Raisonnement heuristique réflexif dans un agent temps réel

Philippe Caillou
LAMSADE, Université Paris IX Dauphine,
caillou@lamsade.dauphine.fr

Résumé *: L'objectif de ce travail est de proposer un modèle d'agent capable de découvrir de nouveaux concepts et des règles heuristiques adaptés au domaine et de les utiliser efficacement, en particulier dans le cadre de jeux en temps réel. Pour cela, l'agent utilise un modèle de raisonnement fondé sur un grand nombre d'heuristiques utilisées de façon réflexive. L'utilisation d'un graphe pour la représentation des connaissances permet de représenter et d'utiliser de façons similaires les concepts et les règles quelque soit leur niveau d'abstraction. Il n'y a pas de distinction de niveaux métas et les meta-règles peuvent être créées et modifiées comme les autres. L'intégration de ce modèle à un agent nécessite d'étudier les cas particulier des perceptions et des actions dans un environnement temps réel. De plus, l'agent devant être totalement autonome, un système d'émotion permet de le réguler en fonction de son état et de celui de l'environnement.*
Mots-clés *: Agent, Apprentissage, Heuristiques, Jeux temps réel.*

1 Introduction

Lorsqu'il apprend à jouer a un jeu, un être humain conçoit et utilise des concepts de plus en plus abstraits, que ce soit pour percevoir, agir ou réfléchir. Ainsi, aux échecs, des heuristiques de perceptions sont d'abord créées pour identifier rapidement les pièces, puis les ensembles de pièces qui correspondent à des structures connues. C'est pourquoi les champions d'échecs peuvent replacer plus de pièces sur un échiquier après l'avoir observé quelques secondes uniquement si l'emplacement initial correspondait à une partie réelle. De même, au niveau de l'action, des concepts plus abstraits sont créés et utilisés, tels que la menace de pièce. Ces concepts de plus en plus abstraits sont de plus utilisés dans des heuristiques de raisonnements construites progressivement en fonction des situations rencontrées. De telles heuristiques permettent notamment de n'étudier et évaluer que certains coups aux échecs parmi les très nombreux possibles. De même, à Tetris, de nombreuses positions de pièces peuvent ainsi être rapidement éliminées en fonction de la configuration actuelle du jeu [KIR 94].
C'est cette utilisation d'heuristiques et de concepts adaptés au domaine qui a été reprise par AM [LEN 78] puis EURISKO [LEN 83]. AM est un programme permettant de découvrir de nouveaux concepts mathématiques. Ces concepts sont obtenus par application d'heuristiques de recherche et d'évaluation de concepts. Il a par exemple

redécouvert des concepts tels que les nombres premiers ou la multiplication. EURISKO a étendu le principe de l'utilisation d'heuristiques de recherche et d'évaluation en permettant de les appliquer aux heuristiques elles-mêmes. En plus de chercher des concepts intéressants et adaptés au domaine (dans le cas d'EURISKO, un jeu de société où il faut constituer une flotte de vaisseaux spatiaux), le programme cherche donc des heuristiques adaptées (comme « intégrer un vaisseau rapide et peu armé dans la flotte »). De plus, comme cette recherche se fait à l'aide d'heuristiques, celles-ci peuvent également être modifiées et améliorées. De même, les types d'attributs et de relations entre concepts (intérêt, est-un, …) ne sont plus fixes comme dans AM, mais également modifiables par des heuristiques.
Un intérêt d'AM et d'EURISKO est de montrer la faisabilité et l'intérêt de l'utilisation d'heuristiques de recherche et d'évaluation de concepts, en particulier lorsque ces heuristiques peuvent s'appliquer à d'autres heuristiques. La notion d'heuristique qui sera utilisée ici est celle correspondant à la définition donnée par Lenat [LEN 82] simple « règle de jugement informelle ».
Pour pouvoir être efficace, un système utilisant des heuristiques à la fois pour agir et apprendre doit nécessairement être en partie réflexif. En effet, si le système crée un nouveau concept C adapté au domaine, il doit être capable de créer de nouvelles heuristiques adaptées, voir même spécifiques, à C. Si ce n'est pas le cas, le système pourra toujours continuer à découvrir des concepts de plus en plus intéressants (comme dans le cas de AM), mais il ne pourra découvrir de nouvelles heuristiques, en particulier pour avoir un comportement plus efficace.
Une de limites reprochée à AM et EURISKO est que le choix des concepts intéressants se fait en partie par des heuristiques d'évaluation, mais également en majorité par l'intervention de l'utilisateur. Un autre programme utilise un mécanisme de raisonnement similaire mais sans intervention de l'utilisateur. Dans CopyCat [MIT 93], et son successeur MetaCat [MAR 02] l'utilisateur n'intervient pas puisque l'objectif est de pouvoir obtenir plusieurs résultats différents lors de plusieurs utilisations différentes. Le programme permet de simuler le raisonnement analogique humain sur des problèmes de type abc->abd alors xyz-> ? Pour y arriver, le système exécute un grand nombre d'heuristiques simples (ou coglets) qui créent, évaluent, confirment et détruisent des liens et des groupes entre les concepts de départ (constitués par les lettres contenues dans l'énoncé du problème). L'efficacité et la qualité du résultat proviennent de l'ordre, et donc du choix, des coglets appliqués. Ce choix est fait grâce à l'activation de concepts tels que longueur, inverse, successeur, 2 ou b présents dans un graphe de concepts fluides. Les coglets liés aux concepts activés ont plus de chance d'être exécutés (directement, ou parcequ'une priorité plus importante leur a été attribuée lors de leur création) ou choisis (lorsqu'il y a besoin de choisir un paramètre à tester). Ce système n'est pas réflexif (les coglets ne peuvent se modifier entre eux) car l'objectif n'est pas l'apprentissage, mais simplement l'analogie. La méthode employée reste toutefois l'application de nombreuses heuristiques sur un graphe de concepts. Un point particulièrement intéressant du système concerne le monitorage global du système, du fait que l'utilisateur n'intervient jamais. Ce monitorage se fait grâce à la température, qui ne décrit pas un objectif précis, mais plutôt un état du système : plus les concepts de départ sont utilisés de façon cohérente, plus la

