

# Autour de Metaciv

Gautier Maillé, Thérèse Libourel, Jacques Ferber, Julien Nigon

# Table des matières

<b>I</b>	<b>Généralités</b>	<b>5</b>
<b>1</b>	<b>Introduction</b>	<b>5</b>
<b>2</b>	<b>Méta-Modèle</b>	<b>6</b>
<b>3</b>	<b>Agent</b>	<b>6</b>
3.1	Cognition . . . . .	6
3.1.1	Notion de cognitons . . . . .	6
3.1.2	Catégorisation des cognitons . . . . .	8
3.2	Objets . . . . .	9
3.3	Plan et actions . . . . .	9
3.4	Groupe et rôle . . . . .	9
<b>II</b>	<b>Le logiciel</b>	<b>10</b>
<b>4</b>	<b>Préambule</b>	<b>10</b>
<b>5</b>	<b>Installation</b>	<b>10</b>
<b>III</b>	<b>Documentation des actions</b>	<b>15</b>
<b>6</b>	<b>Les plans</b>	<b>15</b>
<b>7</b>	<b>Les actions A</b>	<b>16</b>
7.1	Les actions relatives aux objets . . . . .	16
7.1.1	A_CreateObject . . . . .	16
7.1.2	A_AddObject . . . . .	16
7.1.3	A_DropObject . . . . .	17
7.1.4	A_Transform . . . . .	17
7.1.5	A_UseObject . . . . .	17
7.1.6	A_AddObjectXCogniton . . . . .	17
7.2	Les actions relatives aux aménagements . . . . .	18
7.2.1	A_CreateFacility . . . . .	18
7.2.2	A_EraseFacility . . . . .	18
7.2.3	A_GoToFacility . . . . .	18
7.2.4	A_UseFacility . . . . .	18

7.3	Les actions relatives aux cognitons . . . . .	18
7.3.1	A_AddCogniton . . . . .	18
7.3.2	A_ChangeCognitonWeight . . . . .	19
7.3.3	A_DoubleCognitons . . . . .	19
7.3.4	A_EmitCogniton . . . . .	19
7.4	Les actions relatives aux déplacements . . . . .	19
7.4.1	A_MoveTowards . . . . .	19
7.4.2	A_MoveRandomly . . . . .	19
7.4.3	A_DoNothing . . . . .	19
7.4.4	A_CreateSettlement . . . . .	20
7.4.5	A_GetAnotherSettlementPatch . . . . .	20
7.4.6	A_GoBackHome . . . . .	20
7.4.7	A_Move . . . . .	20
7.4.8	A_SearchForResources . . . . .	20
7.4.9	A_SmellAndMove . . . . .	20
7.5	Les actions relatives aux agents . . . . .	21
7.5.1	A_GiveBirth . . . . .	21
7.5.2	A_Die . . . . .	21
7.5.3	A_DieIfAttributeUnderZero . . . . .	21
7.5.4	A_ChangeAttribute . . . . .	21
7.5.5	A_Trade . . . . .	21
7.5.6	A_TravelTrade . . . . .	22
7.5.7	A_CreateGroup . . . . .	22
7.5.8	A_HireForRole . . . . .	22
<b>8</b>	<b>Les actions L</b>	<b>22</b>
8.1	Tests . . . . .	22
8.1.1	L_CompareAttribute . . . . .	23
8.1.2	L_CompareObject . . . . .	23
8.1.3	L_CompareResource . . . . .	23
8.1.4	L_IsFacilityHere . . . . .	24
8.1.5	L_IsAnyFacilityHere . . . . .	24
8.1.6	L_OwnFacility . . . . .	24
8.1.7	L_OwnCogniton . . . . .	24
8.1.8	L_OwnObject . . . . .	24
8.2	Algorithmiques . . . . .	24
8.2.1	L_Instant . . . . .	24
8.2.2	L_Random . . . . .	25
8.2.3	L_RandomPercent . . . . .	25
8.2.4	L_Repeat . . . . .	25

<b>IV</b>	<b>Outils de création d'un modèle</b>	<b>26</b>
<b>9</b>	<b>Phase d'initialisation</b>	<b>26</b>
9.1	Peuplements . . . . .	26
9.2	Environnement physique . . . . .	26
9.3	Les objets . . . . .	27
9.4	Les aménagements ou infrastructures . . . . .	30
<b>10</b>	<b>Phase de modélisation</b>	
	Le canevas : cogniton, plan, action	<b>31</b>
<b>V</b>	<b>Exemple de simulation</b>	<b>33</b>
<b>11</b>	<b>Le modèle</b>	<b>33</b>
11.1	Peuplement et environnement . . . . .	33
11.2	Cogniton, Plan, Actions . . . . .	34
<b>12</b>	<b>Observation de la simulation</b>	<b>38</b>
12.1	La fenêtre d'observation d'un agent . . . . .	38
12.2	La fenêtre d'observation générale . . . . .	43

# Première partie

## Généralités

### 1 Introduction

MetaCiv est un framework de développement de simulations d'évolution de sociétés humaines. Il se distingue des plateformes de simulation multi-agent générales telles que Repast, NetLogo, Gama et TurtleKit, en proposant un ensemble de constructions eco-socio-cognitives génériques permettant de gérer des agents vivant socialement dans un environnement physique en évolution.

Les principes structurels ayant guidé la conception de MetaCiv sont les suivants :

1. Intégrer les aspects réactifs et cognitifs des simulations multi-agents, en disposant d'une architecture générique d'agents à base de *cognitons* [?]. Elle se distingue des architectures hybrides en ce sens qu'elle ne différencie pas un niveau réactif et un niveau cognitif mais qu'elle fusionne tous les facteurs aussi bien réactifs que cognitifs permettant à un agent de prendre une décision (croyances, percepts, compétences, motivations, drives, etc.). Les décisions sont prises par un système de combinaison d'influences ou poids qui déterminent la prise de décision.
2. Utiliser les déplacements des agents comme des points d'ancrages de dynamiques émergentes, pour favoriser l'apparition de routes, de places de marchés, de lieux de rencontre, etc. On inclut, lors de la prise de décision des déplacements, des algorithmes de type A\* qui prennent en compte ces lieux et constructions. Ainsi, toute modification du paysage produit une modification des comportements pouvant donner lieu à d'autres modifications du paysage (stigmergie).
3. Intégrer les dynamiques des objets et des espaces, lesquels ont des effets en retour sur l'état des agents et des lieux.
4. Intégrer les aspects sociaux et culturels en utilisant une extension du modèle AGR, qui fusionne les aspects culturels des groupes (normes, croyances collectives), au niveau de la prise de décision de chaque agent. Cela permet ainsi d'avoir une vision de la culture et des normes comme une motivation qui vient influencer le comportement de l'agent, et non comme des règles à prendre en compte lors d'un raisonnement.
5. Définir et utiliser des sous-frameworks d'interaction pour modéliser le commerce, les combats, les rencontres sociales, etc.

