

Rapport final du TER GMIN20B
du Master Informatique 1^{re} année,
effectué de Janvier à Mai 2012,
encadré par Violaine PRINCE
et Guillaume TISSERANT

Intelligence artificielle : une approche cognitive

RAPPORT FINAL (MAI 2012)

Travail réalisé par l'équipe



William DYCE
Thibaut MARMIN
Namrata PATEL
Clément Sipieter

https://github.com/cogitoTeam/IA_a_cognitive_approach

Rapport final du TER GMIN20B
du Master Informatique 1^{re} année,
effectué de Janvier à Mai 2012,
encadré par Violaine PRINCE
et Guillaume TISSERANT

Intelligence artificielle : une approche cognitive

RAPPORT FINAL (MAI 2012)

Travail réalisé par l'équipe



William DYCE
Thibaut MARMIN
Namrata PATEL
Clément Sipieter

https://github.com/cogitoTeam/IA_a_cognitive_approach

CC BY-SA 3.0



Le projet COGITO réalisé par l'équipe cogitoTeam¹ est mis à disposition selon les termes de la licence Creative Commons Paternité - Partage à l'Identique 3.0 France².

¹Disponible à l'adresse suivante : https://github.com/cogitoTeam/artificial_consciousness

²Plus d'informations à l'adresse suivante : <http://creativecommons.org/licenses/by-sa/3.0/fr/>

Remerciements

Nous tenons tout d'abord à remercier Violaine Prince, Professeur au LIRMM³ et responsable du Master DECOL⁴ de l'université Montpellier 2, ainsi que Guillaume Tisserant, doctorant dans ce même laboratoire, qui ont accepté d'en-cadrer notre équipe sur ce sujet et de nous conseiller tout au long de nos travaux.

Merci à toutes les personnes présentes lors de notre soutenance⁵ et plus particulièrement aux six membres du Jury : Anne-Elisabeth Baert, Hinde Lilia Bouziane, Mountaz Hascouet, Mathieu Lafourcade, Thérèse Libourel et Marie-Laure Mugnier.

Nous remercions également Rodolphe Giroudeau, Marianne Huchard, Frédéric Koriche et Marie-Laure Mugnier pour leurs conseils et les connaissances que nous avons acquises lors de leurs cours et qui nous ont été utiles à la réalisation de ce projet. Citons également Mountaz Hascouet pour avoir géré l'organisation des TER cette année.

Enfin, nous souhaitons finir en remerciant Cécile Artaud et Gaëlle Vassas, secrétaires des Master du département informatique, ainsi que Sylvie Coulon, secrétaire de la Licence du département informatique de l'université Montpellier 2, pour leur disponibilité lors de nos déplacements au bâtiment 16.

³Laboratoire d'Informatique, de Robotique et de Micro-électronique de Montpellier.

⁴Master DECOL : Données, Connaissances et Langage Naturel.

⁵Travail soutenu le 7 Mai 2012 sur le campus universitaire de l'Université Montpellier 2.

Introduction

Dans le cadre de notre formation⁶ nous avons réalisé un projet TER (Travail d'Étude et de Recherche) encadré par Violaine Prince et Guillaume Tisserant, respectivement Professeur et Doctorant au Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier.

Le but de ce TER fut d'implémenter un modèle d'intelligence artificielle basé sur une approche cognitive. L'agent développé devait être capable d'acquérir de nouveaux concepts sémantiques à partir de son environnement courant et de ses expériences passées.

L'intelligence artificielle moderne se basant principalement sur une vision opérationnelle, il nous semblait intéressant de porter une démarche différente avec un modèle proche de la cognition humaine.

Pour ce faire, nous avons débuté notre étude par l'analyse d'un travail théorique proposant une conceptualisation du fonctionnement du cerveau humain, étude qui a été réalisée en 2010 par Guillaume Tisserant, Guillaume Maurin, Ndongo Wade et Anthony Willemot dans le cadre de l'unité d'enseignement « Cognition Individuelle et Collective » proposée par l'offre du Master Informatique de l'université Montpellier 2.

⁶Première année de Master Informatique à l'Université de Montpellier 2.

Abstract

As part of our post-graduate degree⁷ we worked on what is known as a "Study & Research"⁸ project under Violaine Prince and Guillaume Tisserant, respectively Professor and Doctoral student at the "Computer Science, Robotics and Microelectronics Laboratory of Montpellier"⁹.

The goal of this project was to implement an artificial intelligence based on a model of human consciousness. Since modern artificial intelligence tends to focus on performance rather than cognition this unconventional approach seemed like an interesting proposition. Our agent would need to be able to acquire new semantic concepts by observing its current environment and remembering past experiences, and associate these together to come to new conclusions.

In order to do so we began by analysing a theoretical study of human cognition, conducted in 2010 by Guillaume Tisserant, Guillaume Maurin, Ndongo Wade and Anthony Willemot. This was written as part of the second-year Master of Computer Science course "Cognition in Individuals and Groups"¹⁰ at the University Montpellier 2.

⁷first year of a Master Computer Science at the University Montpellier 2.

⁸in French : "Travail d'Étude et de Recherche" (TER).

⁹in French : "Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier" (LIRMM).

¹⁰in French "Cognition Individuelle et Collective".

Table des matières

1 Présentation du sujet	17
1.1 Objectifs	18
1.2 Pourquoi une approche cognitive ?	18
1.2.1 Une approche par apprentissage	18
1.2.2 Une approche par reconnaissance de formes	18
1.3 Un modèle de représentation de la conscience	19
1.3.1 L'inconscient	19
1.3.2 Le Néocortex : la préconscience et la conscience	20
1.3.3 Synthèse	22
 I Analyse	 23
2 Contraintes et restrictions	25
2.1 Contraintes	25
2.1.1 Charge de travail	25
2.2 Restrictions	26
2.2.1 Aperçu général des limitations appliquées	26
2.2.2 Domaine d'application : le jeu de plateau	27
 3 Conception	 31
3.1 Généralités	31
3.1.1 Architecture générale	31
3.1.2 Déroulement d'un coup	32
3.2 Environnement	34
3.2.1 « Stigmergie »	34

3.2.2	Agent « Arbitre »	35
3.3	Module d'analyse	36
3.3.1	L'analyseur conceptuel de base	36
3.3.2	L'analyseur conceptuel poussé	36
3.4	Module de raisonnement	36
3.4.1	Introspection	37
3.4.2	Valuation des formes	39
3.4.3	Moteur de choix	42
3.5	Mémoire	42
3.5.1	Mémoire à court terme	43
3.5.2	Mémoire épisodique	43
3.5.3	Mémoire sémantique	43
3.6	Méthode de travail	44
II	Développement	47
4	Spécifications techniques	49
4.1	Environnement	49
4.1.1	Aperçu globale	49
4.1.2	Bibliothèque <code>game_logic</code>	50
4.1.3	Serveur <code>game_service</code>	52
4.1.4	Clients	55
4.2	L'agent « Cogito »	57
4.2.1	Représentation des connaissances	57
4.2.2	Le package FOL	57
4.2.3	Spécifications des classes <code>Cbs</code> et <code>Rpbs</code>	58
4.3	Analyse	58
4.3.1	L'analyseur conceptuel de base	58
4.3.2	L'analyseur conceptuel poussé	59
4.4	Raisonnement	59
4.4.1	Introspection	60
4.4.2	Valuation des formes	62
4.4.3	Moteur de choix	63

4.5	Mémoire	63
4.5.1	Interface Mémoire	63
4.5.2	Le SGBD Neo4j	64
4.5.3	Éléments en mémoire	65
4.5.4	Mémoire sémantique	66
4.5.5	Mémoire épisodique	67
4.5.6	Vision globale de la mémoire	67
5	Méthode de travail	69
5.1	Répartition des tâches	69
5.2	Outils utilisés	69
5.2.1	Gestionnaire de versions	69
5.2.2	EtherPad, un éditeur de texte collaboratif	70
5.2.3	Visualisation de graphes	71
5.2.4	Outils de développement	71
6	Conclusion & Perspectives	73
6.1	Conclusion	73
6.2	Perspective	73
6.2.1	Un treillis en mémoire?	73
Glossaire		77
Table des figures		79

Chapitre 1

Présentation du sujet

Sommaire

1.1	Objectifs	18
1.2	Pourquoi une approche cognitive ?	18
1.2.1	Une approche par apprentissage	18
1.2.2	Une approche par reconnaissance de formes	18
1.3	Un modèle de représentation de la conscience . . .	19
1.3.1	L'inconscient	19
1.3.2	Le Néocortex : la préconscience et la conscience . . .	20
1.3.3	Synthèse	22

Nous avons choisi d'aborder ce sujet d'une manière différente de l'approche classique. Dans un premier temps, nous n'avons pas cherché à répondre à un besoin fonctionnel mais à une interrogation d'ordre philosophique :

« Est-il possible de synthétiser dans une machine des processus cognitifs se rapprochant de ceux de l'être humain ? Si oui, comment ? »

C'est une démarche réellement différente de celle proposée par l'intelligence artificielle dite « moderne »¹, la tendance de nos jours étant de suivre une voie plus pragmatique visant à aboutir à des systèmes dits « rationnels », c'est à dire se comportant de manière optimale vis-à-vis d'une mesure de performance donnée. Ce nouveau paradigme s'est constitué en réaction à une IA juvénile atteinte par l'ampleur de ces ambitions. À l'origine, les équipes de recherche pensaient pouvoir construire rapidement des systèmes égalant les capacités humaines, mais les résultats ne sont pas arrivés à la hauteur de leurs espérances. Leurs investisseurs déçus ont alors provoqué durant les années 1970-1990 les « hivers de l'IA » en retirant la majorité de leurs soutiens financiers.

Sans oublier cette leçon d'humilité, nous avons fait le choix d'essayer de renouer avec une approche biomimétique.

¹Cf. Russell S. et Norvig P. (2009 – 3rd Ed.), « Artificial intelligence : a modern approach », Prentice Hall, 0-13-604259-7

1.1 Objectifs

Notre objectif lors de ce TER sera donc de construire une intelligence artificielle basée sur une approche biomimétique, c'est à dire que nous tenterons d'apprécier un problème d'intelligence artificielle à partir de mécanismes connues du fonctionnement du cerveau. Dans un premier temps, nous devrons étudier et assimiler le modèle de la conscience présenté dans le rapport sur lequel se base notre travail. Dans un deuxième temps, il nous faudra décider d'un domaine sur lequel appliquer notre IA. Ensuite, nous formaliserons le modèle général pour l'appliquer au domaine choisi. Enfin, nous implémenterons notre formalisation. Et Finalement, nous prendrons du recul sur notre projet afin d'effectuer un travail d'évaluation de façon objective.

1.2 Pourquoi une approche cognitive ?

L'Homme a toujours cherché à comprendre et à reproduire les mécanismes naturels qui l'entourent. Un des domaines les plus passionnants reste celui de l'étude du cerveau. Qu'il soit humain ou animal, nous restons fascinés par sa capacité à analyser, à comprendre et à généraliser les problèmes que posent ou « proposent » l'environnement.

Dans le but de se rapprocher du fonctionnement du cerveau, nous traiterons le sujet de notre TER avec les approches suivantes.

1.2.1 Une approche par apprentissage

Une des principales caractéristiques de l'intelligence, et certainement une des plus importantes, est la capacité d'apprentissage. C'est pourquoi notre IA devra améliorer ses performances avec le temps en se basant sur ses expériences. Nous pouvons citer Alain Bonnet qui dans son livre « L'intelligence artificielle, promesses et réalités », paru en 1984, écrit : « *Les programmes devront apprendre avec l'expérience et s'auto-améliorer sur de simples jugements de leurs performances que les experts humains fourniront. Dans un premier temps, ils pourront améliorer leurs connaissances et dans un deuxième, leurs mécanismes d'utilisation de ces connaissances, c'est à dire leurs stratégies de plus haut niveau* ».

1.2.2 Une approche par reconnaissance de formes

La reconnaissance de formes dans notre environnement est une autre autre des principales capacités de notre cerveau. En effet, le cerveau interprète les informations visuelles qui lui sont transmises et en extrait des concepts ou formes connues. Par exemple, pour un être humain, les formes telles que « chaise », « porte », « fenêtre » sont automatiquement reconnues par notre cerveau.

1.3 Un modèle de représentation de la conscience

Dans le but de concrétiser cette approche, nous nous sommes inspirés du travail réalisé par Guillaume Tisserant, Guillaume Maurin, Ndongo Wade et Anthony Willemot. La figure 1.1 présente la structure de la conscience artificielle décrite dans cette étude. Celle-ci conserve la plupart des caractéristiques d'une conscience humaine grâce à des recherches sur de nombreux travaux comme ceux de Freud et de Laborit².

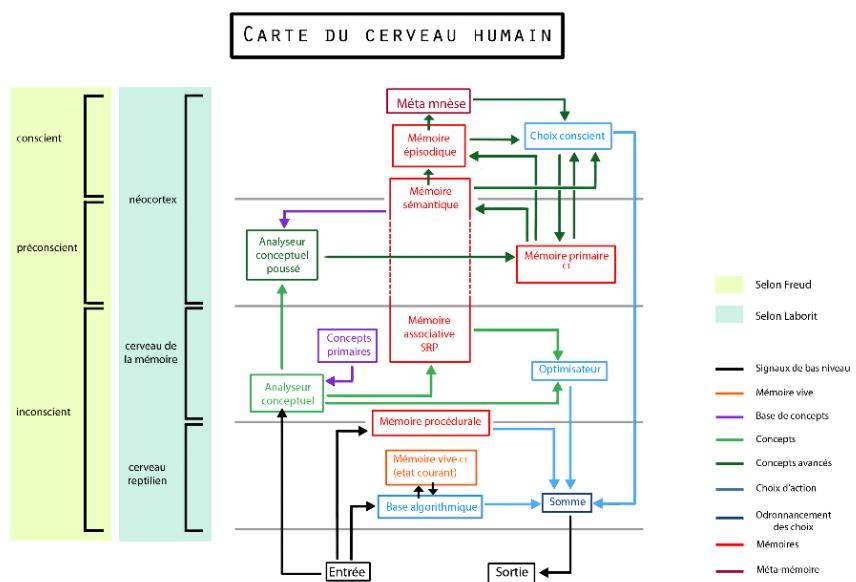


FIG. 1.1 – Schéma du modèle de représentation de la conscience

Cette représentation peut être vue comme un empilement de couches : l'inconscient se situant au plus bas et le conscient au plus haut.

1.3.1 L'inconscient

L'inconscient est composé de deux niveaux : le cerveau reptilien et le cerveau de la mémoire.

Le cerveau reptilien

C'est la couche la plus basse du modèle.

Les entrées/sorties Les entrées permettent au cerveau d'acquérir deux types d'informations :

²Henri Laborit (1914–1995) : Médecin chirurgien et neurobiologiste.

- les percepts externes, émis par l'environnement et perçus par les cinq sens,
- les percepts internes, comme les pulsions, etc.

Les sorties assurent la transmission des informations produites par le cerveau au reste du corps.

La base algorithmique Il s'agit d'une base simple assurant une réaction rapide à des entrées simples, en produisant des sorties simples. C'est cette partie qui stocke les réflexes innés présents chez les êtres vivants.

La mémoire procédurale Elle permet de construire des automatismes en fonction du vécu. Par exemple, selon un schéma essai échec, un bébé apprendra à marcher en utilisant sa mémoire procédurale.

Le cerveau de la mémoire

Cette couche est plus complexe que la dernière. Elle est composée d'un analyseur conceptuel, d'une mémoire associative et d'un optimisateur.

