX (Asynchronous Javascript and XML) d'applications web : difficulté du débogage JavaScript, gestion des appels asynchrones, problèmes de compatibilité entre navigateurs. Lors de la phase de développement, l'application est écrite en Java et c'est seulement lors du déploiement que l'application est compilée en JavaScript.

C'est donc en Java sous Eclipse avec le plugin GWT que j'ai développé l'application SECA (Simulation Environment for Constructivist Agents).

Environnement discrétiser, sous la forme d'une grille 3D de blocs

Après compilation, le programme est exécuté en JavaScript, or ce langage n'est pas orienté objet. Pour décrire SECA, nous utiliserons tout de même des termes réservés aux langages orientés objet étant donné que le programme est écrit en Java et que le Java est un langage orienté objet.

4.2 Architecture de l'environnement

SECA dispose d'une architecture 3-tiers afin de séparer les trois couches logicielles. L'application a donc une couche présentation, une couche métier et une couche accès aux données.

La couche présentation (en vert sur la Figure 4.1) correspond à l'interface utilisateur, la partie visible de l'application pour l'utilisateur. Pour SECA, c'est la seule partie que verra un observateur. C'est une page HTML qui comporte la liste des commandes permettant d'interagir avec l'environnement, une visualisation de l'environnement simulé avec les agents qui le peuplent et un tableau de bord. Ce tableau de bord permet de visualiser l'état interne de la simulation (par exemple : le nombre d'images par seconde) et des agents (le nombre de cycle, les stimuli). Cette couche comporte deux classes : SECA et Tableau de bord.