6. Permettre d'implémenter facilement des *design patterns* cognitivo-eco-sociaux reliant les états mentaux des agents, leurs comportement, les demandes sociales, les organisations et la dynamique de l'environnement.

## 2 Méta-Modèle

Inspiré par le modèle des *quadrants*, la plateforme Metaciv est fondé sur le méta-modèle suivant.

## 3 Agent

Les agents sont vus sous l'angle de *l'intériorité* comme dotés d'un esprit (ensemble de cognitons) qui leur permet d'établir des plans (ensemble d'actions); vus sous l'angle *extériorité* ils sont dotés d'un ensemble de caractéristiques (corps) et peuvent manipuler des objets. Ils évoluent dans un environnement collectif agrégat de *patches*. Côté spatialisation, ont été rajoutés les concepts de zones agrégats de patches liées par des relations spatiales (cf. relations d'Egenhofer). Sur les patches comme sur les zones l'agent peut déposer des aménagements (*facility*). Les cognitions vont influencer les plans, de manière pondérée (selon l'importance que le modélisateur leur accorde), de manière positive (renforcement) ou négative (affaiblissement). L'environnement ainsi que les actions effectuées peuvent affecter les caractéristiques d'un agent et de ce fait influencer sur ses cognitons. De même les actions peuvent directement ou indirectement via les objets influencer sur les cognitons.

### 3.1 Cognition

#### 3.1.1 Notion de cognitons

Les agents étant amenés à devoir opérer des choix dans un large panel d'actions ceux ci doivent donc disposer de capacité de réflexion et de choix suffisamment développée.

Dans cette optique, l'idée retenue consiste à pondérer les différentes actions possibles pour les agents, et ce afin de les faire choisir en fonction de ces poids. Les poids des actions sont déterminés par différents facteurs : ce que voient, pensent, croient les agents... L'ensemble de ces facteurs est regroupé sous le terme de *cogniton* dans ce projet, en reprenant le néologisme proposé par J. Ferber. Le cogniton est donc une « unité de pensée » qui influe sur les choix de l'agent.



### 3.1.2 Catégorisation des cognitons

Pour mieux structurer l'organisation des cognitons, ceux-ci sont catégorisés en cinq types : *Skills*, *Traits*, *Beliefs*, *Percepts*, *Mêmes*. Cette distinction permet de séparer des comportements différents entre ces cognitons, et de les traiter au mieux par la suite.

*Remarque : la catégorisation, pour l'instant, est juste réalisée via une caractéristique du cogniton, à partir de laquelle il est possible de moduler les actions.*

- Les *Skills* (signifiant compétences en anglais) représentent les compétences, savoir-faire et connaissances techniques ou scientifiques des agents. Comme exemples de ce type de cognitons, on peut proposer : Agriculture, Navigation, Fabrication d'outils... La plupart des skills ont la particularité d'être transmissibles d'un agent à l'autre, d'être permanents (ils ne disparaissent pas une fois acquis) et d'être des prérequis indispensables à certaines actions.

Par exemple, même si d'autres cognitons l'influencent, un agent ne peut pas fabriquer un outil s'il ne dispose pas de Fabrication d'outils. Enfin, les skills sont héréditaires. Ici, on parle de l'hérédité au sens de la transmission entre les générations, ce qui représente de manière simple l'apprentissage et la transmission des connaissances.

- Les *Traits* représentent les spécificités individuelles de l'agent, ses traits de caractères, ses façons d'être. Des exemples possibles de ce type de cognitons sont : Ouvert, Renfermé, Paresseux...

Cette catégorie n'est pas la plus importante du point de vue de la simulation et du réalisme, mais est un outil simple pour faire varier le comportement des agents si l'on veut envisager des scénarii spécifiques (des civilisations très agressives, des agents peu travailleurs, etc...). Les traits ne sont pas transmissibles d'agents à agents, sauf de manière héréditaire occasionnellement, ce qui représente l'imitation et l'éducation.

- Les *Beliefs* (Croyances en anglais) représentent ce que l'agent sait ou croit savoir de son environnement et de lui même. Cette catégorie est vaste, et peut regrouper des cognitons aussi variés que : Je porte une pioche, Nous sommes en guerre avec la civilisation 3, Je possède un champ. Ces cognitons ne sont généralement pas transmissibles entre agents, et ne sont pas héréditaires.
- Les *Percepts* représentent ce que l'agent voit, entend ou ressent (physiquement) au moment considéré. Des exemples de percepts sont : J'ai faim, Un ennemi est proche, Je suis près de l'eau. Ce type de cogniton est constamment retiré ou ajouté, contrairement aux autres types



qui sont plus stables. Les percepts ne sont pas transmissibles, ils sont propres à l'agent considéré.

- Les *Mêmes*, d'après le terme inventé par R.Dawkins [2].

Son sens varie suivant les sources. Dans notre cas, nous définissons un même comme étant une croyance ou un comportement culturel transmissible. Ainsi, les mêmes pourraient être : Je crois que l'argent est une fin en soi, je crois en l'existence d'un dieu, je crois que la démocratie est une bonne chose. Typiquement, les opinions religieuses et politiques sont des mêmes. Par définition, les mêmes sont transmissibles, et ils sont potentiellement héréditaires.

Les cognitons ont un poids qui par défaut est 1 (et qui peut être modifié par une action ou par l'effet d'un objet ou par les culturons)

## 3.2 Objets

Les objets peuvent être créés, selon les besoins, par les agents (à partir de concepts d'objets précédemment définis : parmi ces concepts on distingue les objets simples et composites). Les objets feront ensuite partie de l'inventaire d'un agent. Les objets peuvent avoir un effet sur des caractéristiques d'un agent ou sur des actions.

## 3.3 Plan et actions

Les plans sont des ensembles d'actions et sont liés à des cognitons qui les influencent.

## 3.4 Groupe et rôle

Les agents peuvent aussi appartenir à des groupes au sein desquels ils possèdent différents rôles. Le rôle d'un agent dans un groupe lui permet d'accéder à des cognitons dits *culturons* c'est-à-dire , commun à tous les agents ayant le même rôle que lui dans ce groupe.

## Deuxième partie

# Le logiciel

## 4 Préambule

MetaCiv, dans sa forme actuelle est un *framework* permettant de mettre en place des modèles de simulation de sociétés/civilisations. Il repose sur MadKit (qui offre un mécanisme de gestion multi-agents) et TurtleKit (qui fournit en particulier l'interface et la gestion « physique » des agents).

Les principaux outils fournis par MetaCiv sont :

- un Système « cognitif » pour les agents
- un outil de Gestion de l'environnement à base de ressources (Cf. phéromones de Turtle Kit)
- des Outils de visualisation
- des Outils d'édition graphique

## 5 Installation

Il faut disposer du langage de programmation Java (version au moins 1.7) et récupérer le dossier Metaciv qui contient le dossier *Launcher* ainsi que divers modèles exemples dans le Dropbox.