température est élevé. Ce niveau de température va modifier le choix et l'action des coglets, donc le comportement de l'agent.
Ces deux systèmes montrent l'intérêt et l'efficacité de programmes raisonnant et découvrant de nouveaux concepts adaptés aux domaines à l'aide de nombreuses heuristiques, et permettant de découvrir de nouvelles heuristiques grâce à la réflexivité. Ces systèmes permettent une certaine créativité dans la mesure où le résultat obtenu n'est pas toujours le même et une grande adaptabilité grâce à l'utilisation de concepts et d'heuristiques adaptées au domaine.
Un autre point commun entre CopyCat et EURISKO est qu'ils portent sur des problèmes ne nécessitant pas d'interaction avec un environnement. Or, ces avantages d'adaptabilité et de créativité par rapport au domaine semblent particulièrement intéressant à étendre au domaine des agents. Nous pouvons définir un agent intelligent comme un système informatique situé dans un environnement et qui est capable d'agir de façon autonome et flexible sur cet environnement pour atteindre ses objectifs prédéfinis [WOO 99]. La particularité la plus importante d'un agent par rapport à un programme traditionnel est d'être situé dans un environnement. En plus du contrôle implicite réalisé par les émotions, l'intégration d'un modèle de raisonnement heuristique dans un agent doit donc prendre en compte les perceptions, actions et éventuellement le comportement en temps réel de l'agent.
La combinaison de la découverte de concepts adaptés au domaine avec les perceptions et actions d'un agent offrent de nombreuses perspectives mais posent un problème d'intégration. En effet, l'utilisation de concepts et de règles de plus en plus abstraites et adaptés au domaine peut permettre à l'agent de percevoir et d'agir avec des concepts et des règles de plus en plus complexes. Il peut donc, tout en utilisant des règles de perceptions simples, percevoir et agir rapidement de façon complexe. Par exemple, pour un agent jouant à Tetris, l'acquisition des concepts de pièce puis de barre, de S et de L, vont lui permettre de raisonner directement sur ces perceptions. La recherche du nouveau concept perceptif peut ensuite se faire de façon proactive (l'agent se demande si ce concept est perçu ou non et applique une règle d'identification pour le savoir) mais aussi réactive (la combinaison de stimuli simples ressentis généralement lorsque le concept étudié est présent provoque automatiquement l'exécution de la règle d'identification du concept). De même, au niveau des actions, l'apprentissage d'actions plus complexes comme le placement d'une pièce à un endroit donné doivent pouvoir être intégrés directement dans une règle. La combinaison de ces perceptions, actions et apprentissages de concepts abstraits doivent lui permettre de construire et d'exécuter une règle telle que « Si une barre arrive et qu'il existe un canyon dans le jeu, placer la barre dans le canyon ».
Ces exemples montrent l'intérêt de l'intégration d'un tel système de raisonnement et d'apprentissage heuristique dans un agent. Le problème de cette intégration vient du fait que le système de raisonnement heuristique est symbolique, les connaissances sont décrites explicitement. Les perceptions et les actions, elles sont des données reçues ou envoyées en continu à l'environnement. Comment peuvent se réaliser la mise à jour et la réactivité à une perception ou la continuité de l'action ? Par ailleurs, un agent possédant ce type de raisonnement ne peut pas avoir un objectif fixe et explicitement défini sous forme de connaissances, car il doit pouvoir faire évoluer ses buts en

fonction des concepts et règles qu'il apprend. Pour que l'agent soit autonome et que l'évaluation des concepts créés ne nécessite pas d'intervention humaine comme dans Eurisko, nous avons généralisé le concept de température utilisé dans CopyCat pour guider progressivement les règles choisies par l'agent en fonction d'une observation de l'état du système dans son ensemble. Du fait de la similarité fonctionnelle, nous appellerons ses fonctions de monitorage émotions. En effet, comme les émotions humaines, les émotions seront ici des fonctions qui en fonction de l'état de l'environnement et des connaissances de l'agent, modifieront le comportement et influenceront les choix de l'agent.
Comment adapter le modèle de raisonnement heuristique réflexif pour pouvoir percevoir, réfléchir et agir en temps réel ? Après avoir donné un état de l'art rapide des systèmes proposant une problématique proche de celle-ci (section 2), nous commençons par décrire l'architecture globale proposée et le système de réflexion proprement dit (section 3). Puis nous décrivons le contrôle réalisé grâce aux émotions (section 4). Nous expliquons en section 5 comment nous intégrons les perceptions et les actions. L'implémentation réalisée est présentée en section 6. Enfin, nous concluons en section 7.

2 Etat de l'art

2.1 Raisonnement heuristique

Nous regroupons les systèmes de raisonnement d'AM, EURISKO et CopyCat sous le terme de raisonnement heuristique du fait de leurs points communs :
- Une utilisation successive de nombreuses heuristiques sur un graphe de concepts
Du fait du nombre croissant d'heuristiques et de concepts, le choix de la bonne heuristique et des concepts pertinents est particulièrement important.
- Une absence de contrôle centralisé : les heuristiques et concepts sont tous au même niveau, qu'ils soient nouvellement créés ou présents à la création du système
Ce fonctionnement peut se rapprocher du fonctionnement du cerveau humain vu sous forme d'homoncules qui agissent indépendamment pour arriver finalement à une conscience globale émergente [DEN 91].