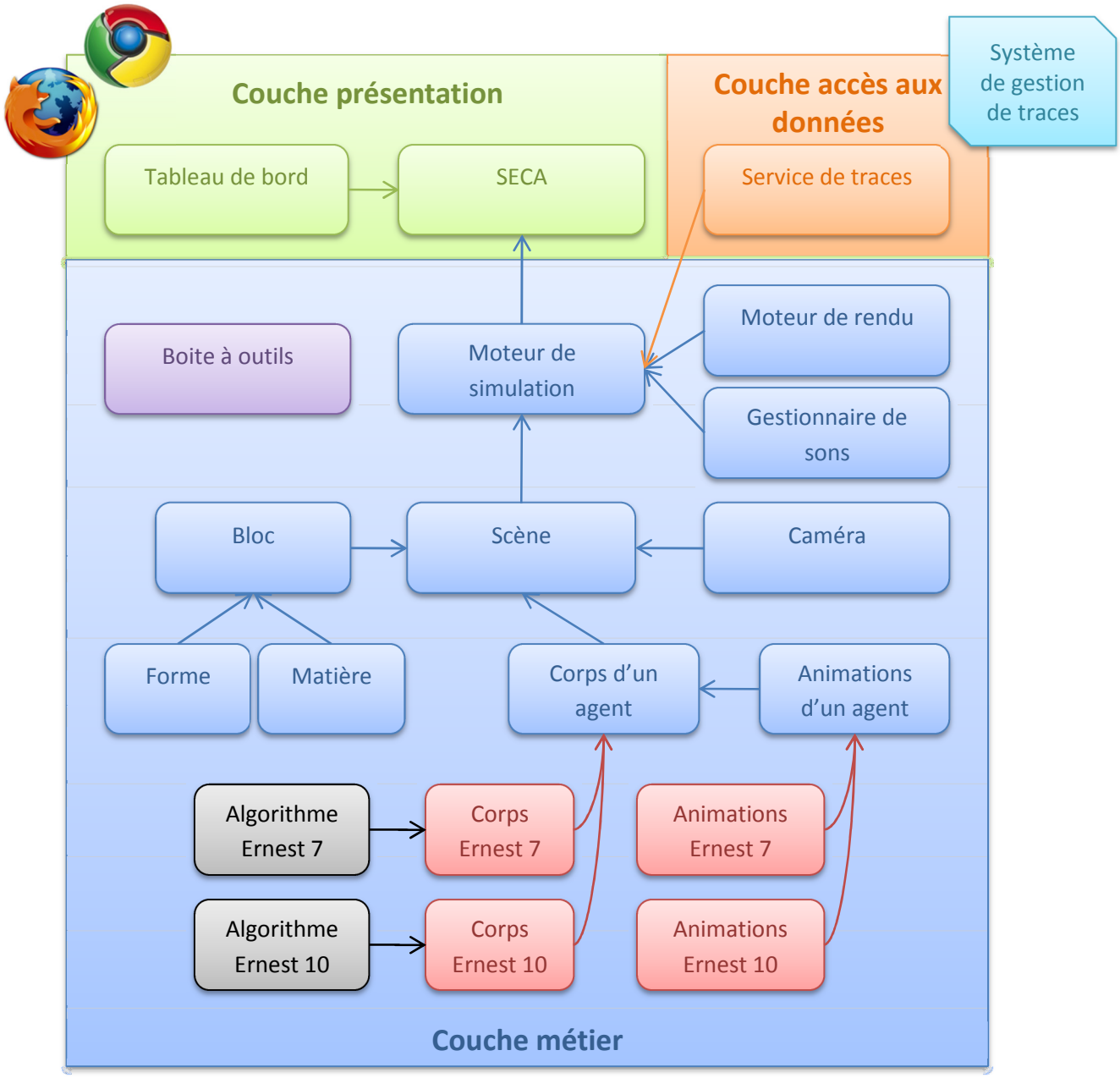

Figure 4.1 : Architecture de SECA

La classe SECA gère la page HTML. Elle implante la zone d'affichage de 1000 pixels de large sur 500 pixels de haut pour le rendu de l'environnement simulé. Elle ajoute également les boutons de contrôle de la simulation qui sont au nombre de six :

- Start : pour initialiser la simulation. La scène, les agents et les caméras sont instanciés et la simulation est mise en pause.
- Stop : pour arrêter la simulation. Les instances de la scène, des agents et des caméras sont supprimées.
- Play : pour démarrer la simulation. La scène et les agents sont alors mis à jour à intervalle régulier.

- Pause : pour mettre en pause la simulation. Les agents ne sont plus mis à jour mais l'observateur peut toujours se déplacer dans la scène pour changer de point de vue, ou la modifier.
- Step : pour faire avancer la simulation pas à pas. Les agents sont mis à jour une seule fois et la simulation retourne en mode pause.
- Restart : pour réinitialiser la simulation. De nouvelles instances sont créées pour la scène, les agents et les caméras.

Cette classe associe également un tableau de bord à un emplacement situé sous la zone de rendu.

La classe Tableau de bord propose aux concepteurs d'agent de créer différents composants d'interface graphique, appelés widgets. Certains widgets sont des sorties, ils permettent d'afficher des informations sur l'état interne d'un agent ou sur ses stimuli. D'autres widgets sont des entrées, ils permettent de paramétrer l'agent. Nous verrons plus en détail ces widgets dans la section 4.4.

La couche métier (en bleu sur la Figure 4.1) s'occupe du traitement de l'information. Elle met à jour l'environnement simulé et calcule le rendu 3D destiné à être affiché par la couche présentation. Ces deux étapes sont répétées en boucle. La mise à jour de l'environnement simulé consiste à :

- Traiter les entrées utilisateur : quand l'observateur appuie sur des commandes, c'est ici qu'elles sont prises en compte et que l'environnement est modifié en conséquent.
- Calculer les stimuli des agents : pour chaque capteur de chaque agent, les stimuli sont calculés en fonction de la position et de l'orientation de l'agent dans l'environnement.
- Lancer les algorithmes des agents : chaque agent déroule son algorithme décisionnel qui prend en entrée les stimuli de la dernière interaction effectuée et qui retourne la prochaine action à effectuer.
- Déplacer, mettre à jour les corps des agents : en fonction des actions retournées par les algorithmes des agents, les positions, les orientations et l'état interne des agents sont mis à jours.
- Résoudre les collisions : c'est la partie qui gère la physique de la simulation. Suite aux déplacements des agents, des incohérences peuvent apparaître telles que la superposition spatiale d'éléments. Ces superposition d'éléments sont appelées collisions. Pour que les obstacles modifient les mouvements des agents, il est nécessaire de prendre en compte ces collisions et limiter le déplacement des agents.
- Jouer les sons : les sons associés aux actions sont démarrés à ce niveau. Le gestionnaire de sons s'occupe ensuite de la continuité de la lecture des sons.

La classe Moteur de simulation contient la boucle de simulation qui ordonnance les tâches vues précédemment, ainsi que le traitement des entrées utilisateur provenant du clavier. Si l'observateur appuie sur une touche, c'est donc ici que l'action associé est exécutée (Voir l'annexe B pour la liste des commandes propres à SECA).