Ouvrir de dossier *Launcher*, double clic sur le module *Metaciv+numversion.jar*

Une fenêtre s'ouvre, il faut alors rechercher et ouvrir le fichier de paramètres initiaux **parametres.metaciv** dans le modèle choisi. (Chaque modèle a un fichier de paramétrage)

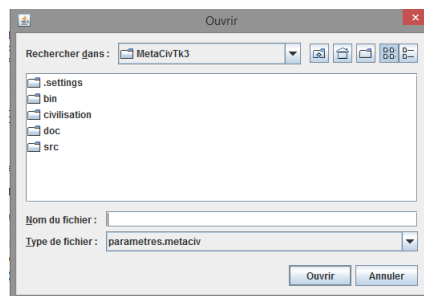


FIGURE 2 – Le choix du fichier de paramétrage initial

Deux fenêtres, constituant l'environnement de fonctionnement de Metaciv s'ouvrent alors :

- l'une *worldviewer* propose un fond cartographique par défaut (environnement d'évolution)

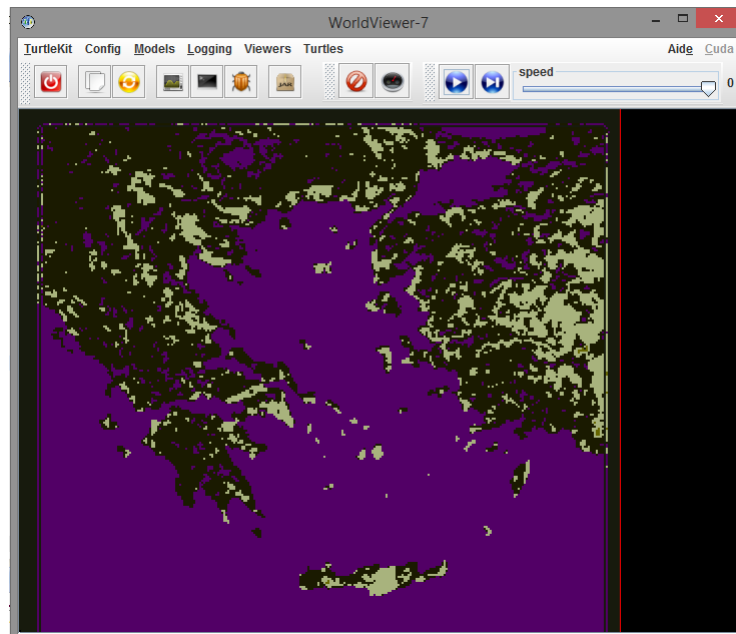


FIGURE 3 – La vision de l'environnement de simulation

- l'autre *viewertab*

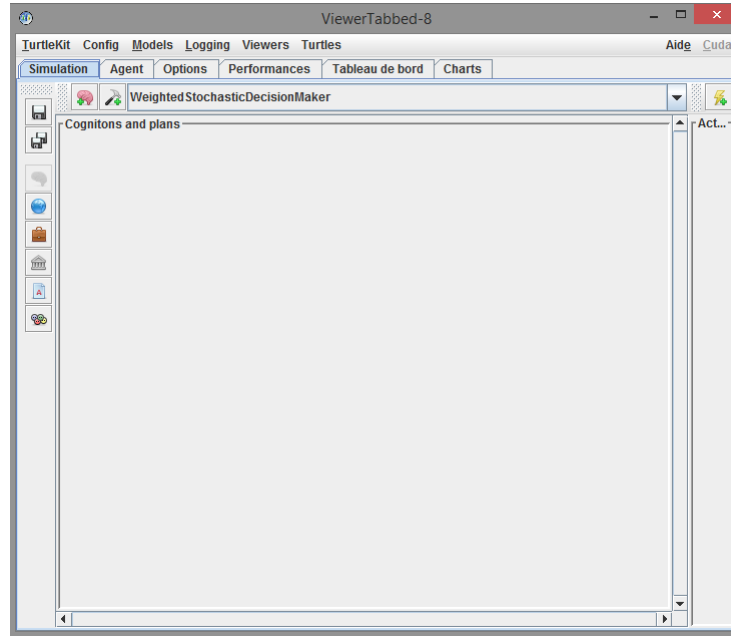


FIGURE 4 – La vision de la fenêtre entrée / sortie pour le modélisateur

Les onglets de la ligne supérieure sont réservés aux experts *turtlekit*.

La ligne suivante propose des onglets permettant d'accéder à différents détails :



FIGURE 5 – Les onglets accessibles

- L'onglet *Simulation* dispose d'une palette d'icônes permettant la création du modèle et la modification de la simulation (cf. Figure 6).

Le détail de ces icônes :

1. Les deux premières icônes : **Save** disquette permet de sauvegarder la simulation dans sa globalité, **Make copy of current save** et double disquette permet de dupliquer la dernière sauvegarde.
2. L'icône **Edit cognitive scheme** Permet de créer et modifier cogniton et plans.
3. L'onglet **Edit Environment** Utilisé pour créer et modifier les différents terrains des patches.



FIGURE 6 – Les icônes de l’onglet Simulation

4. L’onglet **Edit Item** Utilisé pour créer et modifier les différents objets et leurs effets.
  5. L’icône **Edit civilizations** Permet de créer et modifier plusieurs populations mais constituée d’agents initialement identiques. Chaque population a un nom et un nombre d’agents.
  6. L’onglet **Edit Attributes** Utilisé pour créer et modifier pour chacune des caractéristiques (attribut) son nom et sa valeur initiale (exemple vie 100)
  7. L’onglet **Manage Groupe** Cet onglet permet de structurer les populations initiales.
  8. L’onglet **Edit Aménagement** Cet onglet permet de créer et modifier les aménagements.
  9. L’onglet **Edit Configuration** Cet onglet permet de modifier les paramètres : nombre de messages de passages nécessaires sur un patch pour créer une route, nombre de messages de passages à enlever tous les 75 ticks (les messages sont assimilés à des phéromones et peuvent s’évaporer), nombre de patches visibles par l’agent depuis le patch où il est situé (*vision*).
- L’onglet *Agent* : affiche des informations sur les agents, permet de sélectionner l’agent à observer.
  - L’onglet *Options* : permet de positionner différentes options (en particulier d’affichage).
  - L’onglet *Performances* : présente différentes informations sur la consommation mémoire.
  - L’onglet *Tableau de bord* : offre quelques outils de visualisation, par exemple le ratio des cognitons (sous forme de camembert)

- L’onglet *Charts* : présente des graphiques de synthèse et le poids des plans

**Le modélisateur doit donc créer toutes entités nécessaires à son modèle (ce qui nécessite une phase de réflexion et la connaissance d’un méta-langage afin d’invoquer des codes préexistants).**

Lorsque l’univers est créé, revenir sur la fenêtre *worldviewer* et cliquer sur l’icône Play (on peut régler la vitesse d’exécution) et éventuellement passer en mode pas à pas. Il est possible d’effectuer des Zooms sur l’environnement avec la souris. À noter la présence du bouton fin qui clôt la simulation (impératif) et quitter le *CivLauncher*.