2.2 Apprentissage déductif

Les systèmes d'apprentissage déductifs à base d'exemples (EBL), tels que Prodigy [MIN 89], ont pour objectif d'enrichir leur base de règles en expliquant la réussite ou l'échec d'un exemple à partir de leur représentation du monde. Cet objectif est différent du nôtre car il consiste à chercher des règles vraies, prouvées qui puissent être réutilisées par la suite. L'apprentissage implicite de règles d'implication par introspection rappelle le « chunking » de Soar [NEW 90]. Si la méthode est proche, l'objectif diffère puisque l'objectif de Soar, comme pour les systèmes d'EBL, est d'obtenir une règle vraie qui permette de résoudre une généralisation du sous-problème considéré.
Des systèmes réflexifs, tels que MACISTE [PIT 96], permettent une auto-modification

et une auto-analyse. L'objectif est toutefois différent du notre car il s'agit de systèmes logiques dont le but est de déduire des règles logiques dans un système complètement réflexif sans environnement variable ni temps réel.

2.3 Jeux et heuristiques

Les nombreux avantages des heuristiques font qu'elles sont présentes dans de nombreux modèles, en particulier de plus en plus dans le domaine du jeu qui sera notre application. En effet, dans les programmes actuels, du fait de la complexité croissante des jeux étudiés, le nombre de coups possibles à chaque tour et le nombre de tours qu'il est nécessaire d'étudier pour arriver à un coup correct sont de plus en plus élevés et rendent une recherche arborescente simple irréalisable. Des heuristiques sont donc utilisées pour limiter la recherche arborescente. Ainsi, au Go, le très grand nombre d'intersections libres pour jouer (19x19 au début du jeu) conduit à réduire la recherche du meilleur coup en calculant à l'avance un grand nombre de résultats de sous-arbres qui apparaissent dans des situations données [CAZ 96].
De même à Sokoban, l'utilisation d'heuristiques fondées sur la connaissance du jeu par le programmeur, comme par exemple le fait de savoir que le fait de pousser une caisse au début d'un tunnel équivaut au niveau résultat à la pousser jusqu'au bout, permet de réduire considérablement la profondeur de la recherche [JUN 01]. Ces heuristiques sont alors données par le programmeur grâce à son expertise.

2.4 Agents et contrôle heuristique

La méthode de contrôle décentralisée des actions utilisée cii peut être rapprochée de l'architecture de subsomption (des règles sont en concurrence et s'appliquent simultanément pour arriver à un résultat) [BRO 99]. L'intérêt commun avec cette architecture est que d'utiliser des règles simples pour un résultat complexe et efficace. Le modèle proposé ici permet toutefois un monitorage et une modification réflexive des concepts, de même qu'un raisonnement global de l'agent.

2.5 Emotions et contrôle

L'étude des émotions s'est récemment fortement développée en intelligence artificielle [PET 01]. Elles peuvent être utilisées de deux façons.
La première utilisation consiste à essayer de détecter les émotions humaines pour faciliter la communication et la compréhension lors des interactions homme-machine [PIT 97]. L'identification et la restitution d'émotions permettent en effet d'interpréter ce que l'utilisateur désire et de réagir en conséquence. Cette problématique a été en particulier popularisée par le robot Kismet du M.I.T. [BRE 00].
Une deuxième façon d'aborder le problème est de s'intéresser aux fonctions internes des émotions et à leur fonctionnement, c'est-à-dire d'étudier et de reproduire leur influence sur la cognition et le raisonnement humain. Moins développée, cette approche s'appuie en particulier sur les travaux réalisés en neurobiologie par A. Damasio [DAM 95], [DAM 99]. Cette approche a notamment été utilisée par [VEL 97] et [VEL 98]. Dans ces systèmes, l'utilisation d'émotions permet d'orienter en biaisant le

comportement d'agents logiciels ou de robots alors même que le contrôle se fait de façon décentralisée dans une architecture proche de la subsomption. C'est cette approche que nous allons adopter, en utilisant les émotions comme guide, à la fois pour le comportement, et pour l'apprentissage

3 Présentation générale

EURISKO et CopyCat ont présentés et utilisés un modèle de raisonnement permettant de créer et modifier des concepts et heuristiques adaptés au domaine et de les contrôler dans le cas d'un problème abstrait. Des heuristiques telles que celles découvertes ont été utilisées avec succès pour accélérer la résolution de problèmes et de jeux. Comment intégrer un tel mécanisme de raisonnement réflexif dans un agent agissant en temps réel, percevant et agissant face à un environnement ? Comment intégrer un modèle de raisonnement heuristique réflexif dans un agent ?

3.1 Exemple simple

Pour illustrer le système. On considère un environnement constitué de n lumières et n boutons situés au même niveau. Il possède également un bouton pour démarrer la partie et une lumière indiquant son état actuel. L'agent perçoit une lumière et un bouton à la fois, plus la position de la lumière actuellement observée, le score actuel et l'état de la partie. Le but du jeu est de trouver deux lumières allumées successives et d'appuyer le plus rapidement possible sur les boutons situés en face.