La classe Scène décrit la composition d'un environnement simulé. Une scène peut contenir des caméras, des blocs et des corps d'agents. Pour les caméras et les corps d'agents, la scène est pseudo-continue. En effet, leurs positions sont codées avec des nombres à virgule flottante, qui ne sont que des approximations des nombres réels. La scène est donc bien discrète (limite de l'informatique)

mais étant donné l'échelle à laquelle se fait la discrétisation, nous pouvons considérer la scène comme continue. En revanche, pour les blocs, la scène est clairement discrète, sous la forme d'une grille à trois dimensions. Les cellules de cette grille sont des cubes unitaires et les positions des blocs sont codées par des nombres entiers. Nous avons fait l'hypothèse d'une simulation à base de blocs car la modélisation d'un environnement devient très facile graphiquement et la réduction des positions à des cellules n'est pas un handicap pour les agents. Après comparaison du gain en facilité d'utilisation et de la perte de personnalisation, nous pensons que les possibilités de modélisation sont augmentées pour les concepteurs d'agents et les observateurs non spécialistes de modélisation 3D.

La classe Caméra permet de créer des points de vue personnalisables et modifiables à volonté. Ces points de vue sont appelés caméras. Une caméra est définie par trois vecteurs :

- Son vecteur de position, appelé *Eye*. La caméra est alors positionnée mais non orientée.
- Le vecteur de position de la cible, appelé *Target*, qui permet d'orienter la caméra en direction d'une position de l'espace. La caméra s'oriente selon l'axe Eye-Target mais peut encore tourner autour de cet axe.
- Le vecteur d'orientation verticale, appelé *Up*, qui permet de définir la direction correspondant au haut de la caméra. La caméra devient donc complément positionnée et orientée.

Il est possible de créer plusieurs caméras et de passer de l'une à l'autre pendant une simulation. Il est également possible d'attacher une caméra à un agent afin que celle-ci le suive dans ses déplacements. Quand le concepteur d'agent attache une caméra à un agent, il peut choisir quels vecteurs attacher ou pas. Pour obtenir une vue subjective, il est nécessaire d'attacher les trois vecteurs. Pour avoir une caméra qui ne se déplace pas mais qui reste centrée sur l'agent, il suffit d'attacher le vecteur Target.

La classe Bloc définit les objets qui servent à remplir une scène. Ces objets sont la réunion d'une forme, d'une matière, d'une couleur et d'une texture. La texture est utilisée uniquement pour le rendu de la scène destiné à l'observateur. Les agents perçoivent la couleur des blocs et non leur texture. Nous avons choisi d'ajouter les couleurs uniformes et les substituer aux textures pour les agents afin de simplifier les traitements et d'éviter une charge de calcul supplémentaire. En effet, l'analyse des textures en traitement d'imagine n'est pas simple et reste une question fondamentale à l'heure actuelle. L'hypothèse simplificatrice que nous avons faite permet de ne pas déborder sur un domaine de recherche car l'utilisation de couleurs uniformes nous est suffisante. De plus, les changements de luminosité et les ombres ne sont pas calculés pour les perceptions des agents.

La classe Forme définit les formes que peuvent prendre les blocs. Certaines formes sont très simple (comme les cubes) ou très complexes (comme les plantes avec plusieurs milliers de sommets). Le concepteur d'objet peut ajouter de nouvelles formes trouvées sur Internet ou faite à l'aide d'un logiciel de modélisation 3D. Le format du fichier qui contient la forme doit être « .obj ». Cependant, de nombreux fichiers de ce type contiennent des formes décrites par des quadrilatères. WebGL ne permet pas le rendu de quadrilatères, mais seulement de points, de lignes et de triangles. C'est pourquoi j'ai développé un outil (Seca-Quads2Triangles.jar) qui permet de convertir ces quadrilatères en triangles. Ainsi, tous les fichiers « .obj » peuvent être importés dans SECA.

La classe Matière définit les propriétés physiques qu'ont les blocs. Par physique, nous entendons la physique de simulation et non la physique réelle. Actuellement, quatre propriétés peuvent être définies : la possibilité de traverser, la possibilité de casser, la possibilité de manger et la propreté (ou la possibilité de nettoyer). Cette dernière propriété est particulière et tend à disparaître. Elle est un reliquat de l'ancien environnement de simulation, qui était une version personnalisée de Vacuum Cleaner Environment. L'agent avait pour mission de trouver les cases sales et de les nettoyer. A partir de ces propriétés, quatre matières sont déjà implémentées :

- La matière « mur », qu'on ne peut pas traverser, ni manger, ni casser, ni nettoyer.
- La matière « air », qu'on peut traverser, mais pas manger, ni casser, ni nettoyer.
- La matière « saleté » qu'on peut traverser et nettoyer, mais pas manger, ni casser.
- La matière « nourriture », qu'on peut traverser, manger et nettoyer mais pas casser.