L'analyseur conceptuel Il analyse des entrées en dur à une base de données de concepts primaires (comme la faim, la soif, le plaisir, la fatigue, la joie, une odeur nauséabonde, un bruit violent, etc.).

La mémoire associative Elle assure l'association de concepts primaires entre eux. Elle permet par exemple à un animal vivant dans un milieu contenant des prédateurs bruyants d'associer le concept de danger au bruit émis par le prédateur.

L'optimisateur Il a pour vocation d'optimiser un concept qui peut se traduire par le bien-être. Prenons l'exemple précédent : si l'animal sait que l'approche d'un prédateur réduit le bien-être, et que le fait de courir peut éloigner le prédateur et donc faire remonter le bien-être, alors l'optimisateur permettra à l'animal d'inférer l'action de fuir.

1.3.2 Le Néocortex : la préconscience et la conscience

Les couches faisant partie du néocortex représentent la préconscience et la conscience. C'est en montant dans ces éléments que l'on s'approche de la cognition humaine.

Le préconscient

Cette couche est constituée d'un analyseur conceptuel sémantique et des mémoires primaire et sémantique.

L'analyseur conceptuel sémantique Il manipule des concepts d'un niveau supérieur à ceux manipulés par l'analyseur conceptuel du cerveau de la mémoire. Il permet la traduction de concepts primaires en concepts complexes, qui seront stockés en mémoire sémantique.

La mémoire sémantique Elle contient une base de concepts avancés, construite sous forme de treillis, permettant l'association de concepts.

La mémoire primaire Cette mémoire contient un nombre de concepts sémantiques limité. Elle joue le rôle d'une mémoire à court terme qui stocke tout ce dont la conscience est en train de manipuler. La mémoire primaire peut recevoir des informations de l'analyseur conceptuel poussé, qui lui envoie les concepts se trouvant dans l'environnement, ou directement par la réflexion consciente, qui choisit d'y stocker un concept sur lequel une réflexion est nécessaire.

La gestion de la mémoire sémantique Les concepts en mémoire primaire sont instantanément copiés en mémoire sémantique. Toutes les informations qui passent en mémoire primaire y sont transmit. Cependant, les informations accumulées peuvent être oubliées. L'oubli d'un concept dépend de l'activité de ce dernier en mémoire primaire.

Le conscient

Cette partie regroupe les modules les plus avancés du modèle : la mémoire épisodique, la métamnèse et le choix conscient.

La mémoire épisodique Il s'agit d'une mémoire des situations, qui permet la remémoration événements et des émotions vécues sous formes de concepts sémantiques. Les concepts stockés proviennent à la fois des percepts et de l'état interne de la personne au moment de l'événement. Elle gère la persistance au même titre que la mémoire sémantique.

La métamnèse C'est la mémoire de la mémoire. Son rôle est de stocker la façon dont les éléments se sont enchaînés dans la mémoire épisodique.

Le choix conscient (ou réflexion consciente) Il peut être vu comme un optimisateur de haut niveau, maniant des concepts poussés, et de natures différentes. Grâce à la métamnèse, il peut utiliser des séries de situations, et même imaginer des situations passées, présentes ou futures pour prendre des décisions. Il est ainsi capable de comparer des situations imaginaires et choisir celle vers laquelle il a envie de tendre. Il a aussi la capacité à réfléchir sur des concepts abstraits, comme sur ses réflexions passées. Il est capable d'analyser son propre fonctionnement du moment où il arrive à le retranscrire sous forme de concepts.

1.3.3 Synthèse

L'objet représenté par **Somme** dans la couche du cerveau reptilien est une simplification du schéma retour. Il représente la transformation de tous les signaux en signaux de bas niveaux, et gère leur somme pour choisir lesquels inhiber et lesquels transmettre en sortie. Son fonctionnement nécessite d'arriver à combiner les signaux entre eux. Par exemple, si une personne choisit de fumer dans le choix conscient, il doit récupérer la façon d'allumer un briquet dans la mémoire procédurale.

Première partie

Analyse

Chapitre 2

Contraintes et restrictions

Sommaire

2.1	Contraintes	25
2.1.1	Charge de travail	25
2.2	Restrictions	26
2.2.1	Aperçu général des limitations appliquées	26
2.2.2	Domaine d'application : le jeu de plateau	27

2.1 Contraintes

2.1.1 Charge de travail

La charge de travail est directement liée à plusieurs critères : le temps, l'effectif disponible et les compétences de chacun des membres. L'équipe COGITO que nous composons et qui a réalisé ce projet est composée de quatre étudiants en Master informatique première année (DECOL¹ et IMAGINA²). Certaines connaissances nécessaires à la réalisation du projet ont été enseignées durant le semestre, ce qui nous a demandé une effort d'anticipation, d'adaptation et de réactivité. Le projet s'est déroulé du mois de Février au mois d'Avril, soit environ trois mois de réalisation, période durant laquelle devaient être effectuées les tâches suivantes :

- étude du sujet,
- formalisation,
- développement,
- évaluation,
- et préparation du rendu (réécriture du présent rapport et préparation de la soutenance).

¹Master DECOL : Données Connaissances et Langage Naturel.

²Master IMAGINA : Images, Games and Intelligent Agents.

Nous avons donc dû effectuer un gros travail de simplification du modèle initial en restant modeste, afin de pouvoir offrir un outil aboutit et fonctionnel à la date du rendu.

2.2 Restrictions

2.2.1 Aperçu général des limitations appliquées

La figure 2.1 met en évidence le sous ensemble que nous avons choisi d'étudier, celui-ci a été déterminé à partir des contraintes énumérées au chapitre 2. Cette sous partie correspond à ce que Laborit nomme le néocortex, à laquelle nous avons soustrait la Méta mnèse.

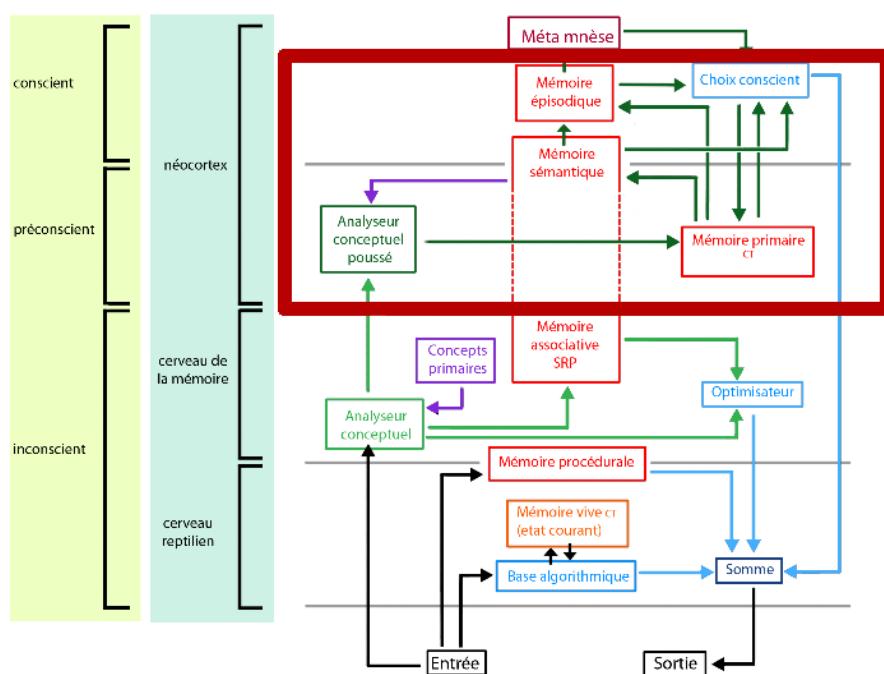


FIG. 2.1 – Modèle restreint déterminé à partir des contraintes énumérées au chapitre 2

La figure 2.2 page ci-contre isole ce sous ensemble afin de le rendre plus lisible.

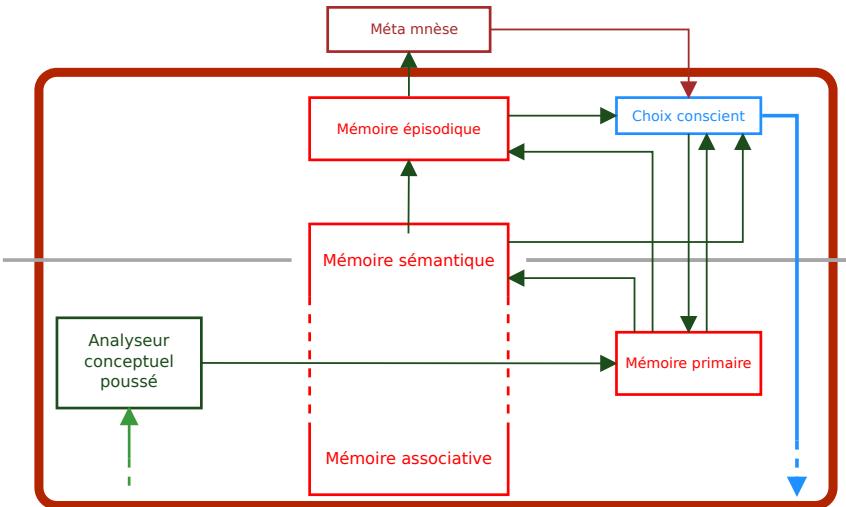


FIG. 2.2 – Zoom sur le modèle restreint.

2.2.2 Domaine d'application : le jeu de plateau

Suite à victoire de Deep Blue contre Garry Kasparov en 1997, un certain Robert Levinson nota que la machine « ne sait même pas qu'il joue aux échecs ». Ici nous tentons justement de construire une intelligence artificielle qui fonctionne comme un être vivant et non comme un calculateur.

Cependant, une IA « forte » étant hors de la portée d'un projet TER de Master, nous devions nous limiter en plus à une application et un environnement précis. C'est donc en s'inspirant de cette histoire que nous avons choisi le jeu de plateau comme arène.

Plus précisément nous nous limiterons à des jeux qui suivent des contraintes très précises, à savoir :

- deux-joueurs (noir et blanc),
- placement à tour de rôle d'un nouveau pion (pas de déplacements),
- un seul type de pion,
- plateau matriciel, cases et côtés sans valeurs associées.

Les jeux les plus connus sont par exemple :

- Morpion / Puissance-4
- Go
- Reversi

Le jeu principal que nous considérons est le « Reversi »³, mais l'IA doit être capable d'apprendre à jouer à tout autre jeu de plateau respectant les contraintes définies ci-dessus.

³« Reversi » : connu également sous le nom commercial « Othello ».

Convergence entre spécialités

Dans les années précédentes les Masters DECOL⁴ et IMAGINA⁵ étaient rassemblés en une seule spécialité appelée I2A⁶. Nous étions plusieurs dans ce groupe à hésiter entre ces deux formations. Un projet sur la cognition appliquée aux jeux présentait donc un très bon moyen de rassembler nos connaissances et de travailler ensemble sur l'intelligence artificielle générale.

Activité purement cognitive

Nous avons déjà parlé de la simulation des parties basses du modèle de cognition, à savoir l'inconscient. La restriction du domaine d'application au jeu de plateau permet de l'ignorer presque complètement. Certes l'intuition peut jouer un rôle dans le jeu, mais ce défi reste principalement un travail de réflexion d'ordre conscient. Nous n'aurons besoin ni de réflexes, ni de mémoire procédurale.

Activité cognitive complète

Gagner un jeu de plateau n'est pourtant pas un travail simple. Il ne suffit pas de prendre le « meilleur » coup à chaque tour : pour bien jouer il faut que nous soyons capables de *modéliser* notre adversaire afin de prédire ses coups, et même de modéliser le modèle qu'il se fait de nous. Il faut ensuite être capable d'utiliser ce modèle pour élaborer une stratégie, donc de *plannifier*. Nous devrons finalement *apprendre* suite à nos erreurs si nous ne voulons pas tomber constamment dans les mêmes pièges.

Évaluations et comparaisons faciles

Pour mesurer empiriquement la performance de notre agent il suffit de le mettre à l'épreuve contre un autre agent à de multiples reprises pour calculer le pourcentage de parties gagnées. Prenons par exemple un agent choisissant aléatoirement ses coups et regardons la proportion de parties gagnées par notre IA :

- $\gg 50\%$ notre agent joue de façon *rationnelle*,
- $\approx 50\%$ notre agent joue *aléatoirement*,
- $\ll 50\%$ notre agent joue de manière *irrationnelle*.

Le vrai défi est évidemment de mettre notre agent face à un agent basé sur la stratégie « MiniMax », théoriquement optimale si elle est appliquée de façon exacte. Cependant il ne faut pas oublier que l'efficacité n'est pas le but principal de ce projet.

⁴Master DECOL : « Données Connaissances et Langues »

⁵Master IMAGINA : « Images Games and Intelligent Agents »

⁶Master I2A : « Ingénierie de l'Intelligence Artificielle »

Existence d'une stratégie optimale

En 1928 John Von Neumann publie un théorème à propos du « MiniMax » prouvant que cette stratégie de minimisation de perte maximum est théoriquement optimale lorsqu'elle est confrontée à des jeux ayant pour caractéristiques :

- d'être joués par deux adversaires,
- d'être à « somme nulle ⁷ »,
- d'avoir une durée d'engagement finie, tout comme le nombre d'options possibles.

Depuis cette époque les ordinateurs ont dépassé les meilleurs humains aux échecs. Pourtant certains jeux comme le Go résistent encore car la combinatoire empêche les ordinateurs d'appliquer de manière exacte la stratégie optimale, qui même avec triage et élagage « Alpha-Beta »⁸ possède une complexité en $O(b^{\frac{d}{2}})$.

Il en résulte que le « MiniMax » doit être complémenté d'heuristiques, donc d'expertise humaine, afin de répondre dans un délai acceptable. Nous pouvons alors nous demander si *la machine ne pourrait pas acquérir d'elle-même cette expertise ...*

⁷jeu à « somme nulle » : le gain de l'un est exactement la perte de l'autre et vice-versa.

⁸« Élagage Alpha-Beta » : forme de séparation et évaluation permettant d'éviter des explorations inutiles de l'espace des possibilités.

Chapitre 3

Conception

Sommaire

3.1	Généralités	31
3.1.1	Architecture générale	31
3.1.2	Déroulement d'un coup	32
3.2	Environnement	34
3.2.1	« Stigmergie »	34
3.2.2	Agent « Arbitre »	35
3.3	Module d'analyse	36
3.3.1	L'analyseur conceptuel de base	36
3.3.2	L'analyseur conceptuel poussé	36
3.4	Module de raisonnement	36
3.4.1	Introspection	37
3.4.2	Valuation des formes	39
3.4.3	Moteur de choix	42
3.5	Mémoire	42
3.5.1	Mémoire à court terme	43
3.5.2	Mémoire épisodique	43
3.5.3	Mémoire sémantique	43
3.6	Méthode de travail	44

Dans cette partie d'Analyse, nous appellerons notre modèle d'IA COGITO.

3.1 Généralités

3.1.1 Architecture générale

L'architecture générale de *COGITO* est présentée sur la figure 3.1. Elle est composée de trois modules :

- l'analyseur, composé d'un *RuleBook*, d'un analyseur conceptuel de base et d'un analyseur conceptuel poussé,

- le raisonneur, qui contient un moteur de choix, et un module d’introspection,
- et la mémoire, structurée avec une interface nommée mémoire primaire.