3.2 Architecture globale

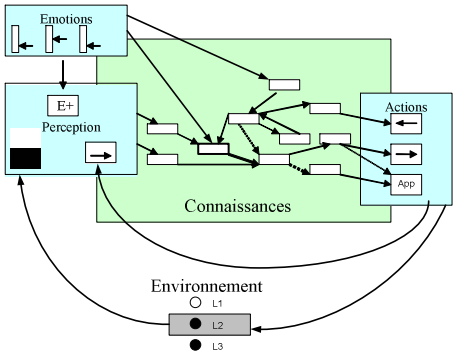

FIG. 1 Schéma global de l'agent

Les connaissances de l'agent sont présentes dans un graphe de connaissances (décrit en 3.3). Les données et les règles y sont représentés et utilisés. Le raisonnement se fait grâce à l'exécution de ces règles. Celles-ci sont exécutées avec une probabilité fonction de leur degré d'activation à chaque instant, comme décrit en 3.4.
Par définition, l'agent se situe dans un environnement. Ses sens (décrits en section 4.1) perçoivent cet environnement et transmettent l'information dans le graphe. Ses actions (décrites en section 4.2) lui permettent d'agir sur l'environnement et sont activées en

fonction de l'état de certaines connaissances du graphe. Elles sont de plus perçues par l'agent qui sait ainsi directement ce qu'il est en train de faire. Pour guider l'agent, des émotions (décrites en section 5) modifient le graphe en fonction de l'état de l'environnement et du graphe lui-même. Ses émotions sont elles aussi perçues par l'agent, ce qui lui permettra, en se les remémorant, de favoriser les situations ayant été perçues comme positives.

3.3 Représentation des connaissances

Le modèle de représentation des connaissances choisi doit à la fois permettre la réflexivité des règles (des règles à la fois exécutables et modifiables par d'autres règles) et l'efficacité du choix des concepts utilisés. Le modèle de graphe orienté d'activation, inspiré des graphes de concepts fluides de CopyCat, permet de répondre à ce double problème. Pour permettre un traitement et une modification des règles, celles-ci sont représentées de façons explicites par plusieurs connaissances liées entre elles. Chaque partie ou fonction d'une règle peut ainsi être analysée et modifiée indépendamment ou par rapport aux autres. Les liens du graphe sont non typés car les relations doivent pouvoir être analysées et modifiées comme les autres concepts.
Une connaissance est un nœud du graphe. Il s'agit d'un atome de savoir. Un agent ne peut ni la diviser ni regarder dans une connaissance. A tout moment, une connaissance a un degré d'activation qui correspond au degré d'attention que l'agent lui porte et qu'elle va transmettre à ses connaissances voisines.
Les liens sont avant tout fonctionnels, ils servent à transmettre de l'activation entre deux connaissances. Un lien a une intensité, correspondant à son degré de vérité (proche de la distance entre deux concepts de CopyCat). Cette valeur sert à calculer la part d'activation transmise à la connaissance-cible. Une valeur négative signifie un effet inhibiteur sur l'activation de la connaissance-cible.

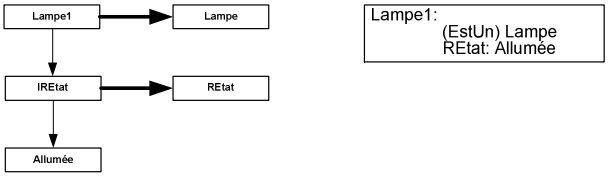

FIG. 2 Concept Lampe1 dans le graphe et son interprétation sous forme de Frame

On peut associer à chaque connaissance un concept. Il correspond à la connaissance de départ plus l'ensemble des connaissances qui lui sont liées avec un minimum d'intensité. Pour simplifier la représentation, ce concept peut être vu comme une Frame, les connaissances liées constituant les attributs de la connaissance de départ.
Lorsque la connaissance cœur d'un concept est activée, elle va transmettre son activation à toutes les connaissances qui seront dans ce concept et ce de façon proportionnelle à la force du lien.
Ainsi, un concept C1 sous-type ou élément d'un autre concept C2 (par exemple *Lampe1* est un sous-type de *Lampe*) sera lié fortement au père mais pas inversement. Les attributs du père appartiendront alors au concept fils, alors que les attributs du fils

n'appartiendront pas au concept père. On peut interpréter une liaison forte comme un attribut de type *EstUn* implicite Les attributs d'un autre type, tels l'état allumé d'une lampe, sont représentés grâce à des connaissances intermédiaires liées à la fois à la connaissance correspondant au type de lien (*REtat,* pour Relation Etat) et à la connaissance cible du paramètre (*Allumée*).
Les règles sont des concepts comme les autres, représentés de façon explicite avec un sous-graphe de connaissance.