Il est possible de traverser les matières « saleté » et « nourriture » car les actions de nettoyer et de manger se font sur le bloc dans lequel se trouve l'agent. L'agent doit donc d'abord rentrer dans le bloc à nettoyer ou à manger d'effectuer l'action. Avec les quatre propriétés définies précédemment, le nombre total de combinaisons possible est de seize. Le concepteur d'agent peut donc créer de nouvelles matières. Il peut également ajouter de nouvelles propriétés, comme « la possibilité de boire » ou une propriété plus réaliste encore : la propriété « liquide ». Si une matière est liquide, nous pourrions en déduire qu'il est possible de la boire. Cependant, avec plus de propriétés, il est possible de faire apparaitre des combinaisons privées de sens. Ceci arrive si deux propriétés sont antagonistes, comme « la possibilité de casser » et « liquide ». En effet, il est possible de casser uniquement des objets solides. Les possibilités d'évolution sont nombreuses mais il faut faire attention au sens donné aux matières.

La classe Corps d'un agent est une classe abstraite qui contient les propriétés essentielles d'un agent, comme sa position et son orientation, et les méthodes nécessaires à la mise à jour et à au rendu d'un agent. Lorsque le concepteur d'agent implémente une classe représentant le corps d'un agent (Corps Ernest 7 et Corps Ernest 10 sur la Figure 4.1), elle doit hériter de cette classe abstraite. Le corps d'un agent comporte son modèle 3D ainsi que les méthodes jouant le rôle des capteurs et des effecteurs de l'agent. La classe abstraite Corps d'un agent gère également l'appel aux animations. Jusqu'à présent, nous avons vu que dans la boucle de simulation, il y a avait alternance entre les mises à jour de l'environnement simulé (avec les agents) et les rendu graphiques, or nous allons voir que c'est un peu plus compliqué que cela. Nous avons recours à des animations pour deux raisons :

- Avoir un temps simulé réaliste
- Diminuer la charge de calcul

En effet, pour obtenir un confort visuel, nous avons besoin de calculer 25 à 30 images par seconde. Si nous adoptions la même cadence pour la mise à jour des agents, ceux-ci semblerait très rapide ou en mode accéléré. Le temps simulé serait plus rapide que le temps réel. La simulation se déroulerait tellement vite qu'il serait impossible d'observer correctement les agents. De plus mettre à jour les agents 25 à 30 fois par seconde demanderait une charge de calcul conséquente et si le processeur ne supporte pas une telle charge, cela pourrait nous empêcher d'atteindre la cadence voulue. En diminuant simplement la cadence de mise à jour des agents, l'environnement simulé serait identique sur plusieurs rendus graphiques et un effet de saccade apparaîtrait. Nous utilisons donc des

animations pour réduire la cadence de mise à jour des agents tout en évitant l'effet de saccade. En définissant un temps d'animation de 500ms, les agents ne sont mis à jours plus que deux fois par seconde. Lors des autres mises à jour de l'environnement, ce ne sont pas les agents qui sont mis à jour mais leur animation. Les animations décomposent les mouvements des agents en plusieurs étapes et assure une fluidité à la simulation.

La classe Animations d'un agent est également une classe abstraite. Elle contient les méthodes qui initialisent les animations, calculent leur avancement et les finalisent. Lorsque le concepteur d'agent implémente le corps d'un agent, il doit également décrire ses animations dans une nouvelle classe qui hérite de la classe abstraite Animations d'un agent. Cette nouvelle classe n'est pas une grosse charge de travail pour le concepteur d'agent. En effet, les animations sont construites par interpolation. Le concepteur d'agent a seulement à décrire la position finale d'une animation et les étapes pour passer de la position de repos de l'agent à la position finale sont générées automatiquement. Pour les animations cycliques (qui reviennent à la position de repos), le concepteur d'agent, décrit la position intermédiaire et précise le nombre de cycles à effectuer. La description d'une position d'un agent se fait par la description de la position de tous les membres amovibles de son modèle 3D.

La classe Moteur de rendu collecte tous les objets qui doivent être pris en compte lors du rendu graphique de la scène. Ainsi, avant d'envoyer les données à la carte graphique, il peut regrouper les objets de même type et optimiser les transferts.