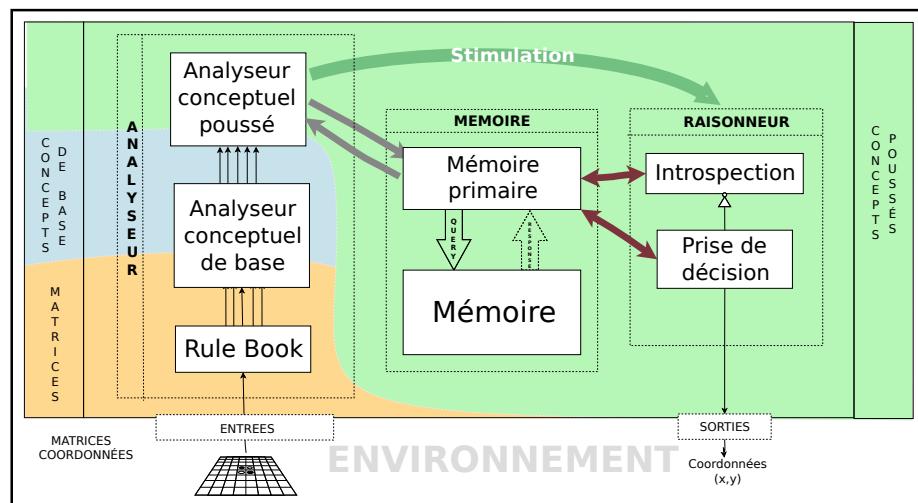


FIG. 3.1 – Schéma général de *COGITO*

L’environnement est externe à l’agent, et interagit avec ce dernier via ses entrées / sorties.

Trois niveaux de données sont manipulés par l’IA :

- des données sous forme matricelle provenant de l’environnement,
- ces mêmes données décrites dans un formalisme logique,
- et enfin, les données précédentes associées à des formes connues, que nous appelerons concepts poussés.

3.1.2 Déroulement d'un coup

La figure 3.2 page suivante décrit les interactions entre les modules de l’IA lorsque qu’un coup doit être joué.

Un plateau, décrit sous forme matricielle, est transmis par l’environnement, via un module d’*E/S*¹, au *RuleBook*. Ce dernier détermine l’ensemble des coups possibles et génère l’ensemble des matrices résultantes correspondantes qu’il transmet à l’*ACB*². Ce module traduit les plateaux reçus d’un format matriciel dans un formalisme logique. On obtient alors un ensemble de faits dans un format logique, pouvant être vus comme une base de faits, qui est fournie à l’*ACP*³. Ce dernier interroge la mémoire pour obtenir une série de formes remarquables. L’*ACP* associe à chaque plateau les formes qui peuvent y être reconnues. Une fois analysé, cet ensemble, plateaux et formes remarquables, est stocké en mémoire primaire. L’*ACP* stimule ensuite le module de raisonnement

¹Entrée/Sortie

²Analyseur Conceptuel de Base

³Analyseur Conceptuel Poussé

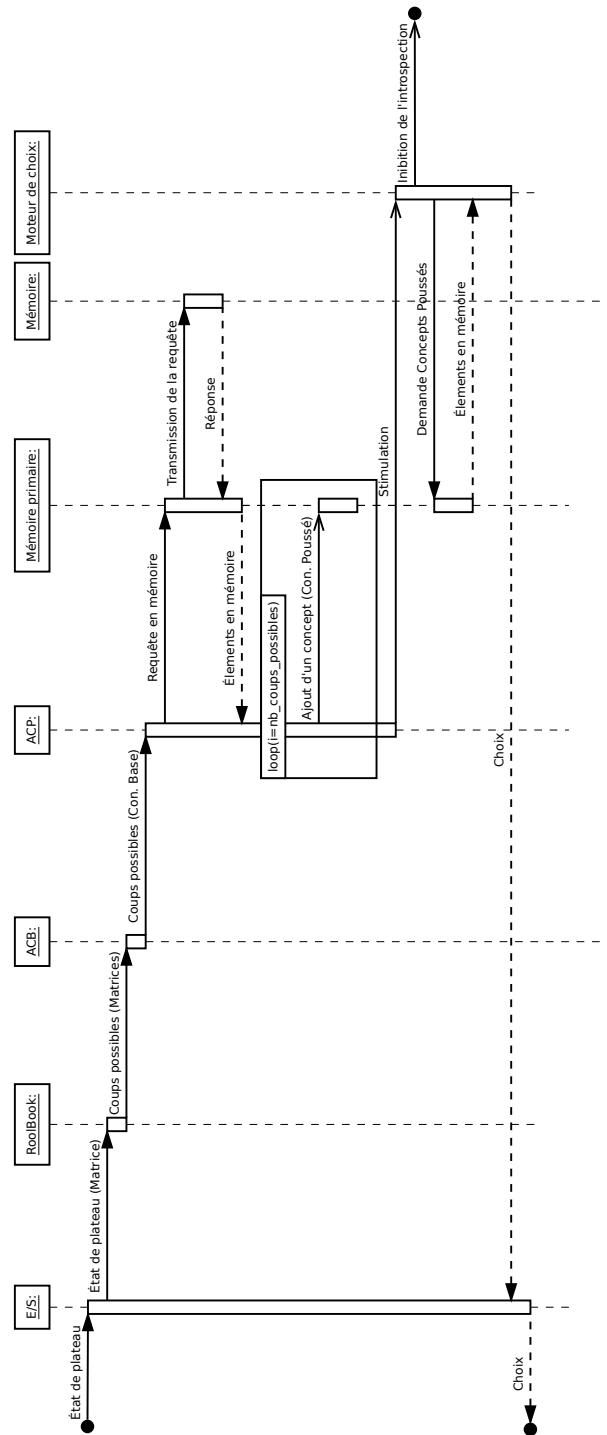


FIG. 3.2 – Diagramme de séquence sommaire de déroulement d'un coup

qui, à partir de l'ensemble des plateaux possibles, des formes remarquables qui y sont associées et de la valuation de ces formes fournit par la mémoire, fait un choix entre les différents coups possibles. Cette décision est successivement transmise au module d'*E/S* puis à l'environnement.

3.2 Environnement

3.2.1 « Stigmergie »

Dans le but de bien dissocier la logique liée à la réflexion de celle qui structure le jeu, nous avons choisi de suivre le paradigme « Agent et Environnement ». *COGITO* est assimilé à un agent qui évolue dans un environnement qu'il peut observer et modifier.

Toute communication entre agents se fait donc par « Black Board »⁴, c'est à dire en laissant des traces dans cet environnement partagé. Cette interaction dite « Stigmergique »⁵ est nécessairement asynchrone mais possède l'avantage de ne nécessiter qu'un seul canal de communication par agent. Ce canal, qui relie l'agent à son environnement, est bidirectionnelle : il permet à l'agent de recevoir des stimuli et d'envoyer des impulsions. Dans la suite de ce document, nous parlerons de « percepts » et d'« actions ».

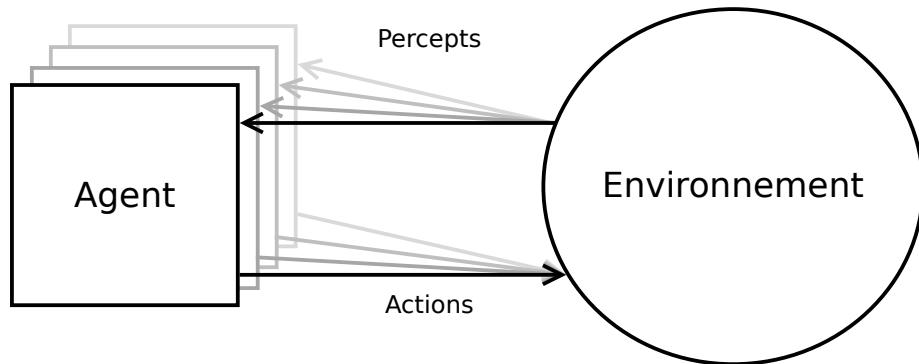


FIG. 3.3 – Modèle « Agent et Environnement »

La séparation entre l'agent et son environnement permet de gagner en flexibilité : la communication ne se faisant qu'au travers de l'envoie de messages précis, tout algorithme peut être encapsulé avec une interface simple pour en faire un agent.

L'agent peut donc être un humain ou une machine. Une machine pouvant exécuter notre modèle aussi bien qu'une intelligence artificielle tiers.

⁴ « Black Board » (Tableau Noir) : base de connaissance partagée et itérativement construite par un ensemble d'agents. (Article « Blackboard system » de Wikipédia en anglais, licence CC-BY-SA)

⁵ « Stigmergie » : méthode de communication indirecte dans un environnement émergent auto-organisé, où les individus communiquent entre eux en modifiant leur environnement. (Article « Stigmergie » de Wikipédia en français, licence CC-BY-SA)

Ainsi, nous disposons d'une abstraction du fonctionnement interne de chaque agent, ce qui permet d'une part de faire jouer *COGITO* contre un autre agent basé sur une stratégie connue, facilitant ainsi l'évaluation de notre modèle, d'autre part de tester les développements qui concernent l'environnement avant même que notre IA ne soit fonctionnelle en faisant jouer des agents naïfs entre eux. Cette méthode permet d'isoler pleinement le développement de la plate-forme de tests de celui de l'IA.

3.2.2 Agent « Arbitre »

Comme indiqué dans la section 2.2.2 (page 27) l'environnement est un jeu de plateau. Il possède donc une grille dont les cases peuvent contenir un pion de couleur donnée ou être vides.

L'environnement doit aussi connaître le prochain joueur qui doit prendre la main ainsi que toute autre information qui ne peut pas être déduite de l'état du plateau. Il doit également pouvoir juger de la légalité des coups proposés par un agent, et par le même biais savoir comment initialiser le plateau lors d'une nouvelle partie.

Il se décompose donc en trois modules :

- un plateau,
- une liste de règles,
- un processeur qui les manipule.

Plutôt que de briser la métaphore du « Black Board » en parlant d'*environnement agent*, nous avons choisi de voir ce processeur comme un agent tiers : une sorte d'arbitre neutre. C'est un agent purement réactif qui maintient l'état du jeu en exécutant les coups qui lui sont envoyés par les agents joueurs dans la mesure du légal. Il doit veiller à notifier le résultat des actions effectuées et envoyer l'état du plateau en cas de demande.

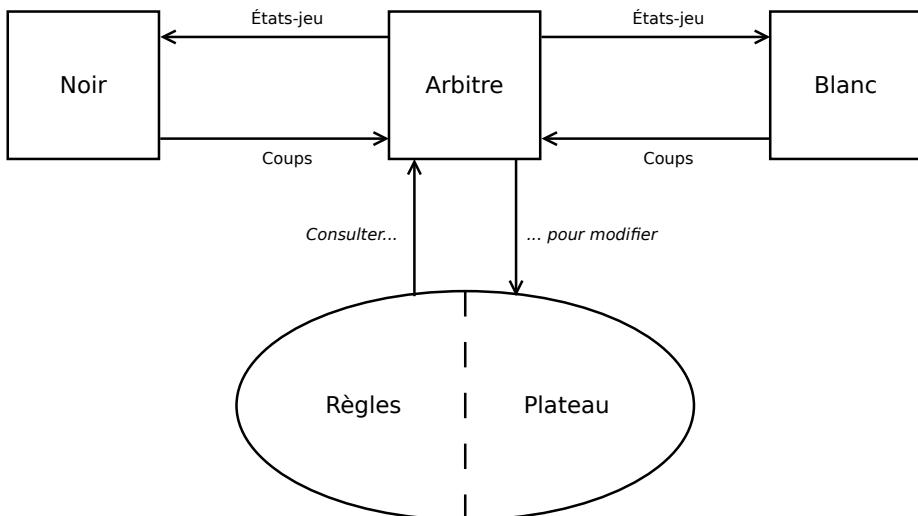


FIG. 3.4 – Interactions entre l'Arbitre et les autres agents

C'est donc à travers cet *agent-filtre* que les joueurs peuvent modifier l'environnement et, par conséquent, communiquer entre eux.

Suite à cette étude préalable, il est claire qu'une architecture *client-serveur* est recommandée pour l'implémentation des interactions entre les agents et leur environnement.

3.3 Module d'analyse

Le rôle du module d'analyse est de traduire la représentation de l'environnement dans un format compréhensible par le système pour ensuite y reconnaître des formes. Ces deux étapes sont gérées respectivement par les deux sous-modules : l'*analyseur conceptuel de base* et l'*analyseur conceptuel poussé*.

3.3.1 L'analyseur conceptuel de base

Lorsque *COGITO* reçoit un plateau en entrée, le *Rule-Book* transforme ce plateau en un paquet qui correspond à l'entrée de l'ACB⁶. Ce paquet contient un plateau courant et l'ensemble des plateaux résultants des coups possibles. Une première analyse d'un plateau au format matriciel lui permet d'extraire des faits représentant la configuration de celui-ci. L'ACB crée alors un paquet correspondant au paquet fourni par l'environnement en convertissant ainsi chaque plateau de ce paquet en un ensemble de faits. Elle passe ensuite ce paquet à l'analyseur conceptuel poussé.

3.3.2 L'analyseur conceptuel poussé

L'ACP⁷ a pour but d'associer à chaque plateau du paquet reçu en entrée, une liste de formes pertinentes reconnues, par un mécanisme de reconnaissance de formes. Il procède par une recherche de formes déjà connues (stockées en mémoire) dans chacun des plateaux présents dans ce paquet. Une fois qu'une forme est reconnue, elle est ajoutée à la liste de formes pertinentes associée à ce plateau. L'analyseur enrichit ainsi le paquet fourni par l'analyseur de base.

L'analyseur passe enfin ce paquet enrichi à la mémoire et stimule le module de raisonnement afin que celui-ci commence son processus.

3.4 Module de raisonnement

Le module de raisonnement se compose de deux éléments :

- le moteur d'introspection, qui recherche de nouvelles formes remarquables en analysant les expériences passées dans le but d'optimiser les chances de victoire lors des jeux futurs,
- le moteur de choix, qui a pour objectif d'annoter les environnements fournis afin de choisir celui qui maximise sa fonction objectif.

⁶ Analyseur Conceptuel de Base.

⁷ Analyseur Conceptuel Poussé.

3.4.1 Introspection

Cadre Général

Le module d'introspection a pour objectif d'extraire, à partir des expériences passées de l'IA, de nouvelles formes remarquables potentiellement discriminantes pour les choix futurs. Pour ce faire, il choisit aléatoirement, en mémoire, au moins deux environnements ayant reçu une même annotation et tente d'étendre les formes déjà connues.

Exemple de la figure 3.5 Nous considérons deux environnements présents en mémoire. Le premier décrit *un homme avec un chapeau et un parapluie*, le second *un homme avec un chapeau et une canne*. Nous souhaitons extraire de ces deux environnements la forme homme avec un chapeau. Pour ce faire, le module se base sur les connaissances relatives à ces environnements présentes en mémoire. Considérons que *COGITO* sache déjà reconnaître un homme, le moteur d'introspection cherche alors à étendre le plus possible la forme *homme* dans un des deux environnements tout en vérifiant que cette forme étendue peut être reconnue dans l'autre environnement, comme illustré dans la figure 3.6.