3.4 Fonctionnement

Le fonctionnement du système est fondé sur deux principes :
Le premier principe de base du fonctionnement est qu'un grand nombre de règles sont appliquées à la suite pour simuler un parallélisme. L'avantage de ce système est que de nombreuses règles peuvent échouer sans conséquences importantes, voir être en désaccord entre elles. Dans ce cas, les plus nombreuses ou les plus activées imposent leur point de vue et constituent la tendance. Il faut noter qu'aucune vérification automatique de cohérence des règles entre elles et de validité des règles n'est effectuée. Bien que certaines règles de vérification puissent être ajoutées, des règles incohérentes et en partie fausses peuvent exister dans le graphe. Cet état ne nuit pas obligatoirement à l'efficacité du système. En effet, les règles, en partie fausses ou incohérentes, peuvent se révéler très efficaces du fait de leur simplicité si elles sont appliquées au bon moment. Une autre conséquence de l'existence possible de telles règles est qu'il n'y a aucune différence de représentation entre une règle, une règle floue ou une heuristique. Dans la suite de l'article, la notion de « règle » correspondra donc à la définition la plus générale du terme qui comprend tout type de règle, quelque soit son degré de vérité. Les règles ainsi exécutées sont celles présentes dans le graphe et s'appliquent au graphe lui-même. Des règles peuvent être modifiées, créées et supprimées par d'autres règles à tout moment. Elles ne peuvent donc être compilées à l'avance et leur programme complet est reconstruit à chaque exécution. Le choix de la règle suivante se fait de façon probabiliste. Plus une règle est activée, plus elle a de chance d'être exécutée.
Le deuxième principe de base pour choisir les règles et les concepts utilisés est la transmission de l'activation. Le principe de la transmission de l'activation est que les connaissances très liées soient activées simultanément. Ainsi plus une connaissance est activée, plus les connaissances qui lui sont liées, donc d'abord ses propriétés et ses composants, sont activées à leur tour. A chaque période, chaque connaissance modifie le degré d'activation de chacune des connaissances à laquelle elle est liée proportionnellement à son propre niveau d'activation et à la valeur d'intensité du lien. Dans un deuxième temps, chaque connaissance diminue son propre niveau d'activation en appliquant un facteur d'actualisation. De l'activation est introduite dans le graphe de deux façons: par les émotions et par les perceptions.
Cette méthode a pour avantage de permettre le raisonnement implicite. En effet, l'apprentissage des liens permet d'activer les bonnes connaissances au bon moment, que ce soit les règles ou les concepts, et ce simplement par transmission de l'activation, sans interprétation de règle explicite.

Lorsque l'agent doit réaliser un choix au hasard dans le graphe, par exemple choisir une règle, un exemple de concept ou un élément dans un ensemble, le choix se fait de façon probabiliste en fonction de l'activation des connaissances. Cette méthode de transmission de l'activation et de choix des règles et des concepts utilisés permet de simuler des états mentaux de l'agent. L'activation de certains concepts entraîne l'activation, et donc l'utilisation, des concepts qui lui sont proches (proche ne signifie pas similaire. Deux concepts peuvent être très proche parcequ'ils sont liés par une relation d'opposition, comme Avant et Apres, ou Allumé et Eteint).

3.5 Raisonnement

L'agent exécute un grand nombre de règles successivement. Elles portent à la fois sur ses perceptions, ses actions et ses connaissances. La sémantique des concepts de l'agent et des règles n'est pas imposée par le modèle de représentation. L'agent peut d'ailleurs par la suite modifier lui-même cette sémantique. Les règles et concepts implémentés au départ dépendent donc des fonctions souhaitées pour l'agent par le concepteur. Ici, l'objectif est d'avoir un agent capable de découvrir de nouveaux concepts et règles sur un jeu et d'y jouer le mieux possible.
Un premier ensemble de concepts inclus dans l'agent concerne donc la découverte de nouveaux concepts. Ces règles sont très proche de celles d'Eurisko puisque l'objectif est le même, réaliser des essais, chercher de nouveaux concepts et règles a partir de ceux existant, chercher des exemples, les évaluer. La méthode de recherche proposée au départ à l'agent reprend la méthode utilisée à la fois par Eurisko et par CopyCat : Proposition, Evaluation, Confirmation (Par exemple Si un grand nombre d'exemples d'un concept C ont un attribut commun A de valeur V, proposer un nouveau concept défini comme un C avec un attribut A de valeur V ; Si un concept proposé C est un sous-concept de C' et que C' est très intéressant, alors augmenter l'intérêt de C proportionnellement à celui de C').
Pour raisonner de façon cohérente, l'agent a également besoin de règles gérant un système de suivi d'objectif. Cette tâche est assurée dans AM et CopyCat par un agenda global, dans Eurisko par 3 niveaux d'objectifs (global-tache-concept). Pour ne pas avoir à identifier de tels niveaux tout en gardant l'avantage de la hiérarchie, le système utilisé ici est une généralisation des niveaux utilisé dans Eurisko. Les règles peuvent proposer des objectifs au niveau global ou en sous-objectif d'un objectif en cours de réalisation, créant implicitement un sous-niveau.
Pour gérer le temps, l'agent dispose de plus d'une fonction de « mise en attente » de règles. Si une action d'une règle réclame d'attendre un évènement donné avant de continuer, cette règle est mise en attente jusqu'à ce que la condition de fin soit réalisée ou qu'elle soit supprimée de la liste des règles en attente par une autre règle.

4 Perceptions et actions

4.1 Perceptions

L'objectif du système perceptif est de permettre à tout type de sens d'introduire de

l'information dans le graphe et que l'agent puisse utiliser cette information. Pour être efficace, l'agent doit de plus être capable d'agir de façon réactive (la perception modifie les connaissances de l'agent automatiquement) et proactive (l'agent consulte l'état de ses perceptions).

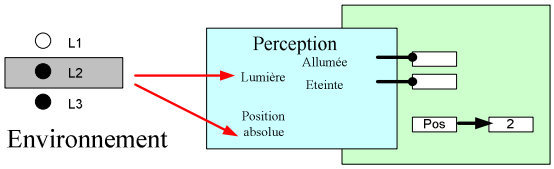

FIG. 3 Perception de l'état Allumée/Eteinte d'une lumière et de la position observée

Pour cela, les perceptions (ou sens) activent des connaissances spécifiques dans le graphe et y introduisent ainsi un certain degré d'activation. Cette activation peut être utilisée de deux façons. De façon réactive, l'activation introduite dans le graphe est transmise à d'autres connaissances, permettant un raisonnement implicite à partir de ces perceptions et activant éventuellement des concepts plus abstraits que la perception brute qui pourront ensuite être utilisés explicitement. De façon proactive, une règle explicite peut lire la valeur de l'activation de la connaissance perçue (ou d'une autre connaissance décrivant un concept plus abstrait activé) pour pouvoir l'interpréter et l'utiliser. Par exemple, une perception binaire, comme l'état d'une lampe, sera liée à deux connaissances, CoAllumée et CoEteinte. Lorsque la lumière observée est allumée, CoAllumée est activée mais pas CoEteinte, et inversement si la lumière est éteinte. Cette information, présente dans le graphe de connaissances puisque CoAllumée et CoEteinte sont présentes dans ce graphe, peuvent être utilisée de façon proactive ou réactive :