La couche d'accès aux données (en orange sur la Figure 4.1) est la partie qui gère l'accès et l'enregistrement des données du système. Pour SECA, cette partie se limite à l'enregistrement des traces des agents dans un système de gestion de traces.

La classe Service de traces s'occupe de l'envoie des traces au serveur web. Cet envoie est asynchrone et ne bloque donc pas le déroulement de la simulation. Le serveur distant réceptionne les traces, contrôle leur intégrité et les envoie au système de gestion de traces Abstract Lite.

4.3 Interactivité de l'environnement

En plus de pouvoir se déplacer dans l'environnement simulé grâce aux touches du clavier, l'observateur peut également interagir avec la souris. En effet, il est possible de sélectionner un bloc avec la souris (Puce 1 Figure 4.2). A partir de la position 2D de la souris, le moteur de simulation fait une projection inverse et retrouve le bloc situé sous la souris. En réalité, la sélection comporte un bloc et une face de bloc. Ainsi, l'observateur peut supprimer le bloc sélectionné par un clic droit de la souris, ou ajouter un nouveau bloc contre la face sélectionnée par un clic gauche de la souris.

La sélection du nouveau bloc à ajouter se fait au travers du tableau de bord. Le widget « liste de choix » (Puce 2 Figure 4.2) permet à l'observateur de sélectionner un bloc parmi la liste des blocs disponibles.

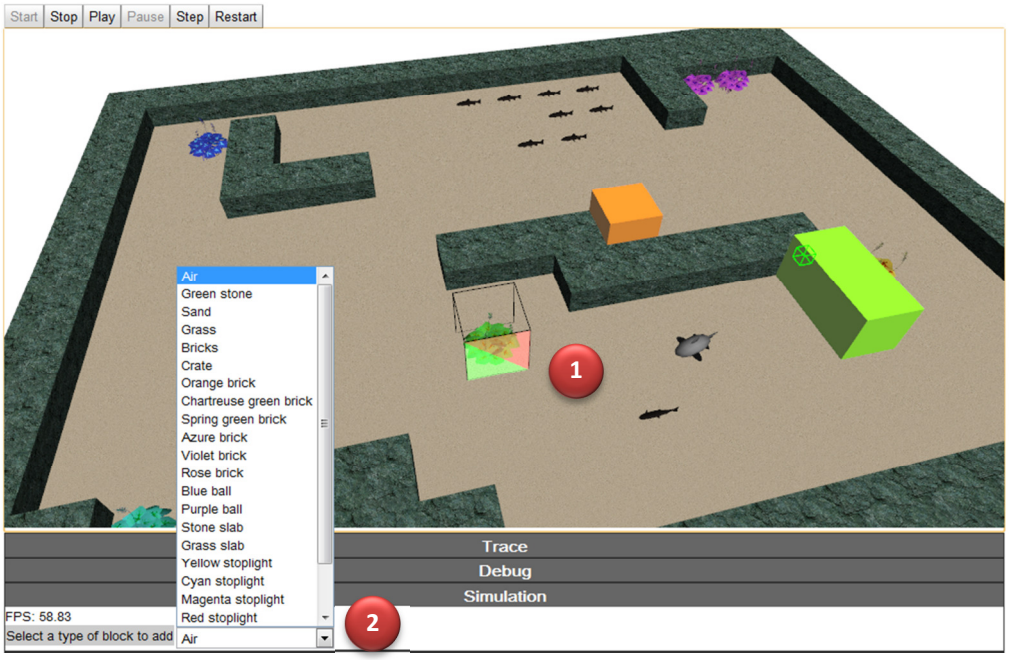

Figure 4.2 : Interactivité avec la souris

Ainsi, même s'il est possible de remodeler l'environnement simulé à partir du clavier, il est beaucoup plus aisé d'utiliser la souris et l'observateur peut rapidement modifier la scène, pour ajouter des poissons par exemple, alors que la simulation est en marche.

4.4 Tableau de bord : visualisation d'indicateurs

Le tableau de bord offre un panneau (Puce 1 Figure 4.3) pour visualiser les stimuli et les états internes d'un agent. L'exemple donné a été réalisé avec l'agent Ernest 10.