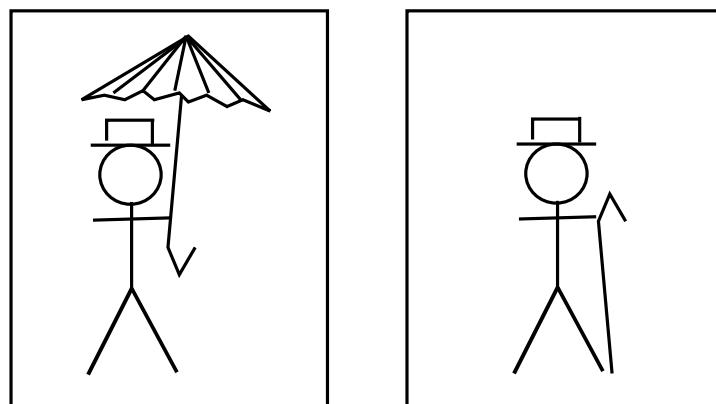


FIG. 3.5 – Exemple d'environnements

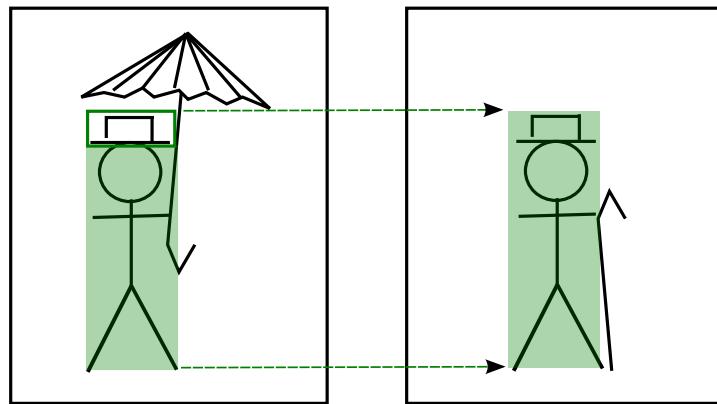


FIG. 3.6 – Exemple d'injection (basé sur la figure 3.5)

Application aux jeux de plateau

La figure 3.7 représente, de façon graphique, le type d'environnement avec lequel doit travailler le moteur d'introspection.

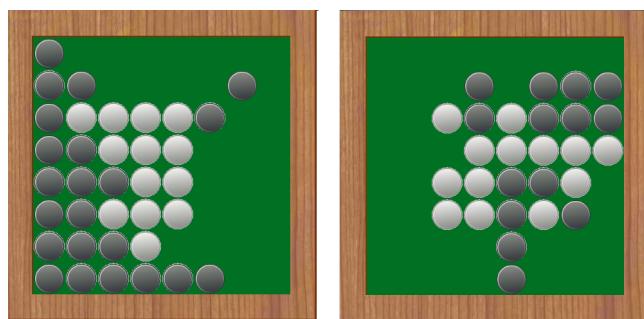


FIG. 3.7 – Représentation graphique de l'environnement

Dans un premier temps, le moteur d'introspection va rechercher en mémoire deux plateaux ayant une même annotation et présentant des formes connues identiques (cf. figure 3.8). Selon notre restriction aux jeux de plateau, l'annotation peut être « perdant » ou « gagnant ».

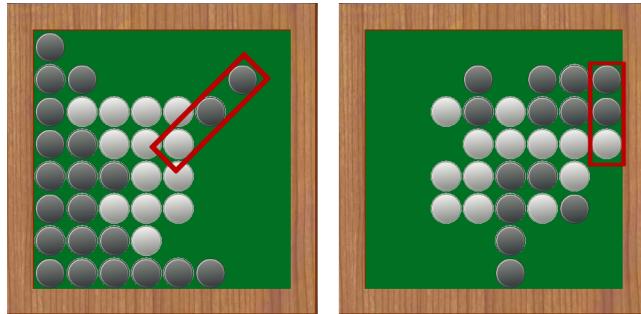


FIG. 3.8 – Deux plateaux ayant des formes connues en commun

Le module essaie ensuite d'étendre cette forme à partir d'un des plateaux tout en vérifiant qu'elle reste présente dans l'autre plateau, comme illustré dans la figure 3.9.

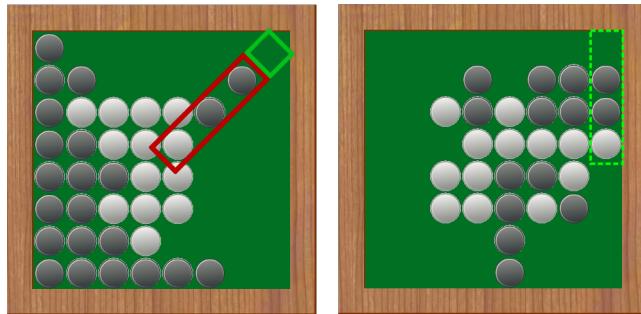


FIG. 3.9 – Illustration de l'injection sur des plateaux

3.4.2 Valuation des formes

Cadre Général

Pour chaque environnement annoté, le module de raisonnement effectue une mise à jour des probabilités d'apparition d'une annotation en fonction d'une forme.

$$P(\text{Annotation}|\text{Forme}) = \frac{P(\text{Forme}|\text{Annotation}) \times P(\text{Annotation})}{P(\text{Forme})}$$

Exemple de la figure 3.10 Imaginons que nous souhaitons entraîner *COGITO* à identifier des formes géométriques basiques : *rond*, *carré*, *croix* et *triangle*. Admettons que notre IA soit capable de reconnaître dans son environnement les

formes correspondantes mais qu'elle ne sache pas les associer aux annotations correspondantes.

	rond	carré	croix	triangle
○				
□				
×				
△				

FIG. 3.10 – Association formes-annotations

Partant de cet état initial, l'IA doit apprendre de ses expériences. Pour ce faire elle doit mettre à jour les probabilités qu'une annotation soit rattachée à un environnement pour chaque nouvel environnement rencontré. La figure 3.11 met en évidence les probabilités calculées suite à la rencontre d'un environnement présentant la forme *rond*.

environnement	annotations	probabilités
	rond ¬carré ¬croix ¬triangle	
○		
□		
×		
△		

FIG. 3.11 – Association formes-annotations basique à l'étape 1

Le calcul des probabilités s'effectue de la même manière pour un environnement qui présente un ensemble de formes, comme illustré sur la figure 3.12. Il existe alors une ambiguïté car le système ne peut pas savoir quelle forme est responsable de quelle annotation.

environnement	annotations	probabilités
	¬rond carré croix triangle	
×		
△		
□		

FIG. 3.12 – Association formes-annotations multiple à l'étape 2

Cependant, cette ambiguïté est levée par l'expérience qui confrontera le système avec des ensembles variés de formes. Par conséquent, seul l'annotation correcte pour chaque forme conserve une probabilité certaine comme il est

possible de l'observer sur les figures 3.13, 3.14 et 3.15 illustrant l'évolution du système.

environnement	annotations	probabilités
	rond ¬carré croix ¬triangle	
X		
O		

	rond	carré	croix	triangle
rond	1	0	0,5	0
carré	0	1	1	1
croix	0,5	0,5	1	0,5
¬triangle	0	1	1	1

FIG. 3.13 – Association formes-annotations à l'étape 3

environnement	annotations	probabilités
	¬rond carré croix ¬triangle	
X O		

	rond	carré	croix	triangle
¬rond	1	0	0,5	0
carré	0	1	1	0,5
croix	0,33	0,66	1	0,33
¬triangle	0	1	1	1

FIG. 3.14 – Association formes-annotations à l'étape 4

environnement	annotations	probabilités
	¬rond ¬carré ¬croix triangle	
△		

	rond	carré	croix	triangle
¬rond	1	0	0,5	0
¬carré	0	1	1	0,5
¬croix	0,33	0,66	1	0,33
triangle	0	0,5	0,5	1

FIG. 3.15 – Association formes-annotations à l'étape 5

Il est important de noter que la certitude, représentée par une probabilité égale à 1, ne peut être garanti que si les annotations fournies au système lors de sa période d'apprentissage sont exemptes d'erreurs. Cependant, une erreur d'annotations lors de l'apprentissage certes diminuerai la probabilité associé à l'annotation correct mais pour vu que le jeux d'apprentissage soit assez grands, celle-ci resterai la plus probable.

Application aux jeux de plateau

Dans le cadre des jeux de plateau deux annotations sont fournies :

- *victoire*,
- *défaite*.

Il est évident, mais nécessaire, de préciser que :

- *victoire* $\rightarrow \neg$ *défaite*
- *défaite* $\rightarrow \neg$ *victoire*

Ces annotations doivent être fournies directement par l'environnement sans intervention humaine, nous sommes donc sûr de l'apprentissage par renforcement.

3.4.3 Moteur de choix

Cadre Général

Ce module entre en action après stimulation de la phase d'analyse qui lui fournit, par l'intermédiaire de la mémoire, l'environnement courant et un ensemble de formes reconnues. Le *moteur de choix* se sert alors de la probabilité d'apparition des annotations associées aux formes reconnues afin d'annoter l'environnement courant.

Application aux jeux de plateau

Dans le cadre des jeux de plateau, le moteur de choix obtient un ensemble de plateaux, chacun associé à un ensemble de formes reconnues. Chaque plateau correspond au plateau résultant d'un coup possible. On peut donc dire que ce moteur doit choisir entre les différents « futurs possibles ».

Afin d'évaluer la probabilité de gain d'un plateau, celui-ci fera la moyenne des probabilités de gain des formes reconnues sur ce plateau. Il choisit finalement l'environnement qui maximise la probabilité de gain.

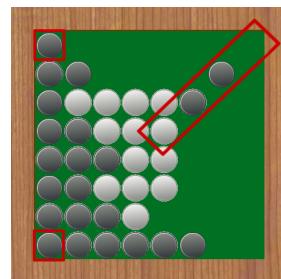


FIG. 3.16 – Représentation graphique de l'environnement

3.5 Mémoire

Le module de mémoire se décompose en trois parties :

- la mémoire à court terme,
- la mémoire épisodique,
- la mémoire sémantique.

3.5.1 Mémoire à court terme

La mémoire à court terme est le module avec lequel communiquent les modules d'analyse et de raisonnement. Elle a un rôle de *buffer* permettant le stockage temporaire des données soumises par le l'*analyseur conceptuel poussé*, et fournies au *module de raisonnement*.

L'accès aux ressources stockées en mémoire sémantique ou épisodique se fait également par la mémoire à court terme : elle peut être décrite comme l'interface mémoire.

3.5.2 Mémoire épisodique

La mémoire épisodique stocke des parties et des coups. Tous les éléments sont datés au moment de l'ajout et peuvent être parcourus par ordre chronologique inversé (du plus récent au plus ancien).

Cette mémoire est structurée sous la forme d'une double liste chaînée :

- liste chaînée de parties, avec accès à la dernière partie jouée, chaque partie donnant accès à son dernier coup joué et la partie précédente,
- liste chaînée de coups par partie, avec, pour chaque coup, accès au coup précédent.

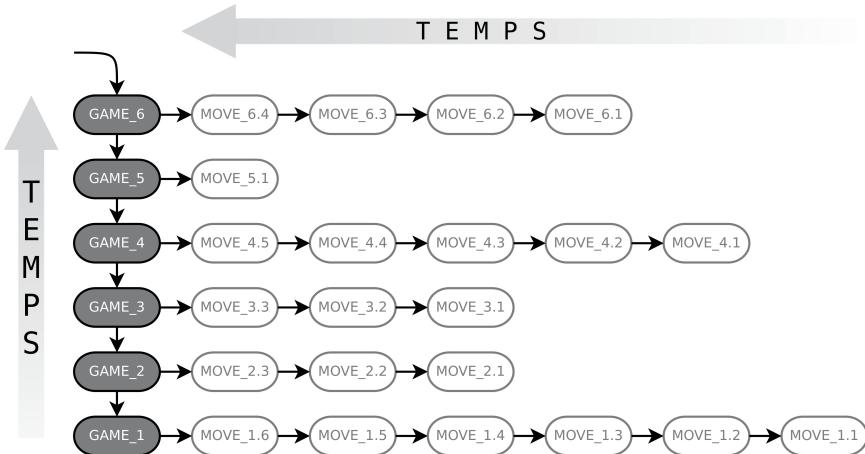


FIG. 3.17 – Représentation de la mémoire, sous la forme d'une double liste chaînée avec accès par ordre chronologique inversé

3.5.3 Mémoire sémantique

La mémoire sémantique permet le stockage :

- d'objets, qui correspondent aux plateaux rencontrés lors des parties,
- d'attributs, qui sont des formes remarquables ajoutées au fil du temps par le module d'introspection,
- des relations entre les objets et les attributs, qui signalent la présence de la forme dans le plateau.

La structure d'une telle mémoire peu être vue comme :

- une matrice de booléens, chaque ligne correspondant à un objet (plateau), et chaque colonne à un attribut (forme remarquable), $(x, y) = \text{TRUE}$ signifiant la présence de la forme y dans le plateau x ,

	ATTR_1	ATTR_2	ATTR_3	ATTR_4	ATTR_5
OBJ_1	●	●		●	
OBJ_2	●		●	●	●
OBJ_3	●	●	●		
OBJ_4		●		●	●
OBJ_5	●		●		●
OBJ_6				●	
OBJ_7	●	●	●	●	
OBJ_8		●	●	●	

FIG. 3.18 – Représentation de la mémoire sémantique sous forme matricielle

- un graphe biparti où la présence d'un arc entre x et y signifie que la forme y est présente dans le plateau x .

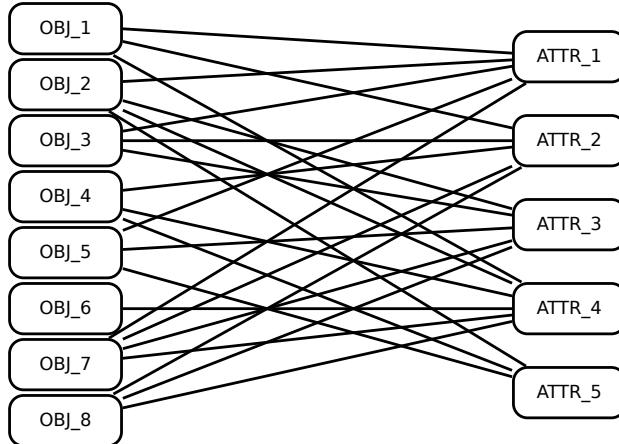


FIG. 3.19 – Représentation de la mémoire sémantique sous la forme d'un graphe biparti

3.6 Méthode de travail

Cette première phase d'analyse a consisté en l'étude du modèle de Conscience Artificielle proposé par Guillaume Tisserant et son équipe. Chaque membre du groupe a commencé par examiner le document. La majorité des décisions prises

par rapport aux contraintes relevant du projet dans sa globalité, nous avons choisi de traiter ces questions collectivement. De nombreuses réunions ont donc été organisées entre les membres de l'équipe, au moins deux fois par semaine, et avec nos encadrants, environ une fois par semaine.