De façon proactive, une règle peut comparer l'état des deux connaissances pour connaître l'état actuel de la lampe. Ainsi, la règle de mise à jour de l'état de la lampe, donnée lors de sa création à l'agent afin qu'il puisse utiliser explicitement l'information perçue, est définie par :

Si Activation(CoAllumée)>Activation(CoEteinte) alors Lumière -> REtat = Allumée
sinon Lumière -> REtat = Eteinte

De façon réactive, une des connaissances peut être liée à une règle ou à n'importe quel autre concept qu'elle activera automatiquement lorsqu'elle sera elle-même activée.

Pour une perception plus complexe, où un grand nombre, voir un nombre infini, de stimuli différents doivent pouvoir être perçu, il peut être impossible d'associer une connaissance à chaque cas possible. Le système perceptif peut alors soit créer une nouvelle connaissance correspondant à la valeur perçue et l'inclure directement en paramètre explicite de la connaissance correspondant à la perception. Ou bien il peut lier explicitement le concept décrivant la situation de la perception au concept correspondant à ce qui est perçu si celui-ci existe déjà dans le graphe.

Par exemple, pour percevoir la position actuelle, on peut ne pas souhaiter rentrer une connaissance par position possible dans le graphe. Dans ce cas, il y a deux possibilités : Soit, à chaque fois que la position change, le système perceptif crée une nouvelle connaissance correspondant à la nouvelle position et l'ajoute en attribut explicite du

concept de position actuelle. Soit, si le nombre correspondant à la nouvelle position existe déjà dans le graphe, on ajoute cette connaissance-nombre en attribut explicite du concept de position actuelle.
Des règles peuvent également être données au départ pour décrire des perceptions plus complexes, et l'agent peut créer lui-même des concepts plus abstraits et les règles d'identification associées. Par exemple, le concept simple de LumiereAllumée défini par la fonction : Une Lumière L est une LumièreAllumée si L->REtat = Allumée. L'interet de ce genre de concepts, même très simples, est de pouvoir raisonner et créer des règles en les utilisant directement plutôt que de rajouter une condition et utiliser le concept père. De plus, ils peuvent avoir leurs caractéristiques propres et être adaptés pour produire progressivement des concepts plus complexe (comme un CoupledeLumièresAllumée pour notre exemple, qui peut être utilisé directement pour atteindre l'objectif du jeu).
L'agent dispose également d'un sens particulier lui permettant d'observer les concepts qui ont suivi la plus forte variation d'activation au cours de la période précédente. Ce concept d'introspection crée une nouvelle connaissance qui est liée à tous ces concepts ayant été fortement activé, permettant à l'agent de savoir ce qu'il vient de faire et penser. Ce sens lui permet également de percevoir les émotions qu'il vient de ressentir, pour pouvoir chercher à les reproduire plus tard, consciemment (par des règles explicites) ou inconsciemment (par réactivation automatique).

4.2 Actions

Symétriquement aux perceptions, une action effectue sa fonction si l'activation de la connaissance particulière impliquant cette action est supérieure à celle inhibant l'action. Cela permet à l'agent d'agir sur l'environnement. L'agent peut avoir connaissance de la réalisation d'une action soit par introspection en analysant la variation des connaissances liées à l'action, soit en percevant les changements consécutifs à son action dans l'environnement, soit enfin grâce à un sens spécifique à cette action.Par exemple, l'action de presser/relâcher un bouton a deux connaissances associées : CoPresser et CoRelacher. Il existe de plus un sens associé au bouton qui lui permet de connaître sa situation actuelle au travers de deux connaissances CoSePressé et CoSeRelaché.

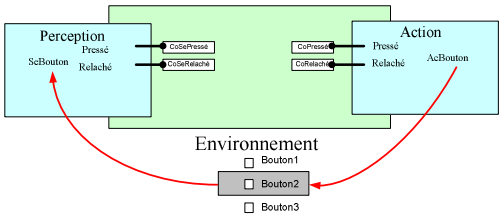

FIG. 4 Action de presser sur un bouton et perception directe de cette action

Une règle est associée à l'action pour que l'agent puisse l'utiliser explicitement dans son raisonnement. Elle est définie de la façon suivante:

PresserBouton : Activer CoPressé jusqu'à ce que Bouton->etat=pressé
Cette règle permet d'activer la connaissance provoquant l'action. L'attente jusqu'à ce que l'état du bouton soit perçu comme pressé implique que la règle est placée en attente jusqu'à ce que la condition soit remplie, ou qu'elle soit retirée de l'ensemble des règles en cours. On peut remarquer ici qu'une autre règle peut simultanément inhiber la connaissance activée CoPressé ou au contraire activer CoRelaché, provoquant dans les deux cas l'effet inverse de PresserBouton. L'action sera alors le résultat de plusieurs règles concurrentes. Celles qui seront le plus activées décideront indirectement de l'action menée car c'est la connaissance qu'elles stimuleront qui sera la plus activée.
Des règles d'action plus complexes peuvent être fournies à l'agent et il peut en concevoir lui-même. En particulier, l'action consistant à appuyer sur le bouton, c'est-à-dire le presser et le relâcher, est simplement décrite par la règle :
AppuyerBouton : PresserBouton puis RelacherBouton