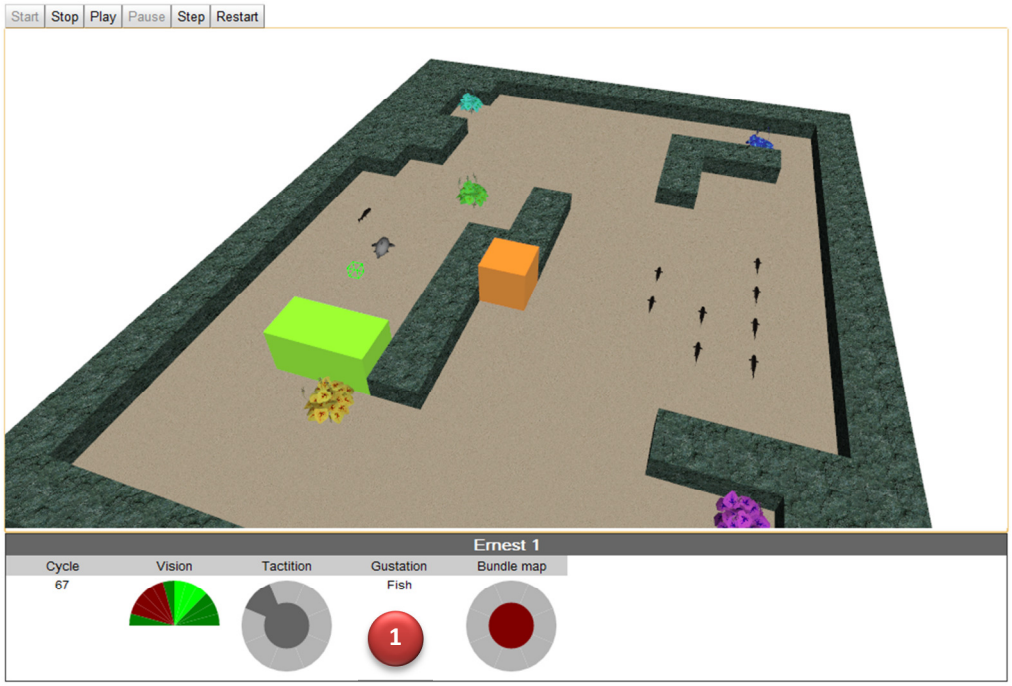

Figure 4.3 : Widgets de visualisation

Le premier widget donne le nombre de cycle que l'agent a déjà effectué. Les trois widgets suivant (Vision, Tactition et Gustation) correspondent aux stimuli envoyés à l'agent. Rappelons que ces stimuli sont calculés au moment de la mise à jour de l'agent et non pendant les animations, d'où le décalage entre les indicateurs et la position de l'agent, celui-ci étant en pleine animation pour tourner de 45 degrés sur la gauche. Le dernier indicateur (Bundle map) est spécifique à l'agent Ernest. Il permet de visualiser l'état interne de l'algorithme, pour vérifier son bon déroulement par exemple.

5 Conclusion

5.1 Bilan

Nous avons vu dans ce rapport que l'évaluation des agents développementaux et intrinsèquement motivés est un réel problème. Les environnements de simulation actuels étant conçu pour des agents dont le but est de résoudre des problèmes, ne sont pas adaptés. L'enjeu était alors de proposer un modèle et une implémentation d'un nouvel environnement de simulation répondant à des critères spécifiques (interactivité, immersivité, en ligne, multi-agent, ergonomie, évolutivité et interactionniste). Ces critères résument la volonté de produire un nouveau protocole de test destiné aux concepteurs d'agent développementale et intrinsèquement motivé, qui reste simple d'utilisation et adaptable aux différents cas d'utilisation.

Nous avons alors proposé un modèle qui tient compte de deux niveaux d'utilisateur afin de séparer les concepteurs d'agent, qui prépare les simulations, et les observateurs, qui jouent les simulations. Le partage des résultats évolue des valeurs statistiques traditionnelles vers la possibilité de rejouer les simulations, de modifier leur cours afin d'étudier l'apprentissage des agents, au même titre que les psychologues étudient l'apprentissage chez l'être humain. Nous avons ensuite centré notre modèle sur les interactions. D'un côté, les interactions agent-environnement permettent de ne plus faire de distinction entre les processus d'action et de perception. La perception se fait au travers de l'action. D'un autre côté, les interactions observateur-environnement permettent de faire participer l'humain aux simulations, sans qu'il n'agisse directement sur l'agent, comme pour les apprentissages à base de récompenses. Enfin, nous avons précisé l'importance des traces qui sont des résultats de simulation permettant d'étudier a posteriori les comportements et l'apprentissage des comportements des agents.