## Troisième partie

# Documentation des actions

## 6 Les plans

Les plans sont des successions d'actions. Nous distinguons trois formes de plans :

- l'*autoplan* qui s'exécute en toile de fond à chaque tick ; il sert à décrire des plans relatifs à la physiologie de l'agent. Il a un label bleu dans l'interface.
- le *birthplan* s'exécute à la création de l'agent (soit à l'initialisation des peuplements, soit après la création par un agent ultérieurement). Il a un label rouge dans l'interface.

*Remarque : autoplan et birthplan ne sont pas liés à des cognitons*

- les *plans* proprement dits, influencés et conditionnés par les cognitons. Un plan doit obligatoirement être conditionné par un ou plusieurs cognitons (qui doivent être dans l'esprit de l'agent pour pouvoir s'exécuter), il peut aussi être influencés par d'autres cognitons (l'influence ne s'exercera que si le cogniton origine de celle-ci est dans l'esprit de l'agent). Ils ont un label noir dans l'interface.

Les actions sont exécutées à chaque pas de temps (*tick*) et sur le *patch* où se situe l'agent au moment de leur exécution.

*Remarque : il existe cependant une action logique  $L\_Instant$  qui permet d'exécuter plusieurs actions en un seul tick.*

Les actions se spécialisent en actions proprement dites (préfixe A) ainsi qu'en actions logiques (préfixe L) qui regroupent des actions et d'autres actions logiques (cf. Figure 7).

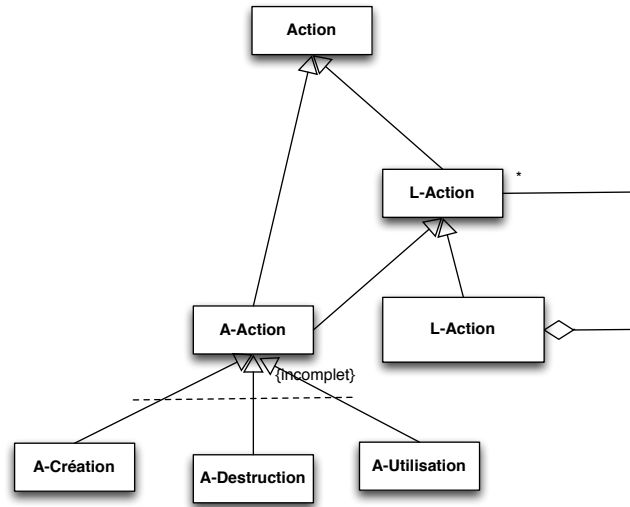


FIGURE 7 – Hiérarchie des actions

## 7 Les actions A

### 7.1 Les actions relatives aux objets

#### 7.1.1 A\_CreateObject

Cette action crée un objet conformément au processus de fabrication (recette ou *recipe*) défini au préalable.

La signature de l'action est `A_CreateObject(Created Object)` :

- `Created Object`, est l'objet à fabriquer.

Exemple (pour un agent agriculteur) `A_CreateObject(Houe)` lance la recette de fabrication de la Houe qui nécessite un bâton pour le manche et un soc (pièce métallique). Si l'agent ne possède pas les deux objets requis pour fabriquer la houe l'action ne fera rien, sinon, elle fabriquera 1 unité de cet objet et l'ajoutera à l'inventaire de l'agent exécutant l'action. Les objets élémentaires ayant servi à la fabrication seront supprimés de l'inventaire.

#### 7.1.2 A\_AddObject

Action permettant d'ajouter N objets à l'inventaire de l'agent.

La signature de l'action est `A_AddObject(Modified Object, N)` :

- `Modified Object`, précise le type d'objet utilisé pour créer directement l'objet à ajouter à l'inventaire de l'agent (sans utiliser la recette)
- `N`, le nombre d'objets à ajouter.



### 7.1.3 A\_DropObject

Action permettant de supprimer N objets de l'inventaire de l'agent.

La signature de l'action est A\_DropItem(Modified Object, N) :

- **Modified Object**, type de l'objet à retirer de l'inventaire de l'agent,
- **N**, le nombre d'objets à retirer.

### 7.1.4 A\_Transform

Cette action transforme une ressource prélevée sur un patch en objet.

La signature de l'action est A\_Transform (Resource to Collect, Modified Object) :

- **Resource to Collect**, le type de la ressource que l'agent prélève,
- **Changed Object**, l'objet correspondant (pour le modélisateur) à ajouter à l'inventaire de l'agent.

Exemple : un agent exécute A\_Transform (Blé, Meule).

### 7.1.5 A\_UseObject

Cette action correspond à l'usage d'un objet.

La signature de l'action est A\_UseObject(Modified Object, N) :

- **Modified Object**, le type d'objet à utiliser,
- **N**, le nombre.

### 7.1.6 A\_AddObjectXCogniton

La signature de l'action est A\_AddObjectXCogniton(Modified Object, variation, base, Cogniton) :

- **Modified object**, le type d'objet à ajouter,
- **variation**, la variation du nombre d'objets à ajouter,
- **base**, le nombre d'objets à ajouter par défaut,
- **Cogniton**, le cogniton référence.

Cette action, suite à l'existence d'un Cogniton dans l'esprit de l'agent, ajoute des objets à l'inventaire de l'agent en fonction de paramètres contextuel :

- si le cogniton n'existe pas dans l'esprit de l'agent, l'action ajoute *base* objets à l'inventaire,
- si le cogniton existe dans l'esprit de l'agent, l'action ajoute *base* + (*variation* \* poids de *Cogniton*) objets à l'inventaire.

Exemple : un agent qui possède le Cogniton Artisan avec un poids de 2, exécute l'action A\_AddObjectXCogniton(Houe, 0.5,1, Artisan) ce qui lui permet d'ajouter  $1 + 0.5 * 2$  c'est-à-dire 2 Houes à son inventaire ; si l'agent

ne possède pas le Cogniton Artisan l'exécution ajoute 1 seule Houe à son inventaire.

## **7.2 Les actions relatives aux aménagements**

### **7.2.1 A\_CreateFacility**

Cette action permet à l'agent de construire l'aménagement sur le patch où il se trouve au moment de l'exécution ; la signature de l'action est A\_CreateFacility(Facility) :

- Facility, le type d'aménagement à construire

### **7.2.2 A\_EraseFacility**

Cette action permet à l'agent de détruire l'aménagement du patch où il se trouve au moment de l'exécution, la signature de l'action est A\_EraseFacility(Facility) :

- Facility, le type d'aménagement à détruire

### **7.2.3 A\_GoToFacility**

Cette action permet à l'agent d'atteindre le patch le plus proche disposant de l'aménagement (si celui ci existe) , la signature de l'action est A\_GoToAmenagement(Facility) :

- Facility, le type d'aménagement à cibler

.