5 Emotions

Comment savoir si l'agent va activer les bonnes connaissances ? On veut qu'il favorise certains comportements dans certaines circonstances. Comme on ne peut pas décrire les connaissances qu'on veut favoriser (les connaissances variant constamment), on définit des facteurs d'évaluation portant soit sur l'extérieur (l'environnement) soit sur le fonctionnement de l'agent. Ces facteurs vont influencer le comportement de l'agent de deux façons : directement par modification de ses paramètres (comme par le renforcement de liens entre les connaissances) et indirectement par la perception qu'il a de ces facteurs. Pour simplifier et pour leur similitude fonctionnelle, on appelle ces fonctions émotions. Ainsi, si on veut un agent curieux, on va ajouter une émotion (positive) curiosité. L'activation de cette émotion augmentera lorsque l'agent créera de nombreuses connaissances (il acquiert ainsi de nouvelles informations). Les liaisons entre les connaissances activées précédemment (y compris celles créées par introspection) seront alors renforcées ce qui conduira l'agent à reproduire son comportement de façon implicite. De façon explicite, l'agent percevra cette augmentation de l'activation de l'émotion et fera un rapport, il pourra alors apprendre que la méthode qu'il a utilisée conduit à une émotion positive et il cherchera à la reproduire.
Un exemple plus simple, mais aussi plus spécifique à l'application à un jeu, est l'émotion relative au score. Cette émotion (que nous appellerons compétitivité) augmente d'autant plus que le score croît. Elle est alors perçue par introspection et associée aux perceptions, actions et concepts activés juste avant que le score n'augmente. Lorsque plusieurs de ces concepts seront à nouveaux activés simultanément, donc lorsque la même situation se reproduira, le souvenir de l'émotion activera les autres concepts associés. Si l'agent choisi un concept ou une stratégie au hasard parmi celles disponibles à ce moment, il aura plus de chance de sélectionner la même que lorsque le score avait augmenté la première fois, car elle aura bénéficié d'une activation supplémentaire par rapport aux autres.

6 Implémentation

Pour tester le modèle proposé, un agent a été implémenté en Java. Cette implémentation a conduit à réaliser des choix de développement qui influencent le fonctionnement du modèle et que nous décrivons ici.

6.1 Cycle de fonctionnement de l'agent

L'agent exécute un cycle d'activité de façon régulière. La périodicité de ce cycle est fixe. Les différentes activités réalisées lors de chaque période sont : Transmission de l'activation, Execution des règles, Perception et Action, Emotions, Oubli.
La dernière étape, l'oubli de connaissances, est une fonction fondamentale pour que le système n'explose pas rapidement. On peut distinguer deux formes d'oubli : suppression des connaissances qui ne pourront plus servir (aucun lien ne permettra de les réactiver dans l'avenir) et dégradation des liens qui ont un poids faible (les liens peu utilisés et renforcés sont progressivement oubliés pour finalement disparaître)

6.2 Représentation et interprétation des règles explicites

Les règles doivent pouvoir être utilisées, analysées et modifiées. Pour cela, les différents composants d'un concept doivent être décrits de façon explicite. La règle peut utiliser des paramètres variables, tels que « un concept ». Pour pouvoir différencier les variables d'une même règle, il n'existe qu'une seule connaissance par variable dans la représentation sous forme de graphe, même si elle est utilisée plusieurs fois dans la règle.
Comme une règle peut être modifiée constamment par d'autre règle, une compilation à la création de la règle est impossible, il faudrait réaliser une reprogrammation dynamique à chacun de ses changements. Pour éviter cette tache complexe et lente, l'exécution de la règle se fait en appelant de façon dynamique chacun de ses sous-composants, chaque sous-composant se chargeant d'appeler ses sous-composants et ainsi de suite.
Les connaissances directement exécutables constituent les briques de bases des règles implicites. Ce sont les fonctions de bases intégrées à l'agent qui effectuent les opérations élémentaires telles que lire un attribut, créer une connaissance, choisir une variable d'un certain type, … Ces fonctions intégrées à l'agent peuvent également correspondrent à des fonctions implémentées directement en java pour être plus efficace. Ce type de fonction a l'avantage d'être plus rapide et plus libre qu'une fonction décrite de façon explicite. L'agent ne pourra bien sur ni l'analyser, ni la modifier, mais il pourra l'utiliser et analyser ses résultats. La réflexivité complète n'étant pas l'objectif du système (qui est de trouver et d'utiliser efficacement des concepts et heuristiques adaptés au domaine), l'utilisation de telles fonctions permet de doter l'agent de fonctions efficace dans des domaines que l'utilisateur ne souhaite pas obligatoirement le voir explorer en profondeur.
L'agent peut également créer ses propres méthodes explicites (test, action, …). L'interprétation d'une règle explicite qui n'est pas composé de fonctions intégrées à l'agent fonctionne alors de façon similaire au cas précédent. L'exécution de la

méthode explicite est réalisée en exécutant les composants de sa définition.