Deuxième partie

Développement

Chapitre 4

Spécifications techniques

Sommaire

4.1	Environnement	49
4.1.1	Aperçu globale	49
4.1.2	Bibliothèque <code>game_logic</code>	50
4.1.3	Serveur <code>game_service</code>	52
4.1.4	Clients	55
4.2	L'agent « Cogito »	57
4.2.1	Représentation des connaissances	57
4.2.2	Le package FOL	57
4.2.3	Spécifications des classes <code>Cbs</code> et <code>Rpbs</code>	58
4.3	Analyse	58
4.3.1	L'analyseur conceptuel de base	58
4.3.2	L'analyseur conceptuel poussé	59
4.4	Raisonnement	59
4.4.1	Introspection	60
4.4.2	Valuation des formes	62
4.4.3	Moteur de choix	63
4.5	Mémoire	63
4.5.1	Interface Mémoire	63
4.5.2	Le SGBD Neo4j	64
4.5.3	Éléments en mémoire	65
4.5.4	Mémoire sémantique	66
4.5.5	Mémoire épisodique	67
4.5.6	Vision globale de la mémoire	67

4.1 Environnement

4.1.1 Aperçu globale

Ce que nous appelons système est en réalité une assemblage de plusieurs logicielles qui communiquent entre eux :

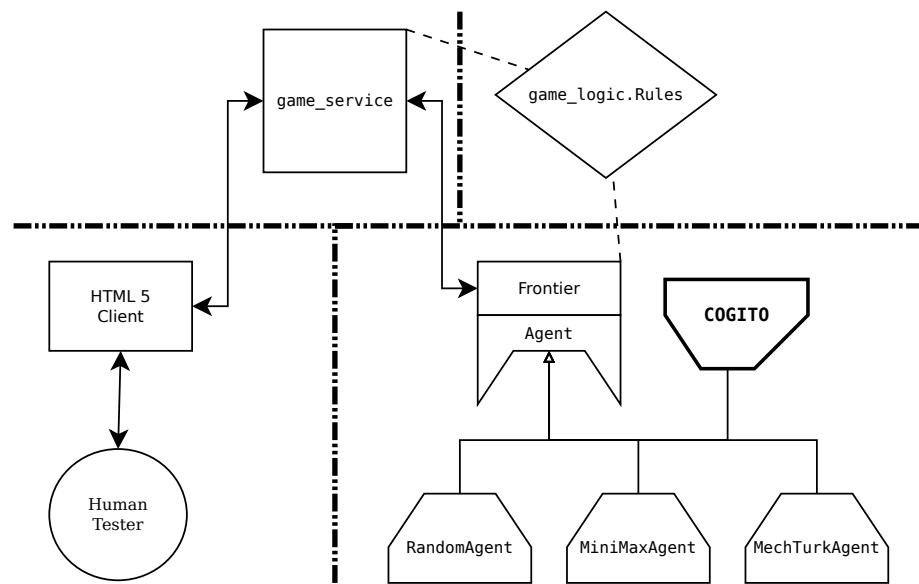
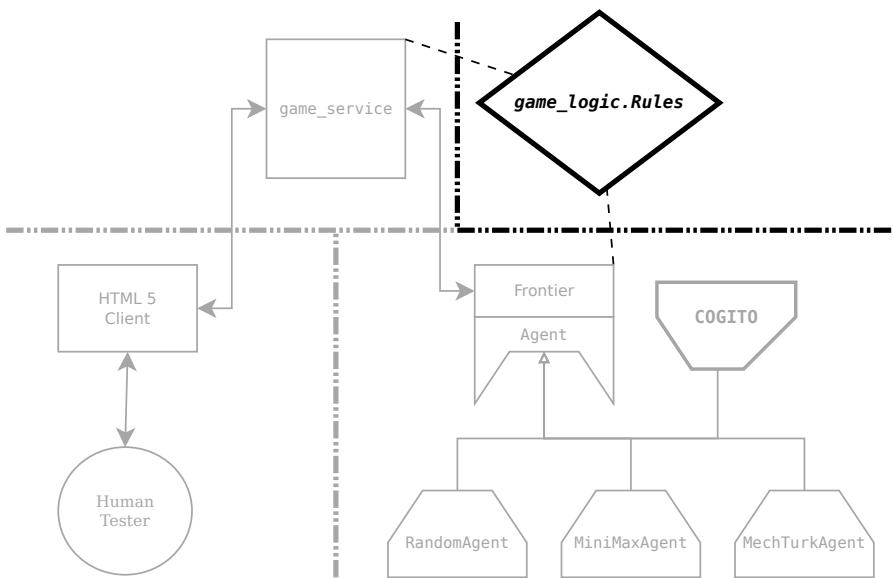


FIG. 4.1 – Aperçu global du système

Avant de se concentrer sur *COGITO* lui-même nous expliquerons rapidement l’implémentation de l’infrastructure de test, que nous pouvons voir comme l’environnement.

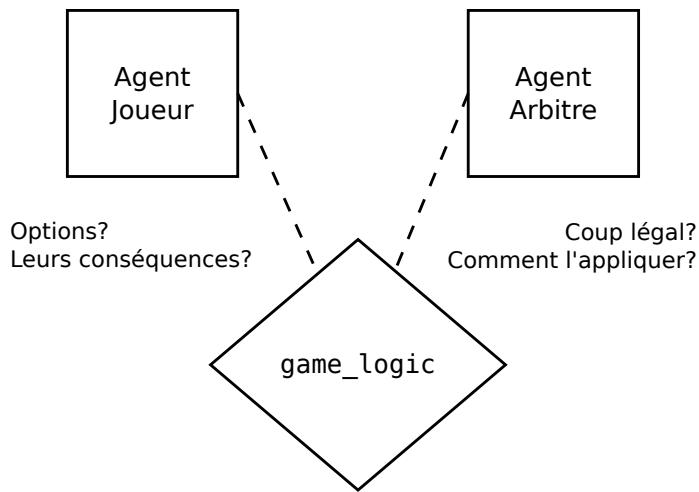
4.1.2 Bibliothèque `game_logic`

Dans le chapitre 3.1.1 nous avons introduit le module conceptuel *Rule Book*, qui génère un ensemble de coups possibles, chacun étant associé à son plateau résultant. Pour ce faire ce module doit pouvoir décider de la légalité d’un coup et connaître ses conséquences.

FIG. 4.2 – Rôle de `game_logic` dans le système

Nous avons également parlé dans le chapitre 3.2 d'un agent « Arbitre » maître du jeu qui a besoin d'un ensemble de règles afin d'initialiser le plateau et d'appliquer ou de refuser respectivement les coups légaux et illégaux.

Il est clair que ces deux objets partagent un même ensemble de fonctions et de structures. Nous avons donc fait le choix de les factoriser dans une bibliothèque que nous appellerons `game_logic`.

FIG. 4.3 – Partage de la bibliothèque `game_logic`

Cette bibliothèque contient trois classes principales :

- `BoardMatrix` : plateau sous forme matricielle avec accesseurs adaptés,

- **Rules** : interface implémentée par chaque jeu. Ses méthodes permettent de connaître :
 - * la forme du plateau et sa configuration initiale,
 - * qui joue en premier,
 - * quand la partie est gagnée ou perdue et par qui, quand le match est nul,
 - * les coups possibles pour un joueur donné.
- **Game** : associe un objet **Rules**, un objet **BoardMatrix**, un état et un joueur courant.

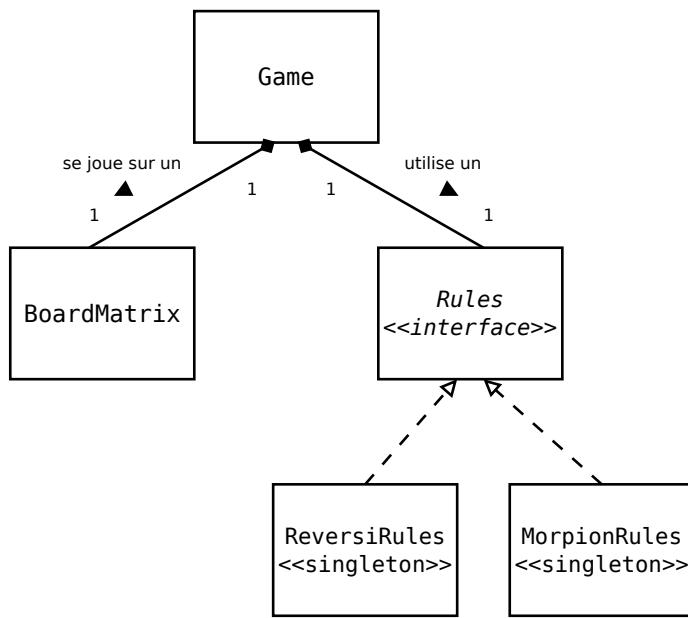
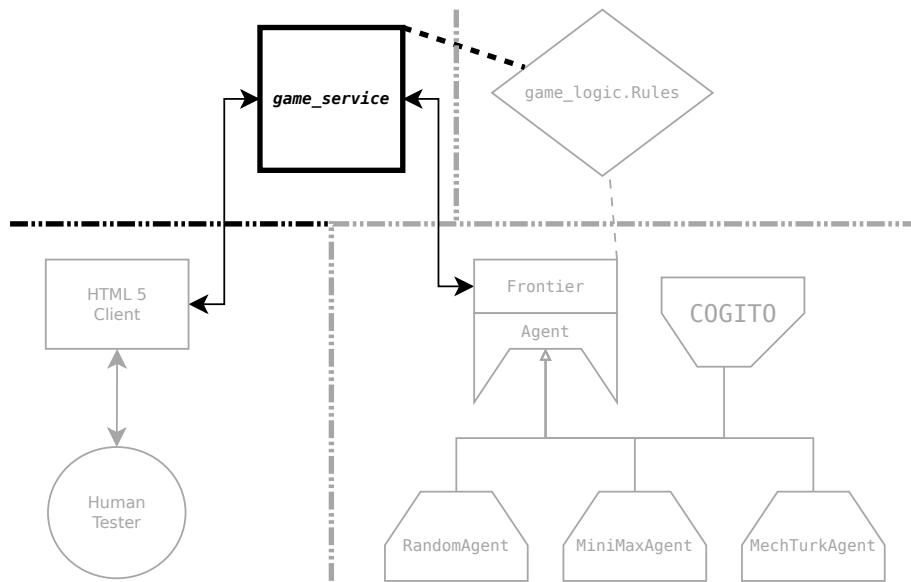


FIG. 4.4 – Classes de la bibliothèque `game_logic`

4.1.3 Serveur `game_service`

Le chapitre 3.2 conclu en proposant le modèle client-serveur pour les communications entre l'*agent-joueur* et l'*agent-arbitre*.

FIG. 4.5 – Rôle de `game_service` dans le système

L’arbitre et l’environnement, c’est à dire l’état du jeu, seront hébergés dans une application à part entière : un « web service » appelé `game_service`.

Transfert des percepts

Le client-joueur à tout d’abord besoin de percevoir son environnement, c’est à dire de connaître l’état courant du jeu. Il est envoyé par le serveur sous forme d’un document XML¹ grâce au protocole HTTP². Pilier du web, l’HTTP a l’avantage d’être très répandu, ubiquité qui assure l’existence de bibliothèques ouvertes, complètes, bien documentées et simples d’utilisation pour tous langages et plateformes. Ce choix nous a permis de faciliter le développement du serveur indépendamment du reste du système.

Transfert des actions

Le client a également besoin d’envoyer ses coups au serveur. Pour ceci il suffit de paramétriser la requête pour indiquer :

- celui qui joue,
- une ligne,
- une colonne.

Il nous est désormais possible d’interagir avec le serveur en écrivant manuellement nos coups dans la barre d’adresse et en lisant le document XML renvoyé. Cette manière de jouer est peu ergonomique, mais elle nous a permis de tester le

¹XML : Extensible markup language.

²HTTP : Hyper-text transfer protocol.

fonctionnement du serveur dans son intégralité bien avant la construction d'une interface graphique.

À noter également qu'il n'y a pas d'authentification ou de gestion des utilisateurs. Il est donc possible de « tricher » en prenant le tour de l'autre joueur : nous supposons ici qu'il n'y a simplement pas de tricheur. Nous avons choisi d'ignorer ces problèmes de sécurité dans le but de se concentrer sur d'autres parties du système plus en rapport avec notre sujet. Il serait cependant simple d'ajouter une identification via des sessions et des cookies.

Constitution des réponses

Pour analyser ces requêtes paramétrées et générer une réponse, la technologie « Java Servlet » est utilisée. Une classe Java est instanciée par un serveur *Tomcat* pour traiter une seule requête : l'objet est détruit immédiatement après avoir envoyé sa réponse sous la forme d'un document XML.

Cette technologie orientée prototypage rapide est très simple à utiliser mais nous permet pourtant de gérer de multiples requêtes concurrentes, donc lancer un très grand nombre de jeux entre différentes paires de clients. Il suffit d'ajouter un paramètre à la requête pour préciser l'identificateur du jeu, et un singleton côté serveur pour stoker les jeux et gérer les accès concurrents.

Conclusion

Le serveur correspond finalement un service web, c'est à dire un programme fournissant des données destinées à être lues par d'autres programmes et non des humains.

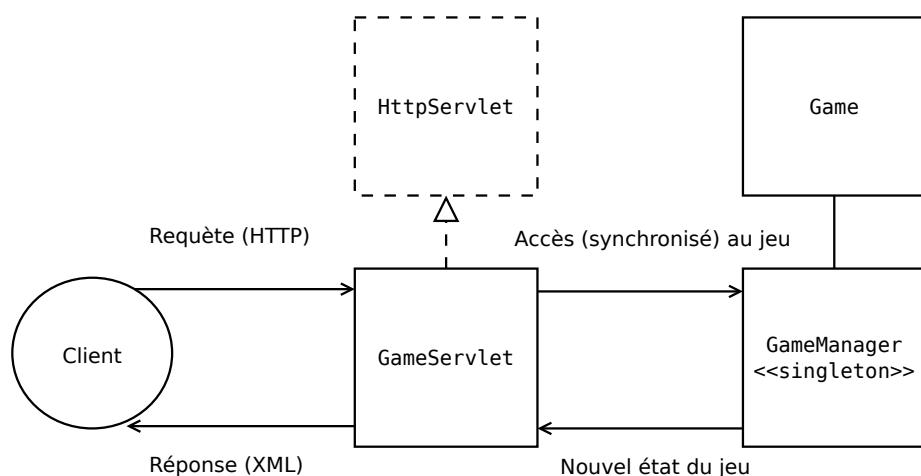


FIG. 4.6 – Classes du Service Web `game_service`

Le `GameManager` sert de base de données temporaire : c'est un conteneur de `Game`, classe de la bibliothèque `game_logic`. Chaque `Game` possédant une

référence vers le singleton `Rules`, il est possible pour le serveur d'arbitrer tous les parties simultanément.

4.1.4 Clients

Client humain

Le client humain est notre plateforme de visualisation des performances de *COGITO*.

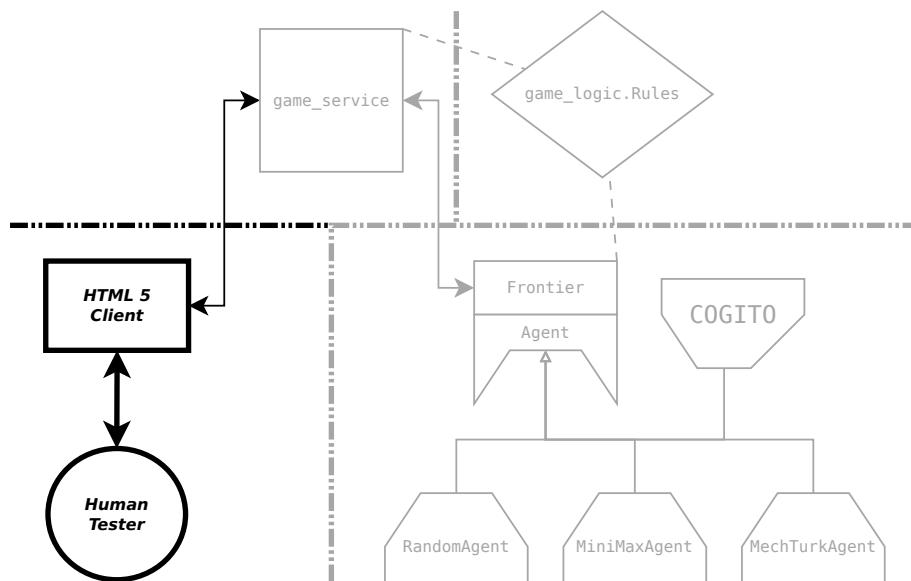


FIG. 4.7 – Rôle du client humain dans le système

L'homme se repose sur sa vision pour comprendre le monde. Posséder une sortie graphique prototypique au début du développement a facilité le développement et le débogage. L'HTML 5 a comme avantage d'être un standard multi-plateforme et non-compilé. Seul un navigateur web récent est nécessaire.