### **7.2.4 A\_UseFacility**

Cette action permet à l'agent l'usage de l'aménagement (à condition qu'il soit sur le patch sur lequel se trouve l'agent), la signature de l'action est A\_UseAmenagement(Facility) :

- Facility, le type d'aménagement à utiliser

## **7.3 Les actions relatives aux cognitons**

### **7.3.1 A\_AddCogniton**

Cette action ajoute un cogniton à l'esprit de l'agent.  
La signature de l'action est A\_AddCogniton(Cogniton) :

- Cogniton, le cogniton à ajouter

### 7.3.2 A\_ChangeCognitonWeight

Cette action modifie le poids d'un cogniton.

La signature de cette action est A\_ChangeCognitonWeight(Cogniton,N) :

- Cogniton, le cogniton à modifier
- N, la valeur à ajouter au poids du cogniton

### 7.3.3 A\_DoubleCognitons

Cette action ajoute deux cognitons à l'esprit de l'agent.

La signature de cette action est A\_DoubleCognitons(Cogniton1,Cogniton2) :

- Cogniton1, le premier cogniton à ajouter,
- Cogniton2, le deuxième cogniton à ajouter.

### 7.3.4 A\_EmitCogniton

Cette action modifie le poids du cogniton de tous les agents se trouvant sur le même patch que l'exécutant.

La signature de l'action est A\_EmitCogniton(Cogniton, n)

- Cogniton, le cogniton à modifier,
- n, la valeur à ajouter au poids du cogniton.

## 7.4 Les actions relatives aux déplacements

### 7.4.1 A\_MoveTowards

L'action permet à l'agent de se déplacer vers une cible spécifiée.

*Remarque : la cible aura été définie par l'action A\_GetAnotherSettlementPatch.*

### 7.4.2 A\_MoveRandomly

L'action permet à l'agent d'effectuer un pas dans une direction aléatoire.

*Remarque : la longueur du pas dépend de la facilité de traversée du patch (cf. passability)*

### 7.4.3 A\_DoNothing

Cette action permet à l'agent de n'exécuter aucune action.

#### 7.4.4 A\_ CreateSettlement

Cette action permet à l'agent de se déplacer aléatoirement jusqu'à ce qu'il trouve un ensemble de patchs (les 8 patchs voisins immédiats) dépeuplé de manière à créer un nouveau *Settlement*.

#### 7.4.5 A\_ GetAnotherSettlementPatch

Cette action définit la cible utilisée dans l'action A\_ MoveTowards comme étant un des *Settlement* pris au hasard dans l'ensemble des *Settlement* existants.

#### 7.4.6 A\_ GoBackHome

Cette action permet à l'agent de se déplacer d'un pas vers son lieu de création s'il s'agit d'un des agents des peuplements initiaux, sinon vers le lieu où se trouvait l'agent qui l'a engendré.

#### 7.4.7 A\_ Move

Cette action déplace l'agent d'un pas dans une direction donnée.

La signature de cette action est A\_Move(String) :

- **String**, "NORTH", "SOUTH", "WEST", "EAST" correspond à la direction que prendra l'agent

#### 7.4.8 A\_ SearchForResources

Cette action permet à l'agent de rechercher autour de lui (dans son champ de vision) la ressource passée en paramètre, pour cela il se dirigera vers le patch le plus proche et possédant cette ressource en quantité maximum. S'il ne trouve pas de patch contenant cette ressource, il se déplacera aléatoirement de un pas.

La signature de cette action est A\_SearchForResources(Resource To Collect) :

- **Resource To Collect**, correspond à la ressource recherchée.

#### 7.4.9 A\_ SmellAndMove

Cette action permet à l'agent de chercher dans son voisinage immédiat le patch contenant la ressource passée en paramètre en plus grande quantité et de faire un pas dans sa direction.

La signature de cette action est A\_SmellAndMove(Resource To Collect) :

- **Resource To Collect**, correspond à la ressource recherchée.

## 7.5 Les actions relatives aux agents

### 7.5.1 A\_GiveBirth

L'action permet à l'agent qui l'exécute de créer un nouvel agent sur le patch où il se situe : cet agent sera identique à ceux du peuplement initial.

### 7.5.2 A\_Die

L'action permet à l'agent qui l'exécute de se supprimer.

### 7.5.3 A\_DieIfAttributeUnderZero

L'action supprime l'agent qui l'exécute si la valeur de l'attribut (ou caractéristique) passé en paramètre descend en dessous de zéro. La signature de cette action est `A_DieIfAttributeUnderZero(attributeToCompare)` :

- `attributeToCompare`, correspond à l'attribut à vérifier

### 7.5.4 A\_ChangeAttribute

L'action modifie une caractéristique de l'agent.

La signature de l'action est `A_ChangeAttribute(Modified attribute, n)` ;

- `Modified attribute`, correspond à la caractéristique à modifier
- `n`, donne la valeur à ajouter à la caractéristique

### 7.5.5 A\_Trade

L'action `A_Trade` comporte deux actions internes, elle correspond à un échange conditionnel, car elle suppose que si après un temps fixé (*turns*) l'agent n'a pas réussi à trouver un partenaire il exécute la deuxième action interne sinon la première.

La signature de l'action est `A_Trade(turns, objectToGive, nObjectToGive, objectToTake, nObjectToTake, myTag, compatibleTag)` ;

- `turns`, correspond au nombre de ticks durant lequel l'agent attend sur son patch un autre agent pour réaliser un échange,
- `objectToGive` donne l'objet que l'agent échange,
- `nobjectToGive` correspond au nombre `n` d'objets à échanger,
- `objectToTake` précise l'objet reçu en échange,
- `nobjectToTake` correspond au nombre `n` d'objets reçus,
- `mytag`, message que l'agent passe à l'attention des autres,
- `compatibletag` message recherché chez le partenaire échangeur potentiel.

### 7.5.6 A\_TravelTrade

L'action A\_TravelTrade comporte deux actions internes, elle correspond à un échange conditionnel, car elle suppose que si après un temps fixé (*turns*) l'agent n'a pas réussi à trouver un partenaire il exécute la deuxième action interne sinon la première.

La signature de l'action est A\_TravelTrade(turns, objectToGive, nObjectToGive, objectToTake, nObjectToTake, myTag, compatibleTag) ;

- **turns**, correspond au nombre de ticks durant lequel l'agent cherche des partenaires commerciaux dans son voisinage et se déplace vers eux,
- **objectToGive** donne l'objet que l'agent échange,
- **nobjectToGive** correspond au nombre n d'objets à échanger,
- **objectToTake** précise l'objet reçu en échange,
- **nobjectToTake** correspond au nombre n d'objets reçus,
- **mytag**, message que l'agent passe à l'attention des autres,
- **compatibletag** message recherché chez le partenaire échangeur potentiel.

### 7.5.7 A\_CreateGroup

L'action permet à l'agent de créer un groupe.

La signature de l'action est A\_CreateGroup(GroupToCreate) :

- **GroupToCreate**, donne le type du groupe à créer.

### 7.5.8 A\_HireForRole

L'action permet à l'agent exécutant d'ajouter un agent présent sur le même patch que lui à son groupe .