La présentation de SECA, implémentation du modèle mis en place, est venue par la suite. Nous avons décrit son architecture, l'interactivité du point de vue de l'observateur qui devient acteur à part entière dans les simulations et le tableau de bord qui offre une représentation des stimuli reçus par l'agent et de son état interne.

5.1.1 Retour sur la problématique

Rappelons que la frontière entre un agent et un environnement simulé est floue. Afin de situer cette frontière dans notre modèle, nous avons mis en place un modèle d'interaction. L'interaction est alors vue comme un couple indissociable d'une action et d'un stimulus. Cette frontière n'est

cependant pas fixe. Elle dépend du niveau d'abstraction et du niveau de détails des interactions. Elle reste ajustable par le concepteur d'agent qui choisit ces niveaux d'abstraction et de détails. Pour modifier le niveau d'abstraction, il est nécessaire d'avoir recours à un système sensori-moteur dans l'agent. En revanche, pour modifier le niveau de détails, il suffit d'utiliser les six processus présentés : fusion, généralisation, spécification, différenciation, diversification et simplification.

L'étude et l'analyse du comportement des agents sont rendues possibles à pendant la simulation et après celle-ci. Lors de la simulation, c'est la participation de l'observateur à la simulation et les indicateurs du tableau de bord qui permettent à celui-ci de suivre, voire de guider l'apprentissage d'un agent. Après la simulation, il reste les traces. Elles permettent une étude plus approfondie et objective de la chronologie des événements, des comportements, or l'apprentissage d'un agent se reflète dans l'évolution de ses comportements.

5.1.2 Retour sur le comparatif des environnements

En reprenant les critères utilisés lors de l'état de l'art, nous pouvons observer que SECA répond bien à l'ensemble de ces critères (Tableau 5.1), conformément à l'implémentation désirée. Il y a cependant une petite exception pour l'analyse de comportements car, actuellement, SECA propose uniquement d'enregistrer les traces des agents. Les traces correspondant aux interactions de l'observateur sur l'environnement font encore défaut.

Critères	Environnements « généralistes »			Environnements « ad hoc »			SECA
	oRis	RoboCode/dTank	Vacuum	Gridland	Environnement de Perotto et al	Environnement de Guerin et al	
Interactivité	+	-	+	-	-	-	+
Immersivité	-	+	+	-	-	-	+
En ligne	-	+	-	-	-	-	+
Multi-agent	+	+	-	+	-	-	+
Ergonomie	--	+	+	+	+	+	+
Evolutivité	++	-	+	-	-	-	+
Interactionniste	-	-	-	+	+	-	+
Analyse de comportements	-	-	-	-	-	-	+/-

Tableau 5.1 : Comparatif des environnements de simulation de l'état de l'art et de SECA

5.2 Perspectives

Bien que l'environnement SECA permette déjà de tester et d'évaluer des agents, il reste encore à implémenter l'enregistrement des traces de l'environnement. En effet, seules les traces des

agents sont actuellement enregistrées donc les actions de l'observateur ne peuvent pas être prises en compte lors de l'étude des traces.

De plus, bien que l'environnement ait été codé pour des simulations en trois dimensions, les agents testés n'évoluaient que dans un plan de la scène, en deux dimensions donc. Il sera intéressant de vérifier le passage à des scènes en trois dimensions utiles.

Un travail d'optimisation du rendu 3D donnerait de meilleures performances et permettrait de créer des scènes de plus grande dimension.

Bibliographie

Brooks, R. A. (1991). Intelligence Without Representation. *Artificial Intelligence 47(2)*, 139-159.

Cañamero, D. (1997). Modeling motivations and emotions as a basis for intelligent behavior. *In Proceedings of the first international conference on Autonomous agents*, 148-155.

Cohen, M. A., Ritter, F. E., & Haynes, S. R. (2005). Herbal: A high-level language and development environment for developing cognitive models in Soar. *Proceedings of the 14th Conference of Behavior Representation in Modeling and Simulation*, 133-140.

De Loor, P., Manac'h, K., & Tisseau, J. (2010). Enaction-Based Artificial Intelligence: Toward Co-evolution with Humans in the Loop. *Minds and Machine, num 19*, 319-343.

Drescher, G. L. (1991). Made-Up Minds, Aconstructivist Approach to Artificial Intelligence. *MIT Press*.

Eisenstein, J. (2003). Evolving Robocode Tank Fighters. *Memo AIM-2003-023, MIT AI Lab*.