« HTML 5 » est plus qu'un ensemble de balises HTML. Il s'agit surtout d'une bibliothèque `javascript` permettant d'afficher du contenu dynamique. De plus ce langage étant conçu pour manipuler le DOM³ il est très facile d'analyser les réponses XML du serveur. Couplé avec l'implémentation AJAX⁴ de la bibliothèque `jquery` nous pouvons facilement interagir avec notre serveur via des requêtes HTTP formatées en XML.

En résumé, nous nous sommes basés sur des standards Libres qui nous ont donné accès à de nombreuses ressources (librairies, documentation, etc.) du à l'omniprésence de ces technologies dans le monde du web. Ce choix nous a permis de focaliser notre énergie sur les éléments plus pertinents avec notre sujet.

³DOM : Document Object Model.

⁴AJAX : acronyme d'Asynchronous Javascript and XML.

Client machine

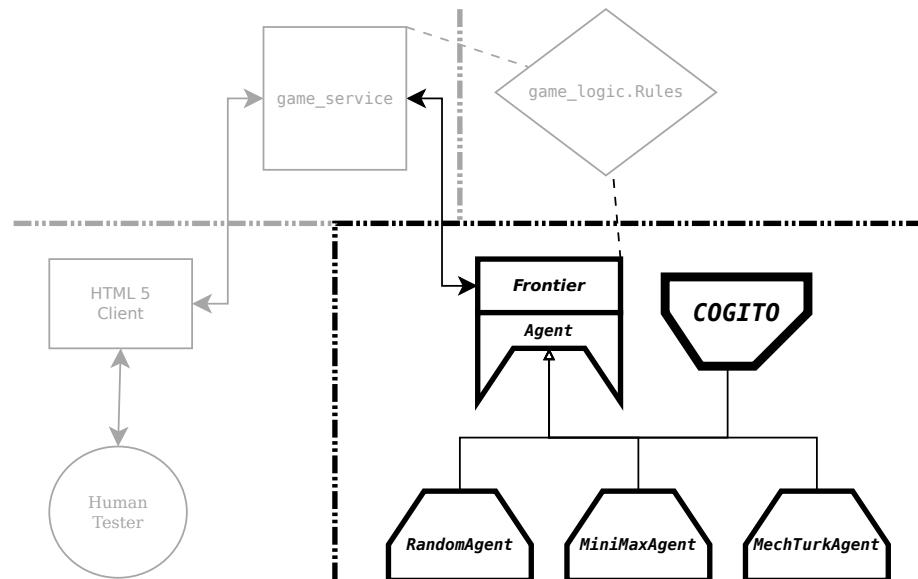


FIG. 4.8 – Rôle de l'agent-client dans le système

Classe Frontier

La frontière est un module purement réactif composé d'un **Actuator**, traduisant les objets **Action** qu'il reçoit en requêtes HTTP, et d'un **Sensor**, qui transforme les réponses XML en objets **Percept** dont le plus important est le percept **Choices**.

En claire ce sont des traducteurs : le **Sensor** traduit du langage **Agent** au langage arbitre, l'**Actuator** l'inverse. La **Frontier** comporte aussi une sorte de mémoire sensorielle lui permettant de juger si son environnement a changé. Nous ne passons donc que les percepts intéressants aux couches plus hautes dans la hiérarchie.

Classe abstraite Agent

Un **Frontier** est toujours encapsulé par un **Agent** et lui sert d'outil. Cette classe **Agent** est instanciée au démarrage du client machine. Le fil d'exécution principal alterne entre demandes de mises à jour du serveur, réflexion et choix de coup. C'est une classe purement abstraite, car l'implémentation de ces deux dernières fonctionnalités varie énormément d'un algorithme à un autre. Parmi nos agents-paramètres se trouvent :

- **MinimaxAgent** : qui applique la stratégie de maximisation de gain minimale,
- **RandomAgent** : qui joue au hasard,
- **MechTurkAgent** : qui exhibe une intelligence quasi-humaine⁵,

⁵En réalité c'est bien un utilisateur qui joue via la console.

- **Cogito** : notre cognition artificielle qui apprend à jouer en s'exerçant contre les autres.

Notons que rien n'empêche les confrontations entre deux clients machines avec des IA différentes, un client humain et un client machine ou bien deux clients humains.

4.2 L'agent « Cogito »

Jusqu'à présent nous nous sommes intéressés au système d'un point de vue globale. Nous passons désormais à une analyse détaillée de l'agent *COGITO*.

4.2.1 Représentation des connaissances

Les connaissances sont décrites en logique du premier ordre. D'une part le plateau sera représenté comme un ensemble de faits et d'autre part les « formes » seront exprimées par des règles dont l'hypothèse symbolise la sous structure à reconnaître et la conclusion un identifiant de cette forme.

Vocabulaire

- *isMine(x)* : un qui m'appartient occupe la case x ,
- *isOpp(x)* : un qui appartient à l'adversaire occupe la case x ,
- *isEmpty(x)* : il n'y a pas de pion sur la case x ,
- *isEdge(x)* : la case x se trouve au bord du plateau,
- *isCorner(x)* : la case x se trouve dans un coin du plateau,
- *near(x, y)* : les cases x et y sont adjacentes horizontalement, verticalement ou en diagonale,
- *aligned(x, y, z)* : les cases x , y et z sont sur la même ligne, colonne ou diagonale dans cet ordre (ou dans l'ordre inverse).

4.2.2 Le package FOL

Le package **FOL**⁶ contient l'ensemble des classes représentant des informations en logique du premier ordre.

- **CompleteBoardState** : version logique du premier ordre de **BoardMatrix** définie section 4.3 page 51,
- **RelevantPartialBoardState** : classe qui décrit la configuration d'une sous-partie pertinente d'un plateau comme une règle logique. Elle correspond aux « formes » extraites par le module de raisonnement qui doivent être reconnues dans les **CompleteBoardState**,
- **Option_FOL** : c'est la version logique de la classe **Action.Option**,
- **Choices_FOL (FOL_objects)** : version conversion d'un **Percept.Choices** sous forme de logique du première ordre avec, en supplément, une liste de formes pertinentes associée à chaque plateau du paquet.

⁶FOL pour « First Order Logic » (« Logique du Premier Ordre » en Français).

4.2.3 Spécifications des classes Cbs et Rpbs

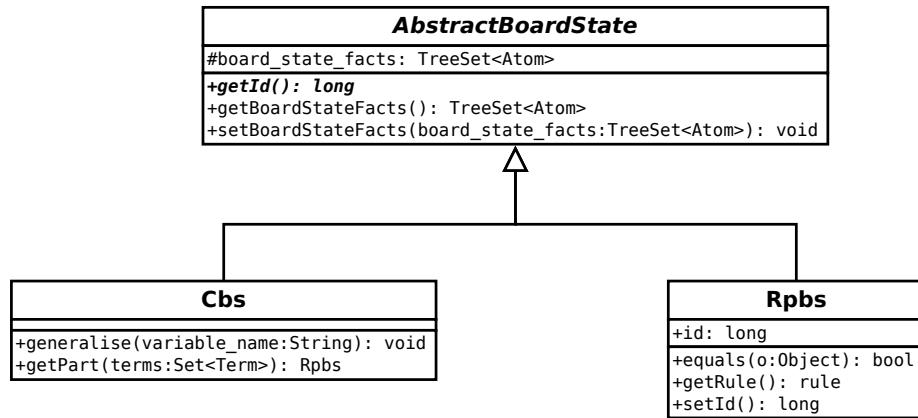


FIG. 4.9 – Diagramme de classe des Cbs et Rpbs

Les classes **Cbs** et **Rpbs** représentent respectivement un plateau et une sous configuration d'un plateau dans le format logique décrit dans la section 4.2.1 page précédente.

4.3 Analyse

Le rôle de l'analyseur est de traduire la représentation de l'environnement dans un format compréhensible par le système pour ensuite y reconnaître des formes connues. Pour ce faire, le module utilise la bibliothèque **game_logic** paquetage **agent**. Les classes principales utilisées sont :

- **Percept.Choices** (**agent**),
- **BoardMatrix** (**game_logic**),
- **Choices_FOL** (**FOL_objects**),
- **CompleteBoardState** (**FOL_objects**),
- **RelevantPartialBoardState** (**FOL_objects**),

Le module est divisé en deux parties : l'*analyseur conceptuel de base* et l'*analyseur conceptuel poussé* décrites ci-après.

4.3.1 L'analyseur conceptuel de base

L'analyseur conceptuel de base (comme défini dans la partie 3.3.1, page 36) convertit une instance de **Percept.Choices** en une instance de **Choices_FOL** en se servant de classes qui modélisent la logique du premier ordre.

Une première analyse de plateau au format matriciel (**BoardMatrix**) génère l'ensemble des faits logiques permettant de représenter sa configuration. Cet ensemble constitue une base de faits qui est ensuite stockée comme attribut de

la l'objet `CompleteBoardState` correspondante à ce plateau. L'analyseur crée alors un instance `Choices_FOL` correspondant à l'instance `Percept.Choices` fournit par le client machine en convertissant chaque plateau (`BoardMatrix`) de `Percept.Choices` en un plateau (`CompleteBoardState`) de `Choices_FOL`. Elle passe ensuite ce paquet à l'analyseur conceptuel poussé.

4.3.2 L'analyseur conceptuel poussé

Lorsque l'analyseur conceptuel poussé, comme défini dans la partie 3.3.2(page 36), reçoit ce paquet, il associe à chaque plateau l'ensemble les formes pertinentes reconnues (des `RelevantPartialBoardState`).

Les formes (`RelevantPartialBoardState`) récupérées de la mémoire correspondent à des règles logiques représentées comme une conjonction d'atomes, le dernier atome étant la conclusion de cette règle. Ici, l'hypothèse décrit une configuration (forme) pertinente, et la conclusion introduit un identifiant. Par exemple, le fait d'avoir pris un coin est représenté par la règle :

$$isCorner() \wedge is(Mine) \implies _rpb034.$$

Ensuite, l'analyseur, en tant que moteur d'inférence, sature la base de faits de chaque plateau rencontré dans le paquet `Choices_FOL` par l'application de l'ensemble de ces règles (`RelevantPartialBoardState`). La reconnaissance des formes dans chacun de ces plateaux revient alors à rechercher l'homomorphisme de l'atome représentant l'identifiant d'une forme pertinente dans la base de faits saturée de ce plateau. Une fois qu'une forme est reconnue (existence d'un homomorphisme), la règle `RelevantPartialBoardState` est ajoutée à la liste de formes pertinentes associée à ce plateau.

Pour résumer, le rôle de l'analyseur conceptuel poussé est donc de déterminer et d'ajouter cette liste de formes pertinentes à chaque plateau présent dans le paquet `Choices_FOL`.

L'analyseur passe enfin ce paquet enrichi à la mémoire et stimule le module de raisonnement afin que celui-ci commence son processus.

4.4 Raisonnement

Le module de raisonnement a été implémenté en trois classes comme le montre la figure 4.10 page suivante. La classe `ReasoningEngine` sert d'interface avec les autres modules de *COGITO*.

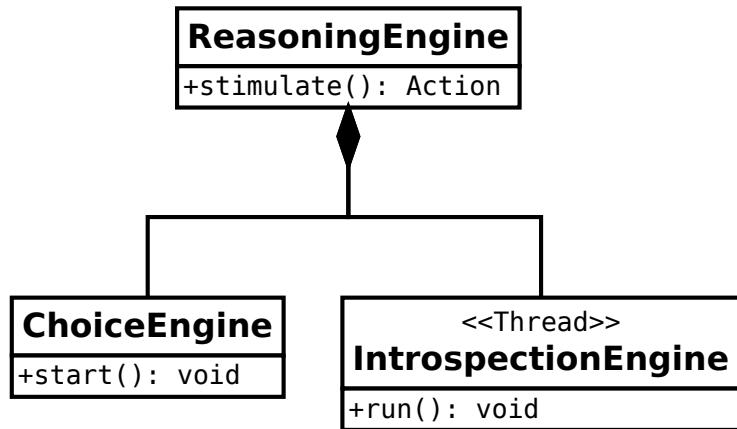


FIG. 4.10 – Diagramme de classe du module de raisonnement

4.4.1 Introspection

Le module d'introspection est implémenté en tant que *thread*, ce qui lui permet de pouvoir s'exécuter en continue. On s'éloigne donc de l'analyse décrite section 3.4.1 page 37. Ce module implémente la recherche de nouvelles formes, pour ce faire il implémente les algorithmes détaillés ci-après.

L'algorithme 1 page ci-contre sert de point d'entrée du moteur d'introspection. Il est appelé indéfiniment dans une boucle « tant que » lorsque le moteur

d'introspection est activé.

Algorithme 1: searchNewRpbs

Data : memory

```

1 Cbs cbs1, cbs2;
2 List < Rpbs > list;
3 Game game1, game2;
4 if random() > 0.5 then                                /* Won games */
5   | game1 = memory.getRandomWonGame();
6   | repeat
7   |   | game2 = memory.getRandomWonGame();
8   | until cbs1 ≠ cbs2;
9 else                                                 /* Lost games */
10  | game1 = memory.getRandomLostGame();
11  | repeat
12  |   | game2 = memory.getRandomLostGame();
13  | until cbs1 ≠ cbs2;

14 cbs1 = game1.getLastCbs();
15 cbs2 = game2.getLastCbs();
16 while (cbs1 = cbs1.getPrevious()) ≠ nil &&
17 (cbs2 = cbs2.getPrevious()) ≠ nil do
18   | list = findCommonRpbs(cbs1,cbs2);
19   | forall the rpbs ⊆ list do
20     |   | extendRpbs(rpbs, cbs1, cbs2);

```

L'algorithme 2 recherche les formes identiques associées à deux plateaux.

Algorithme 2: findCommonRpbs

Data :

Cbs cbs1;
Cbs cbs2;

Result : *List* < *Rpbs* >

```

1 List < Rpbs > list;
2 foreach rpbs1 associated with cbs1 do
3   | foreach rpbs2 associated with cbs2 do
4     |   | if rpbs1 == rpbs2 then
5       |         | list.add(rpbs1);

6 return list;

```

Enfin, l'algorithme 3 page suivante cherche une nouvelle forme à partir de

deux plateaux et d'une forme connue (et contenue) dans les deux plateaux.