La signature de l'action est A\_HireForRole(GroupToCreate) :

- **GroupToCreate**, donne le type du groupe dans lequel l'agent sera admis

*Remarque : un agent peut appartenir à plusieurs groupes*

## 8 Les actions L

### 8.1 Tests

La structure générale des actions logiques de type Test est :

```

begin
  | if condition then
    | action interne 1
  end
  | else
    | action interne 2
  end
end

```

*Remarque : ces actions logiques sont toutes composées de deux actions internes (qui peuvent elles-mêmes être composites).*

Les diverses actions suivantes vont préciser le type de condition.

#### 8.1.1 L\_CompareAttribute

La condition porte sur la valeur d'un attribut de l'agent.

La signature de cette action est `L_CompareAttribute(attributTocompare, comparator, n)` :

- `attributTocompare`, est l'attribut concerné,
- `comparator` l'opérateur de comparaison `<`, `>`, `<=`, `>=`, `==`,
- `n` la valeur de comparaison.

#### 8.1.2 L\_CompareObject

La condition porte sur le nombre d'objets d'un type donné possédés.

La signature de cette action est `L_CompareObject(objectTocompare, comparator, n)` :

- `objectTocompare`, est le type d'objet concerné,
- `comparator` l'opérateur de comparaison `<`, `>`, `<=`, `>=`, `==`,
- `n` le nombre d'objets.

#### 8.1.3 L\_CompareResource

La condition porte sur la valeur de la ressource présente sur le patch où est situé l'agent.

La signature de cette action est `L_CompareResource(resourceTocompare, comparator, n)` :

- `resourceTocompare`, est le type de la ressource concernée,
- `comparator` l'opérateur de comparaison `<`, `>`, `<=`, `>=`, `==`,
- `n` la valeur de la ressource.

#### 8.1.4 L\_IsFacilityHere

La condition teste la présence de l'aménagement (*facility*) concerné sur le patch où est situé l'agent.

La signature de cette action est L\_IsFacilityHere(*facility*) :

- *facility*, le type d'aménagement testé.

#### 8.1.5 L\_IsAnyFacilityHere

La condition teste la présence d'un aménagement de n'importe quel type sur le patch où est situé l'agent.

#### 8.1.6 L\_OwnFacility

La condition teste la possession d'un aménagement donné.  
La signature de cette action est L\_OwnFacility(*facility*) :

- *facility*, le type d'aménagement testé.

#### 8.1.7 L\_OwnCogniton

La condition teste la possession d'un cognition.  
La signature de cette action est L\_OwnCogniton(*cogniton*) :

- *cogniton*, le type de cognition testé.

#### 8.1.8 L\_OwnObject

La condition teste la possession d'un objet.  
La signature de cette action est L\_OwnObject(*object*) :

- *object*, le type d'objet testé.

### 8.2 Algorithmiques

Ces actions logiques composites sont associées à des structures standard algorithmique.

#### 8.2.1 L\_Instant

Cette action composite effectue la globalité de ses actions internes en un seul tick.



### 8.2.2 L\_Random

Cette action composite effectue une seule de ses actions internes choisie aléatoirement en un seul tick.

### 8.2.3 L\_RandomPercent

Cette action composite n'a que deux actions internes et choisit selon le pourcentage l'action interne à exécuter.

La signature de cette action est `L_RandomPercent(n)` :

- `n`, le pourcentage de chance qu'à l'action interne 1 de s'exécuter.

### 8.2.4 L\_Repeat

Cette action composite ne comporte qu'une action interne qu'elle va exécuter `n` fois en `n` tick.

La signature de cette action est `L_Repeat(n)` :

- `n`, le nombre d'itérations.

## Quatrième partie

# Outils de création d'un modèle

## 9 Phase d'initialisation

### 9.1 Peuplements

Dans la fenêtre *Wiewertabs*, cliquer sur l'icône *Edit civilisation*, sur la droite la liste des peuplements existants (par défaut rattaché à l'environnement par défaut). Pour créer un nouveau peuplement, cliquer sur l'icône *create new civilization*, donner son nom (par exemple Volque), sa population (par exemple 100) et valider (touche entrée).

### 9.2 Environnement physique

Dans la fenêtre *Wiewertabs*, cliquer sur l'icône *Edit environnement*, la liste des terrains attribuables aux *patches* et définis dans le dossier de la simulation apparait dans la partie droite de la fenêtre (cf. Figure 8).

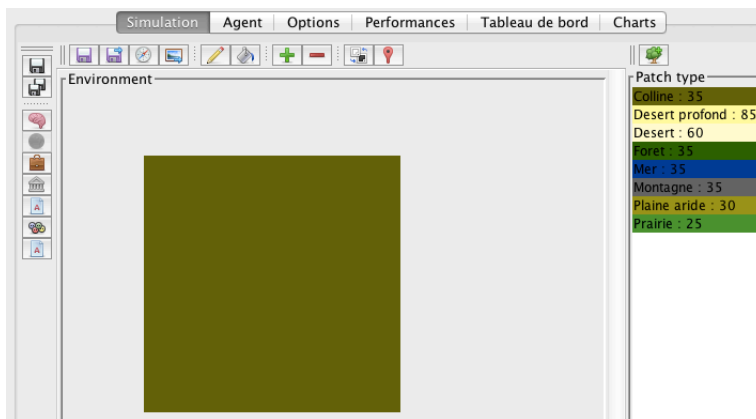


FIGURE 8 – Environnement

Pour la simulation deux terrains sont utilisés *Forêt* et *Prairie* avec une valeur liée à leur facilité de traversée (*passability*).

Les modifications sur un terrain se font par l'intermédiaire d'un clic droit sur le terrain voulu. On peut alors éditer, changer la couleur ou supprimer le terrain. Si on édite notre exemple (cf. Figure 9 ), on observe le terrain *Prairie* avec la valeur 25 pour *passability*, puis une liste d'attributs correspondant

chacun à un type de ressource disponible sur le terrain. Pour chaque ressource on peut attribuer une valeur initiale et une valeur de croissance (à chaque *tick*).

FIGURE 9 – Edition d'un terrain

Dans la partie centrale de la fenêtre, l'environnement global peut être construit, dimensions (icône boussole *set environment bounds*), chargement d'un environnement existant (icône *Load Environment*), définir à partir d'une image (*generate an environment from an existing picture*) ou dessiner (icône crayon et pot de peinture).

On peut sauvegarder l'environnement (icône disquette), choisir l'environnement de la simulation (qui peut donc être un autre) icône *choose environment to use for simulation*, et ajouter des ressources à l'environnement (icône *manage pheronom*).

Dernière chose, situer le ou les peuplements sur l'environnement : pour cela clic droit sur la position voulue (possibilité de zoom) et donner le nom du peuplement dans la fenêtre qui s'ouvre (et penser à sauvegarder).

### 9.3 Les objets

Dans la fenêtre *Wiewertabs*, cliquer sur l'icône *Edit Item*.

Sur la partie droite une liste d'objets apparaît et en cliquant sur l'icône *Add Item* on accède à la fenêtre suivante (cf. Figure 11).