6.3 Transmission de l'activation

Le principe de la transmission de l'activation est que les connaissances très liées soient activées simultanément. Ainsi plus une connaissance est activée, plus les connaissances qui lui sont liées, donc d'abord ses propriétés et ses composants, sont activées à leur tour. La fonction de calcul de l'activation d'une connaissance i en t+1 est :

$$a_i^{t+1} = 1 - \left(1 - a_i^t\right)^{-\beta} \prod_j \left(1 - a_j^t\right)^{\alpha \frac{v_{ji}^t}{\sum_k v_{jk}^t}}$$

Cette équation peut s'interpréter de la façon suivante : pour avoir l'activation $a^{t+1}{}_i$ d'une connaissance i en t+1 : Actualiser la valeur de a_i^t par un facteur β pour prendre en compte le temps. Puis, pour chaque connaissance j qui a un lien allant de j vers i avec une intensité $v_{i,j}$, diviser cette intensité par le total des intensité des liens partant de j ($\sum_k v_{jk}^t$) (pour pouvoir contrôler l'activation totale du graphe). Transmettre une part de l'activation de j a_j^t pondéré par cette intensité et par un facteur α. La forme de cette équation s'explique simplement par les avantages qu'elle procure.
En effet, elle permet à l'activation d'être conservée entre 0 et 1, avec un nombre de liens pointant vers la connaissance, et donc le nombre d'augmentation qu'elle peut subir à chaque période, qui reste libre. Un autre avantage de cette fonction est que la variation de la valeur de l'activation totale contenue dans le graphe de connaissance peut être connue. Cette valeur ne dépend que du degré d'actualisation (β) et du taux de transmission de l'activation entre les connaissances (α).

7 Conclusion et perspectives

L'objectif de ce travail est de proposer un modèle d'agent permettant de découvrir et d'utiliser efficacement des concepts adaptés au domaine dans un graphe de connaissance en l'appliquant particulièrement aux jeux en temps réels. Pour cela, nous avons proposé d'utiliser un graphe d'activation orienté pour représenter les connaissances, ce qui permet de traiter tous les types de connaissances au même niveau. La méthode de raisonnement choisi est le raisonnement heuristique déjà utilisé en particulier dans Eurisko et CopyCat. Après avoir décrit le fonctionnement d'un agent utilisant une telle représentation des connaissances, nous avons montré comment il était possible d'intégrer les spécificités propres aux agents dans ce modèle de raisonnement. L'implémentation réalisée est toujours en cours de test, l'objectif étant de l'appliquer a des jeux de plus en plus complexes. Dans un premier temps, le jeu de Tetris nous semble constituer un bon exemple de jeu temps réel où cette approche pourrait se révéler intéressante. Une partie du travail théorique non développé ici porte sur la capacité d'apprentissage de règles implicites dans ce modèle en utilisant l'analyse de données symboliques sur les données issues de l'introspection. L'étude des similarités entre les comportements passés de l'agent devrait lui permettre de créer automatiquement de nouveaux liens et de reconstituer les résultats précédents.

De nombreuses autres perspectives d'extension théoriques restent disponibles : étudier les connaissances nécessaire pour intégrer la communications avec d'autres agents semble la première et la plus logique dans un système multi-agents. Cela permettrait par la suite d'étendre l'application du modèle à des jeux à plusieurs joueurs qui interagissent entre eux.

8 Bibliographie

Breazeal C., Sociable Machines: Expressive Social Exchange Between Humans and Robots, Sc.D. Dissertation, M.I.T. Dept. of EECS, 2000

Brooks R. A., Cambrian Intelligence. MIT Press ed. 1999.

Cazenave T., Système d'Apprentissage par Auto-Observation, application au jeu de Go, LIP6, rapport de thèse, Université Paris 6, 1996

Damasio A. R., L'erreur de Descartes: la raison des émotions. Paris, Odile Jacob, 1995.

Damasio A. R., Le sentiment même de soi. Paris, Odile Jacob, 1999.

Davis R.,Lenat D., Knowledge-based Systems in artificial intelligence. advanced computer science, McGraw-Hill, 1982.

Dennett D., La conscience expliquée. Paris, Odile Jacob, 1991.

Hofstadter D., Fluid Concepts and Creative Analogies. New York, BasicBooks, 1995.

Junghanns A.,Schaeffer J., "Sokoban: Enhancing general single-agent search methods using domain knowledge", Artificial Intelligence, 129, 2001, p. 219-251.

Kirsh D.,Maglio P., "On Distinguishing Epistemic from Pragmatic Action", Cognitive Science, 18, 1994, p. 513-549.

Lenat D., "The Ubiquity of Discovery", Artificial Intelligence, 9, 1978, p. 257-285.

Lenat D., "The Nature of Heuristics", Artificial Intelligence, 19, 1982, p. 189-249.

Lenat D., "EURISKO: A program that Learns New Heuristics and Domain Concepts", Artificial Intelligence, 21, 1983, p. 61-98.

Marshall J. B., "Metacat: A Program that Judges Creative Analogies in a Microworld", ECAI 02 workshop on creative systems, Lyon, 2002, 77-84.

Minton S., Carbonell J. G., Knoblock C. A., Kuokka D. R., Etzioni O.,Gil Y., "Explanation-Based Learning: A Problem Solving Perspective", Artificial Intelligence, 40, 1989, p. 63-118.

Mitchell M., Analogy-making as perception : a computer model. Cambridge, Massochusetts, MIT Press, 1993.

Newell A., Unified Theories of Cognition. Cambridge, Harvard University Press, 1990.

Petta P.,Trappl R., "Emotions and Agents", ACAI 2001, Prague, 2001, Springer-Verlag, 301-316.

Pitrat J., "Implementation of a reflective system", Future Generation Computer Systems, 12, 1996, p. 235-242.

Pitrat J., "Reconnaitre les émotions des êtres humains", Colloque Intelligence Artificielle Berder 1997, 1997, 33-46.

Velasquez J. D., "Modeling Emotions and Other Motivations in Synthetic Agents", AAAI 97, 1997, MIT Press.

Velasquez J. D., "When Robots Weep: Emotional Memories and Decision-Making", AAAI 98, Madison, Wisconsin, 1998, MIT Press, 70-75.

Wooldridge M. J., Intelligent Agents. Multiagent Systems, G. Weiss, Ed., MIT Press, 1999, 27-78.