Algorithm 3: extendRpbs

Data :

Rpbs rpbs;

Cbs cbs1;

Cbs cbs2;

Memory memory;

```

1 Rpbs new_rpbs;
2 list < Term > choices;
3 list < Term > used_terms;
4 Term t;
5 List < Substitution > substitution_list1 =
   cbs1.getHomomorphisms(rpbs);
6 List < Substitution > substitution_list2 =
   cbs2.getHomomorphisms(rpbs);
7 foreach substitution1 ⊆ substitution_list1 do
        /* On recherche un terme qui, si il est instancié,
       transforme un atome partiellement instancié en un atome
       complètement instancié. */
8   choices = cbs1.getChoices(rpbs.getTerms());
9   while choices.size() > 0 do
10    t = choices.getRandom();
11    choices.remove(t);
12    used_terms.clear();
13    used_terms.add(rpbs.getTerms());
14    used_terms.add(t);
        /* Création d'un nouveau rpbs */ new_rpbs =
        cbs1.getPart(used_terms);
15   foreach substitution2 ⊆ substitution_list2 do
16     if cbs2.existsHomomorphisms(new_rpbs, substitution_list2)
         then
17       if !memory.contains(new_rpbs) then
18         memory.putRpbs(new_rpbs);

```

19 **return** *list;*

4.4.2 Valuation des formes

Afin d'attribuer un poids à chacune des formes présentes en mémoire, nous avons choisi de nous appuyer sur la théorie de la probabilité. Nous cherchons donc, pour chaque forme, à déterminer la probabilité qu'une forme remarquable soit présente dans un jeu gagnant :

$$P(Gain|Forme)$$

En s'appuyant sur le théorème de Bayes, nous obtenons la formule suivante :

$$P(Gain|Forme) = \frac{P(Forme|Gain) \times P(Gain)}{P(Forme)}$$

Pour des raisons de simplicité, cette valuation est implémentée dans le module de mémoire et est exécutée par l'appel à la méthode `endOfGame`. Cette méthode prend en paramètre le statut de la partie, à savoir : « VICTORY », « DEFEAT », « DRAW » et « INTERRUPTED ».

4.4.3 Moteur de choix

Le moteur de choix est implémenté à travers la classe `ChoiceEngine`. Celle-ci est extrêmement simple. Dans un premier temps, la liste des plateaux possibles est récupérée en mémoire sous la forme d'une `List<Option_FOL>`. Ensuite, pour chaque plateau un poids est calculé, celui-ci correspond à la moyenne des poids de chaque `Rpbs` qui lui est associé. Finalement, le plateau ayant le poids le plus élevé est choisi. Si deux plateaux ont un poids maximum, le plateau choisi est tiré aléatoirement

4.5 Mémoire

4.5.1 Interface Mémoire

Le module de mémoire a été conçu de manière générique à l'aide d'interfaces java. Pour cela elle se décompose en trois parties :

- l'interface principale de la mémoire (`Memory`), qui déclare les méthodes appelées par les autres modules (Analyse et Raisonnement), et qui est implémentée en tant que mémoire à court terme (`ActiveMemory`). Cette implémentation joue un rôle de *buffer* en stockant temporairement les options possibles fournies par le *module d'analyse* et récupérées par le *module de raisonnement*. Elle est également composée d'une mémoire épisodique et d'une mémoire épisodique,
- l'interface de la mémoire épisodique (`EpisodicMemory`) et l'interface de la mémoire sémantique (`SemanticMemory`), qui déclarent les méthodes appelées par les implémentations de l'interface `Memory`.

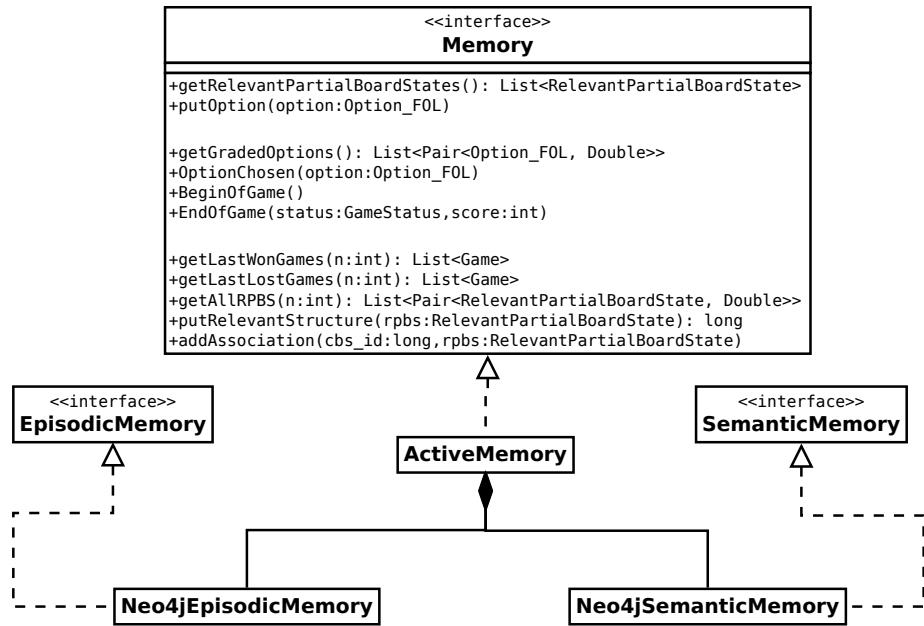


FIG. 4.11 – Diagramme de classe du module mémoire simplifié

La mémoire assure également la persistance des données, qui se fait via un module de persistance qui est utilisé les implémentations `Neo4jEpisodicMemory` et `Neo4jSemanticMemory`.

4.5.2 Le SGBD Neo4j

La persistance des données est assurée par un SGBD né de la mouvance NoSQL : Neo4j. Il permet la gestion d'une base de données orientée graphe. Nous avons fait le choix d'utiliser un tel système pour plusieurs raisons :

- nous avions la volonté de **découvrir une solution NoSQL**, que nous n'avons pas eu l'occasion d'étudier lors de notre formation,
- Neo4j est disponible en **plusieurs versions**, notamment en version serveur, version webservice REST et version java embarquée. La mémoire ne nécessitant pas d'accès concurrents mais aussi par soucis de légèreté, c'est la version embarquée (*embedded java*) qui a été utilisée,
- ce type de SGBD NoSQL permet d'obtenir des **temps d'accès** plus rapides qu'avec des SGBD relationnels traditionnels,
- la vision graphe de la base de données est **adaptée** à la conception de notre mémoire⁷,
- cette solution conserve les **propriétés ACID** des transactions des SGBD relationnels traditionnels,
- la **documentation complète** et la **communauté active** permettent de s'initier très rapidement à cette nouvelle technologie,

⁷Notons tout de même qu'il aurait été possible de stocker les données sous forme de tables relationnelles classiques.

- Neo4j est une **solution libre** distribuée sous licence GPLv3.

Neo4j étant un SGBD NoSQL orienté graphe, la base de donnée est représentée sous la forme d'un graphe orienté, composé d'un nœud « root ». Chaque nœud et chaque arc peut posséder des attributs. Cependant, il n'est possible de typer que les arcs. Pour typer les noeuds une astuce consiste en la création d'un « master_node » comme le montre la figure 4.12.

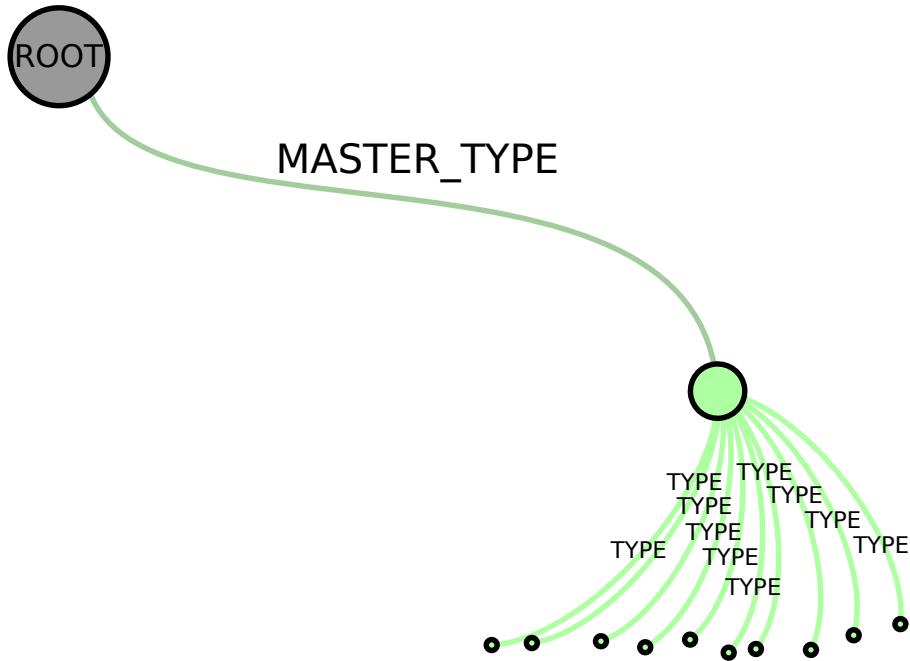


FIG. 4.12 – Exemple de typage des nœuds avec Neo4j

4.5.3 Éléments en mémoire

Le module de persistance permet le stockage de la mémoire épisodique et sémantique, les deux étant liées par les relations entre les *MOVE* et les *OBJECTS*. Nous avons donc besoin des types suivants :

Types de nœud :

- GAME : une partie en mémoire épisodique,
- MOVE : un coup joué en mémoire épisodique,
- OBJECT : un plateau en mémoire sémantique,
- ATTRIBUTE : une forme remarquable en mémoire sémantique.

Types de liens :

- Relations maîtres
 - * MASTER_ATTR,
 - * MASTER_OBJ,
 - * MASTER_GAME,
 - * MASTER_MOVE.

- Relations de type
 - * ATTRIBUTE,
 - * OBJECT,
 - * MOVE,
 - * GAME,
 - * LAST_GAME (relation GAME spéciale, signifiant que cette partie est la dernière jouée).
- Relations en mémoire épisodique
 - * PREV_GAME : permet à partir d'une partie d'accéder à la précédente,
 - * LAST_MOVE : permet à partir d'une partie d'accéder au dernier coup joué,
 - * PREV_MOVE : permet à partir d'un coup d'accéder au précédent,
 - * STATE_BOARD : permet de lier un coup à l'état de plateau correspondant.
- Relations en mémoire sémantique
 - * RELATED : décrit la présence d'une forme remarquable (ATTRIBUTE) dans un plateau (OBJECT).

4.5.4 Mémoire sémantique

Comme vu dans la partie 3.5.3 (page 43), la mémoire sémantique stocke une matrice de booléens ayant comme attributs (colonnes) des formes remarquables, et comme objets (lignes) des plateaux.

On peut voir sur la figure 4.13 les deux « master_node » permettant de déclarer des nœuds d'attributs et d'objets. Ces nœuds sont en liés via des relations de type *RELATED*.

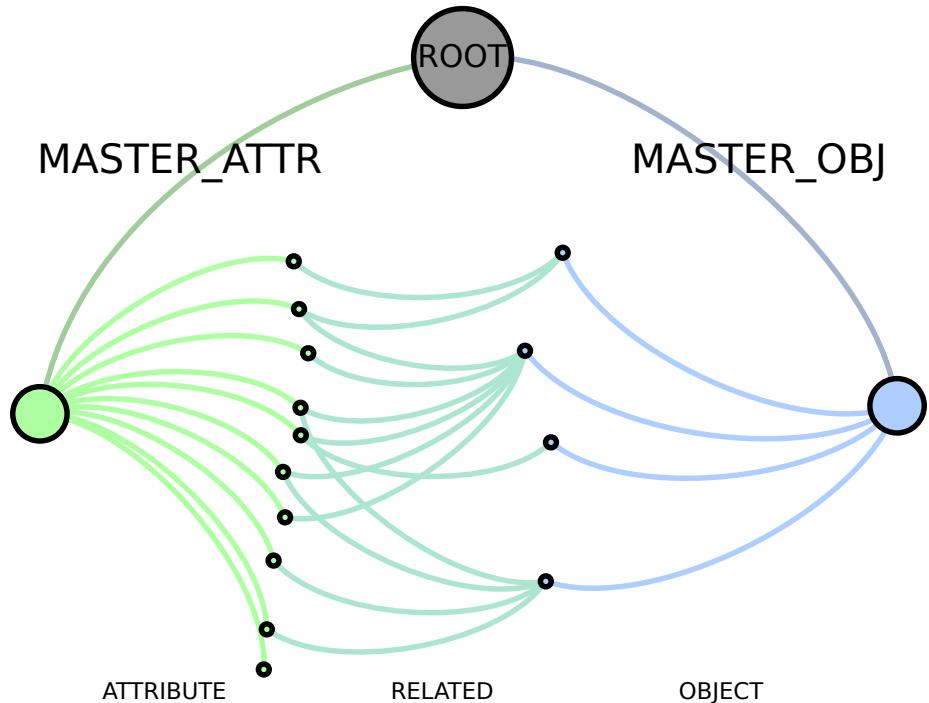


FIG. 4.13 – Représentation de la mémoire sémantique dans Neo4j

4.5.5 Mémoire épisodique

La représentation sous forme de graphe se prête parfaitement à la mémorisation des parties et des coups sous la forme d'une double liste chaînée. La figure 4.14 représente la mémoire épisodique telle qu'elle est stockée dans Neo4j.

On visualise les trois « master_node » permettant de typer les GAME, MOVE et ATTRIBUTES. Le parcours des parties se fait via la relation PREV_GAME et celui des coups via la relation PREV_MOVE.

On remarque également que :

- le « master_node » GAME possède une relation sortant LAST_GAME permettant l'accès au dernier jeu joué,
- les relations BOARD_STATE permettent de faire la liaison entre la mémoire sémantique et la mémoire épisodique.

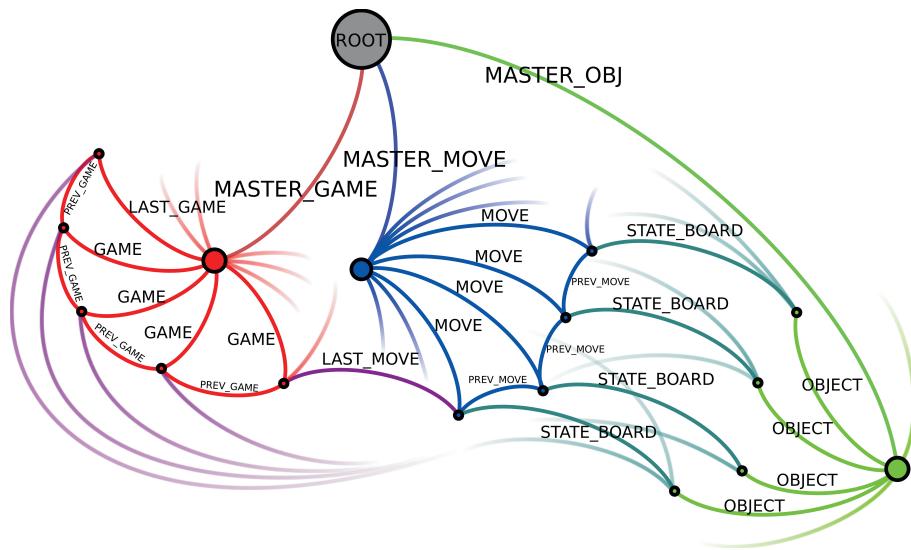


FIG. 4.14 – Représentation de la mémoire épisodique dans Neo4j

4.5.6 Vision globale de la mémoire

La figure 4.15 présente le stockage de la mémoire épisodique et de la mémoire sémantique dans Neo4j. Les relations de type BOARD_STATE assure la liaison entre ces deux mémoires.