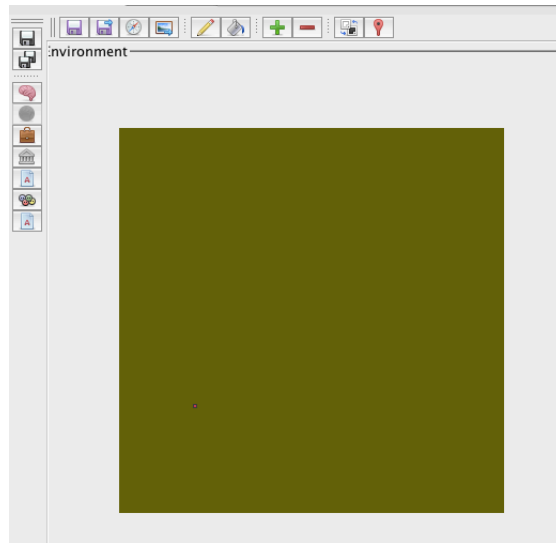


FIGURE 10 – Localisation du peuplement, le point marque la localisation

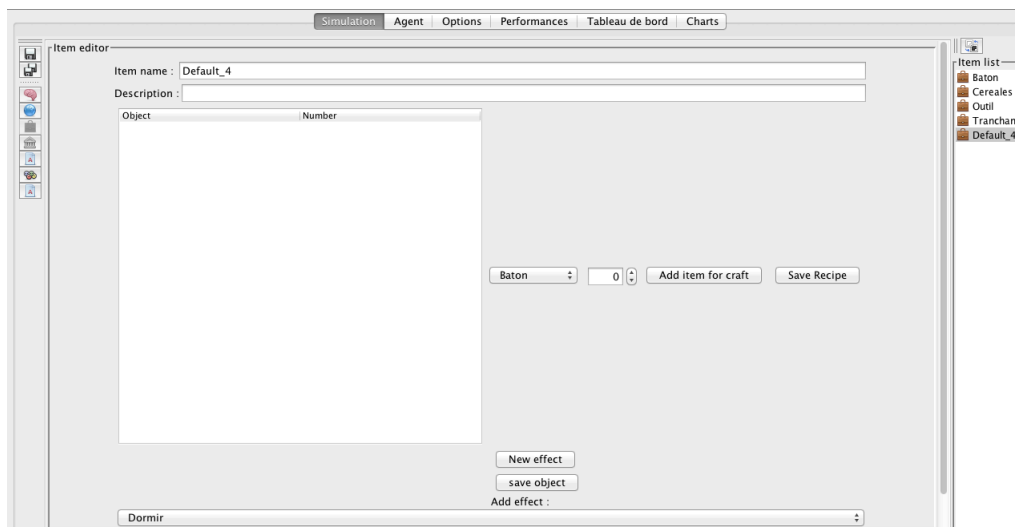


FIGURE 11 – Ajout d'objet

Dans cette fenêtre donner un nom et une description à l'objet avant de sauvegarder.

Si l'on souhaite créer des descriptions d'objet complexe, on peut saisir le nom et le nombre d'éléments constitutifs (ce qui constitue la recette ou *recipe*).

On peut de plus ajouter des effets existants ou en créer de nouveaux (cf. Figure 12).

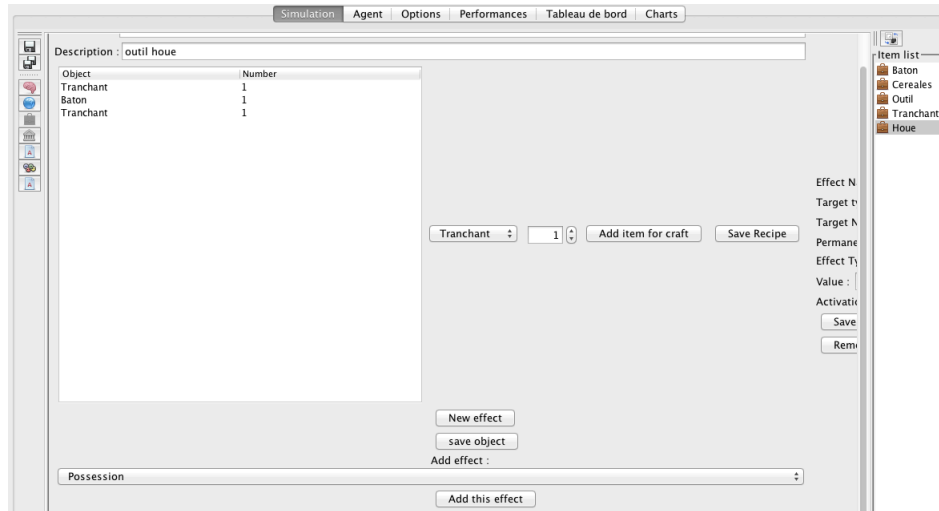


FIGURE 12 – Objet complexe

En ce qui concerne les effets (cf. Figure 13), tout effet a un nom, une cible qui peut être un attribut (caractéristique de l’agent) ou un cogniton. Le type de cible étant choisi, on détermine la cible exacte, la durée de l’effet (permanent ou non c’est-à-dire lié à la durée de vie de l’objet), ce qu’il fait (*set*, *add*, *remove*) sur la cible en terme de valuation et en fonction de toute action relative à la création, à l’utilisation et à la destruction de l’objet auquel l’effet est rattaché.

Exemple, l’objet *Houe* a pour effet *Possession* dont la cible est le cognition *Produireautrechose*, cet effet est transitoire (appliqué tant que l’agent possède la houe), son action sur le cognition est *add* (ajouter) la valeur 0.2 au poids de celui-ci, son activation sera effective chaque fois qu’un agent ajoutera une *Houe* à son inventaire.

Effect Name :

Target type : ☐ Attributes ☒ Cognitons ☐ Actions

Target Name :

Permanent ? : ☐ Yes ☒ No

Effect Type : ☐ Set ☒ Add ☐ Remove

Value :

Activation : ☒ Possession ☐ Use ☐ Remove

FIGURE 13 – Effet pour un objet

## 9.4 Les aménagements ou infrastructures

Dans la fenêtre *Wiewertabs*, cliquer sur l'icône *Edit aménagement*, sur la droite on dispose de la liste des aménagements existants. Pour créer un nouvel aménagement, cliquer sur l'icône *create new aménagement*, donner son nom (par exemple Champ) et sauvegarder.

Effect Name :

Target type : ☐ Attributes ☐ Cognitons ☐ Objet ☒ Ressources/tick

Target Name :

Permanent ? : ☒ Yes ☐ No

Effect Type : ☐ Set ☒ Add ☐ Remove

Value :

Activation : ☐ Possession ☒ Use ☐ Remove

FIGURE 14 – Effet pour un aménagement

Tout comme pour les objets, il est possible de saisir une recette pour la conception d'aménagements complexes, le processus étant identique à celui présenté dans la section sur les objets. De même, les aménagements peuvent avoir des effets ceux ci étant un peu différents de ceux présentés pour les objets (cf. Figure 14)

En effet, les aménagements ont la possibilité d'avoir pour cible (en sus des attributs ou caractéristiques d'agent et des cognitions) des objets ou des ressources, et dans ce cas, l'effet ajoutera, modifiera ou bien supprimera l'objet

sélectionné de l'inventaire, ou bien, modifiera, ajoutera, modifiera ou supprimera la ressource ciblée présente sur le patch où se situe l'aménagement.