Guerin, F., & McKenzie, D. (2008). Apiagetian Model of Early Sensorimotor Development. *Proceedings of the Eighth International Conference on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems*, 29-36.

Harrouet, F. (2000, December). oRis : s'immerger par le langage pour le prototypage d'univers virtuels à base d'entités autonome.

Le Moigne, J.-L. (1995). Les épistémologies constructivistes. Dans J.-L. Le Moigne, *Les épistémologies constructivistes* (pp. 70-74). PUF, "Que sais-je ?".

Li, S. (2002, January 01). *Rock 'em, sock 'em Robocode!* Retrieved July 13, 2011, from IBM Web site: www.ibm.com/developperworks/java/library/j-robocode/

Oudeyer, P.-Y., Kaplan, F., & Hafner, V. V. (2007). Intrinsic Motivation Systems for Autonomous Mental Development. *IEEE Transactions on Evolutionary Computation, Vol. 11*, 265-286.

Perotto, F. S., & Alvarez, L. O. (2006, December 26). Incremental Inductive Learning in a Constructivist Agent. *Proc. of SGAI*, 129-144.

Piaget, J. (1937). *The construction of reality in the child.* New York: Basic Books.

Ritter, F. E., Kase, S. E., & Bhandarkar, D. (2007). dTank Updated: Exploring Moderated Behavior in a Light-weight Synthetic Environment. *Proceedings of the 16th Conference on Behavior Representation in Modeling and Simulation*, 51-60.

Russel, S., & Norvig, P. (1995). *Artificial Intelligence: A Modern Approach.* Englewoods Cliffs: Prentice-Hall.

Suchman, L. A. (1987). *Plans and situated actions.* Cambridge: Cambridge University Press.

Wilson, M. (2002). Six views of embodied cognition. *Psychonomic Bulletin & Review*, 625-636.

Annexe A : Correspondance entre les noms de classes français de SECA et les noms de classes réels en anglais

Noms de classe français	Noms de classe réels
SECA	SECA
Tableau de Bord	DashboardManager
Moteur de simulation	SimEgnine
Moteur de rendu	Renderer
Gestionnaire de sons	SoundManager
Scène	SceneManager
Caméra	Camera
Bloc	Block
Forme	Mesh
Matière	Material
Corps d'un agent	AbstractModel
Animation d'un agent	AbstractAnim
Boite à outils	Line, Quad, Triangle AxisAlignedBox, Plane, PlaneBoundedVolume, Sphere Ray ArcGrid, ArcGridIterator, CircularGrid, CircularGridIterator SecaColor, SecaMath, SecaPrimitiveFactory, SecaTextRenderer Keys Pair, RandomString
Service de traces	XMLStreamTracer
Corps Ernest 7	Ernest7Model
Animation Ernest 7	Ernest7Anim
Corps Ernest 10	Ernest10Model
Animation Ernest 10	Ernest10Anim

Annexe B : Commandes propres à SECA

Commandes de SECA	Actions associées
F1	Afficher uniquement les sommets des objets de la scène
F2	Afficher uniquement les arrêtes des objets de la scène
F3	Afficher les objets de la scène avec leur couleur uniforme
F4	Afficher les objets de la scène avec leur texture
↑ (ou NUM[7] 8)	Déplacer la caméra vers le haut
↓ (ou NUM 2)	Déplacer la caméra vers le bas
← (ou NUM 4)	Déplacer la caméra vers la gauche
→ (ou NUM 6)	Déplacer la caméra vers la droite
NUM 5	Placer la caméra au Sud du point visé, dirigée vers le Nord
Page précédente (⇞)	Zoomer
Page suivante (⇟)	Dézoomer
CTRL + ↑	Translater la caméra vers l'avant
CTRL + ↓	Translater la caméra vers l'arrière
CTRL + ←	Translater la caméra vers la gauche
CTRL + →	Translater la caméra vers la droite
ALT + ↑	Translater le curseur clavier vers le Nord de la scène
ALT + ↓	Translater le curseur clavier vers le Sud de la scène
ALT + ←	Translater le curseur clavier vers l'Ouest de la scène
ALT + →	Translater le curseur clavier vers l'Est de la scène
ALT + Page précédente (⇞)	Translater le curseur clavier vers le haut de la scène
ALT + Page suivante (⇟)	Translater le curseur clavier vers le bas de la scène
Tabulation (↹)	Changer de caméra (la permutation est cyclique)

[7] Pavé numérique