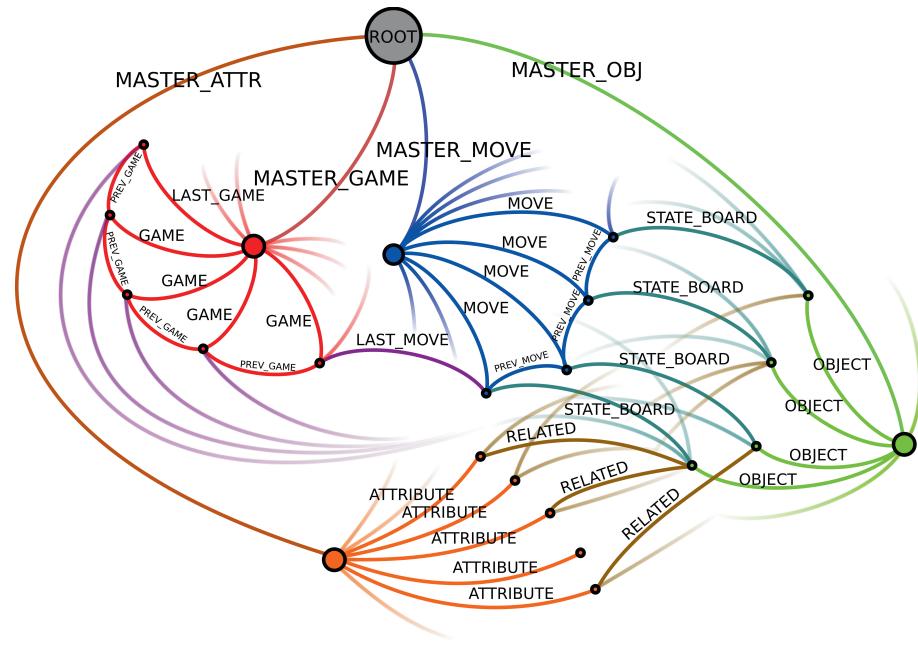


FIG. 4.15 – Représentation complète de la mémoire stockée dans Neo4j

Chapitre 5

Méthode de travail

5.1 Répartition des tâches

Dans le but de pouvoir travailler séparément et de manière autonome sur chaque module lors de l'implémentation, nous avons rigoureusement œuvré sur les spécifications techniques et l'architecture globale de l'application. Les tâches de développement ont ensuite été menées de manière parallèle par chacun des membres de l'équipe :

- Namrata était responsable du module d'analyse,
- Clément du raisonnement,
- Thibaut de la mémoire,
- et William de l'environnement (architecture client-serveur, librairies, etc.).

Nous nous sommes réunis tout au long du développement pour assurer l'avancement global du projet et répondre aux interrogations de chacun.

5.2 Outils utilisés

5.2.1 Gestionnaire de versions

Tout au long du projet, nous avons utilisé *git* : un gestionnaire de versions décentralisé. Un tel outil permet la mise en commun des travaux, la gestion des versions et la gestion des *merges*¹.



Nous avons choisi *git* pour plusieurs raisons :

¹Fusion de deux fichiers différents.

- plutôt simple d'utilisation (par rapport à ses homologues comme *SVN*),
- hébergement gratuit via GitHub (sous condition de diffusion du code sous une licence libre),
- *git* est une solution libre sous licence GPLv2.

5.2.2 EtherPad, un éditeur de texte collaboratif

EtherPad se présente sous la forme d'un éditeur de texte léger permettant de faire un minimum de mise en page.

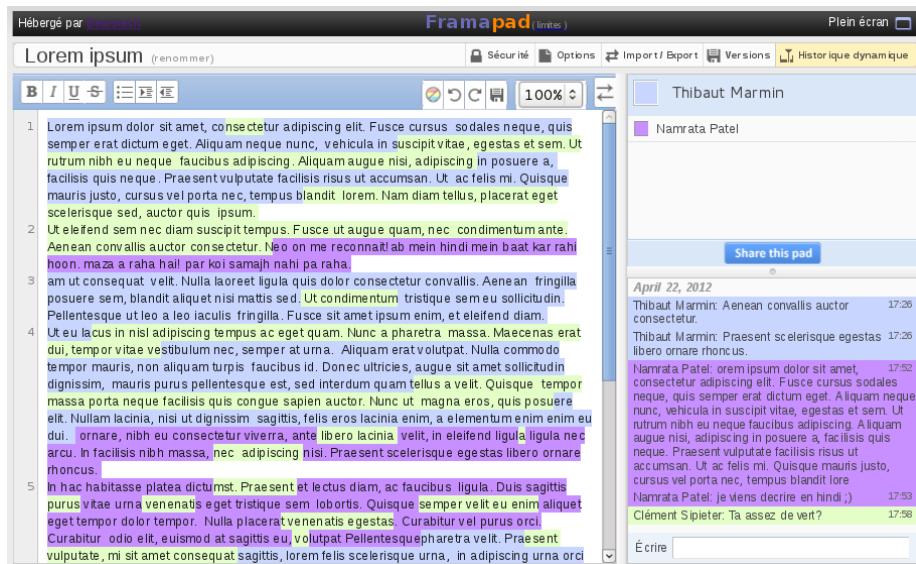


FIG. 5.1 – Aperçu de l'interface d'*EtherPad* hébergé sur *framapad* (site mis à disposition par Framasoft utilisant le code d'*EtherPad*)



L'interface d'*EtherPad* se présente sous la forme d'une interface web (figure 5.1) où chaque utilisateur connecté possède une couleur. La singularité de cet outil réside dans le fait que toutes les modifications effectuées sur le pad sont visibles par tous en temps réel.

Un chat est également disponible ce qui facilite la collaboration entre les utilisateurs connectés.

Il est important de noter qu'*EtherPad* est sous licence Apache v2.

5.2.3 Visualisation de graphes

Pour débugger la base de données Neo4j, nous avons utilisé *Gephi*, un utilitaire basé sur la *NetBeans Platform*.



Il permet de visualiser et de manipuler des graphes. C'est un outil libre qui embarque quelques plugins dont un connecteur Neo4j permettant l'import de bases de données du SGBD.

5.2.4 Outils de développement

Voici une liste non exhaustive des outils que nous avons utilisés lors de nos développements :

Eclipse / NetBeans Il s'agit des deux principaux EDI² libres, permettant une vérification et compilation dynamique du code écrite.

Javadoc Nous avons pris soin de documenter la totalité de notre code dans le but de rendre notre code réutilisable. Javadoc est aujourd'hui devenu un standard industriel utilisé par la majorité des développements Java.

Log4j Librairie Java libre qui permet la journalisation sous formes variées (stdout³, fichiers de log, envoie de mail, etc.).

YourKit Java Profiler Afin de permettre d'optimiser certaines méthodes, nous avons utilisé cet outil en version d'évaluation (sous licence propriétaire). Il s'intègre parfaitement dans la plupart des environnements de développement.

²Environnement de Développement Intégré

³Flux de sortie standard.

Chapitre 6

Conclusion & Perspectives

6.1 Conclusion

La réalisation de ce projet d'Étude et de Recherche nous a permis de vivre une expérience enrichissante tout d'abord du point de vue humain, par la réalisation d'un travail collaboratif, mais aussi d'un point de vue pratique par l'utilisation de connaissances acquises au cours des deux premiers semestres de ce master et de la découverte de nouveaux outils.

Ce travail d'équipe a aboutit sur un modèle d'intelligence artificielle ayant une approche cognitive, dont nous avons réalisé une implémentation fonctionnelle capable de jouer et d'améliorer ses performances avec l'expérience. De plus, notre modèle fournit des données exploitables qui nous permettront, par la suite, d'analyser les critères de décision choisis par COGITO afin de procéder à son évaluation.

6.2 Perspective

6.2.1 Un treillis en mémoire ?

Comme nous l'avons vu dans la partie 3.5.3 (page 43), la mémoire sémantique entrepose une matrice de booléens associant des formes remarquables à des plateaux. Ces associations correspondent typiquement à la définition d'un contexte, permettant la construction d'un treillis de concepts.

Analyse de concepts formels

FCA¹ est l'étude de concepts définis de manière formelle, via un contexte. Cette section ne présente pas en détail FCA. Pour plus d'informations sur cette

¹Formal Concept Analysis (en Français « Analyse de Concepts Formels »).

méthode d'analyse et les treillis de Galois nous vous recommandons la lecture des cours de Marianne Huchard et de Michel Liquière².

Le contexte Il s'agit d'un triplet (G, M, I) avec G et M des ensembles et $I \subseteq G \times M$ des relations de G dans M . Les éléments contenus dans G sont appelés *objets* et ceux de M *attributs*. I est une relation entre un object de G et un attribut de M , et qui se dit « l'object g possède l'attribut m ». (Source : Wikipedia³)

En application à *COGITO* nous aurions :

$$\begin{aligned} G &= \text{ensemble des plateaux (objets),} \\ M &= \text{ensemble des formes remarquables (attributs),} \\ I &= \text{ensemble des relations plateau / formes remarquables} \\ &\quad \text{définies dans la matrice actuelle.} \end{aligned}$$

Construire des concepts à partir du contexte formel permettrait de raisonner à un niveau d'abstraction supplémentaire. Actuellement *COGITO* travail sur des formes remarquables, après la construction d'un treillis de Galois, il pourrait raisonner sur des ensembles de formes remarquables.

Et concrètement ? Prenons un exemple simple et parlant pour la plupart des passionnés d'informatique⁴ :

Nous aimons le **café**



Situation *café*

Nous adorons l'**ordinateur**



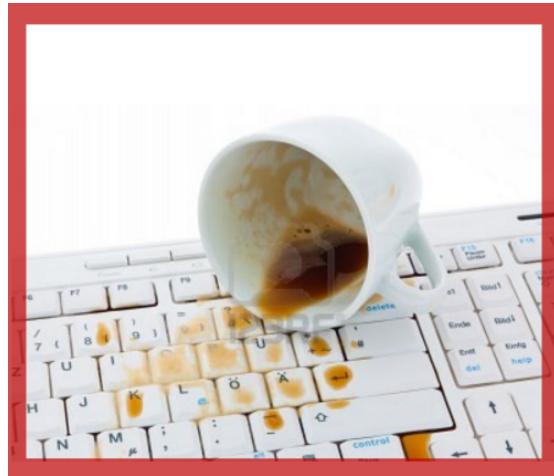
Situation *ordinateur*

En revanche nous n'appréciions pas le café à proximité de l'ordinateur.

²Cours de Marianne Huchard, Professeur en informatique à l'université Montpellier 2 (Faculté des Sciences) et Directrice adjointe du LIRMM (Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier) et de Michel Liquière, Maître de Conférences dans la même université, disponibles à l'adresse suivante : <http://www.lirmm.fr/~huchard/Huchard/teaching.html>

³L'article « Analyse de concepts formels » de Wikipedia en Français à considérablement aidé à la rédaction de cette partie. Contenu sous licence CC-BY-SA.

⁴Alias « Nerd ».



Situation *café renversé sur l'ordinateur*

Le modèle *COGITO* actuel raisonnerait distinctement sur le *café* et sur l'*ordinateur*, qui seraient des attributs. Lors de la rencontre de la situation *café renversé sur l'ordinateur*, les valeur des attributs *café* et *ordinateur* seraient donc diminués. Avec la construction d'un treillis de Galois, il serait possible de dévaloriser le concept regroupant les deux attributs *café* et *ordinateur*, sans nuire aux concepts qui ne sont composé que d'un des deux attributs.

Glossaire

ACID	Propriété ACID d'une transaction : Atomique, Cohérente Isolée et Durable, 64
Action.Option	Pair Action.Move-BoardMatrix correspondant à une action et son résultat, 57
Apache v2	Licence libre en version 2 proposée par la <i>Apache Software Foundation</i> , 70
BoardMatrix	Classe correspondant à un plateau de jeu sous forme matricielle avec un ensemble d'accesseurs adaptés aux jeux de plateau, 57, 58
Cbs	voir CompleteBoardState, 14, 49, 57, 58, 80
Choices_FOL	Version logique de Percept.Choices contenant un Cbs, le plateau courant, et une liste de Option_FOL, 58
client machine	Client HTML implémenté par la classe agent.Frontier qui sert à communiquer avec le serveur jeu pour que nos agents puissent rester synchronisés avec le jeu, 58
CompleteBoardState	Classe correspondant à un plateau du jeu sous forme logique (cf. 4.2.3 page 58), 57, 58
FOL_objects	Package contenant l'ensemble des classes représentant des informations en logique du premier ordre., 58
game_logic	Bibliothèque composé des classes BoardMatrix, Rules et Game, permettant de définir et de gérer des jeux de plateau génériques, 14, 49, 50, 58, 79
GPLv2	<i>GNU General Public License</i> en version 2 (licence libre copyleft), 70
GPLv3	<i>GNU General Public License</i> en version 3 (licence libre copyleft), 64

NoSQL	catégorie de SGBD (récents pour la plupart) qui se différencie du modèle SQL par une représentation des données non relationnelle. La vision NoSQL abandonne certaines fonctionnalités du modèle relationnel standard au profit d'une plus grande scalabilité. NoSQL ne signifie pas <i>No SQL</i> mais <i>Not only SQL</i> (se veut un complément à SQL et non un concurrent), 64
Option_FOL	Version logique de Action.Option contenant une action, son résultat et une liste de formes présentes dans ce plateau résultant, 57
Percept.Choices	Classe héritier de la classe abstraite Percept, contenant un ensemble d'instances de Action.Option, 57, 58
RelevantPartialBoardState	Classe représentant la configuration d'une sous-partie d'un plateau comme une règle(cf. 4.2.3 page 58), 57, 58
Rpbs	voir RelevantPartialBoardState, 14, 49, 57, 58, 63, 80
SGBD	Système de Gestion de Bases de Données, 14, 49, 64

Table des figures

1.1	Schéma du modèle de représentation de la conscience	19
2.1	Modèle restreint déterminé à partir des contraintes énumérée au chapitre 2	26
2.2	Zoom sur le modèle restreint.	27
3.1	Schéma général de <i>COGITO</i>	32
3.2	Diagramme de séquence sommaire de déroulement d'un coup . .	33
3.3	Modèle « Agent et Environnement »	34
3.4	Interactions entre l'Arbitre et les autres agents	35
3.5	Exemple d'environnements	37
3.6	Exemple d'injection (basé sur la figure 3.5)	38
3.7	Représentation graphique de l'environnement	38
3.8	Deux plateaux ayant des formes connues en commun	39
3.9	Illustration de l'injection sur des plateaux	39
3.10	Association formes-annotations	40
3.11	Association formes-annotations basique à l'étape 1	40
3.12	Association formes-annotations multiple à l'étape 2	40
3.13	Association formes-annotations à l'étape 3	41
3.14	Association formes-annotations à l'étape 4	41
3.15	Association formes-annotations à l'étape 5	41
3.16	Représentation graphique de l'environnement	42
3.17	Représentation de la mémoire, sous la forme d'une double liste chaînée avec accès par ordre chronologique inversé	43
3.18	Représentation de la mémoire sémantique sous forme matricielle	44
3.19	Représentation de la mémoire sémantique sous la forme d'un graphe biparti	44

4.1	Aperçu global du système	50
4.2	Rôle de <code>game_logic</code> dans le système	51
4.3	Partage de la bibliothèque <code>game_logic</code>	51
4.4	Classes de la bibliothèque <code>game_logic</code>	52
4.5	Rôle de <code>game_service</code> dans le système	53
4.6	Classes du Service Web <code>game_service</code>	54
4.7	Rôle du client humain dans le système	55
4.8	Rôle de l'agent-client dans le système	56
4.9	Diagramme de classe des <code>Cbs</code> et <code>Rpbs</code>	58
4.10	Diagramme de classe du module de raisonnement	60
4.11	Diagramme de classe du module mémoire simplifié	64
4.12	Exemple de typage des noeuds avec Neo4j	65
4.13	Représentation de la mémoire sémantique dans Neo4j	66
4.14	Représentation de la mémoire épisodique dans Neo4j	67
4.15	Représentation complète de la mémoire stockée dans Neo4j	68
5.1	Aperçu de l'interface d' <i>EtherPad</i> hébergé sur <i>framapad</i> (site mis à disposition par Framasoft utilisant le code d' <i>EtherPad</i>)	70