## 10 Phase de modélisation

### Le canevas : cogniton, plan, action

La création d'un nouveau cogniton se fait en cliquant sur l'icône .

Un nouveau cogniton est alors créé sous la forme d'une ellipse portant le nom "Nouveau cogniton".

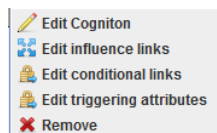


FIGURE 15 – Interface d'édition de cogniton

Un clic droit sur ce dernier permet d'ouvrir l'interface d'édition (cf. Figure 15).

Le premier lien permet de donner le nom du cogniton, son type ( *Belief* s'il s'agit d'une croyance de l'agent, même si il s'agit d'une croyance populaire, *Percept* s'il s'agit d'une perception de l'agent, *Skill* s'il s'agit d'une compétence ou d'un savoir-faire, *Trait* s'il s'agit d'une caractéristique psychologique de l'agent et *Culturon* s'il s'agit d'un cogniton partagé par un groupe), et ses pourcentages de chances d'apparition au lancement de la simulation.

Vient ensuite l'édition des *liens d'influence*, à savoir, spécifier quels plans vont être influencés par ce cogniton et dans quelle mesure vont ils l'être. Pour cela, l'on doit sélectionner le plan à influencer ainsi décider le poids de cette influence sur le plan en question. Une fois toutes les influences définies cliquer sur le bouton *OK* pour les enregistrer dans la simulation.

Dans le même menu se trouve l'édition des *liens conditionnels*. Ces liens spécifient les actions pour lesquelles ce cogniton doit exister obligatoirement dans l'esprit de l'agent pour qu'elle s'exécute. Pour les spécifier il suffit de sélectionner un à un les plans à lier avec l'existence de ce cogniton dans la liste déroulante, puis, cliquer sur le bouton *OK* pour terminer.

Enfin cette interface permet la création de *triggers*. Ces *triggers* permettent de réguler l'apparition d'un cogniton dans l'esprit d'un agent en fonction de la valeur des caractéristiques de ce dernier. Comme un *trigger* est un déclencheur du type *si condition alors*, la création d'un *trigger* se fait

en définissant la condition de déclenchement, c'est-à-dire, en sélectionnant la caractéristique de l'agent à comparer, la fonction de comparaison à utiliser, et enfin, la valeur de comparaison. Le cogniton ne se déclenchera, ainsi, dans l'esprit de l'agent, que si cette condition est respectée. Il est possible de définir plusieurs conditions au sein d'un même *trigger* . Une fois ces conditions définies, cliquer sur le bouton *OK* pour finir la création du trigger.



## Cinquième partie

# Exemple de simulation

*La simulation AgriculteurMaisonBois2*

## 11 Le modèle

### 11.1 Peuplement et environnement

Dans cette simulation, un seul peuplement Civ1 avec une centaine d'agents.

Ceux-ci, ont initialement tous les mêmes caractéristiques : sommeil (50) et vie (20).

La simulation aura lieu dans un environnement constitués de *patches* de deux types de terrains : forêt et prairie de *passability* 25 et 35.

Après chargement de l'environnement Test.metaciv, on dispose de l'environnement et de la localisation de départ de Civ1 (cf. Figure 16)

L'inventaire des agents pourrait potentiellement comporter les objets Baton, Céréale, Outil.

Céréale a l'effet possession qui ajoutera 0.2 au poids du cogniton *Produireautrechose* à chaque fois qu'un agent ajoutera un objet céréale à son inventaire.

En ce qui concerne les aménagements (*facility*), on trouve Champ et Maison. Champ a un effet permanent *Renforcer agriculture* qui ajoute 0.05 au poids du cogniton *Agriculteur* à chaque fois qu'il utilise le champ.

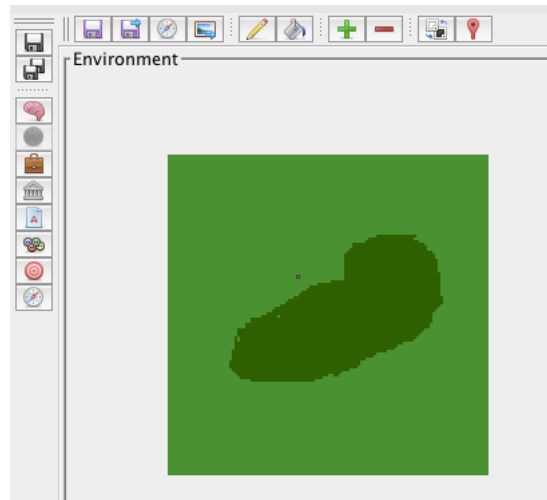


FIGURE 16 – Environnement

Maison a un effet permanent *Dormir* qui ajoute 40.0 à l'attribut *Sommeil* de l'agent à chaque fois qu'il utilise cet aménagement.

## 11.2 Cogniton, Plan, Actions

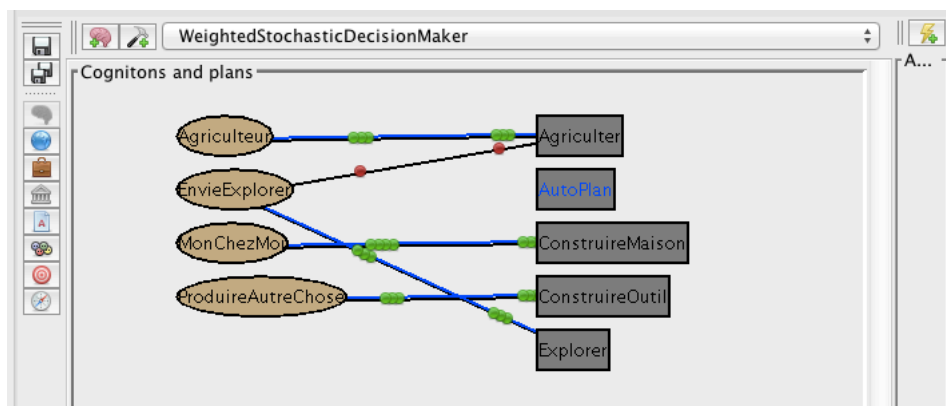


FIGURE 17 – Environnement

La figure 17 présente la *cognition* : l'ensemble des co-

gnitons (Agriculteur, EnvieExplorer, MonChezMoi, ProduireAutreChose) et plans (Autoplan et Agriculter, ConstruireMaison, ConstruireOutil, Explorer) ainsi que les liens conditionnels (bleu) et d'influence (noir).

Les billes vertes correspondent à une influence positive, les rouges à une influence négative et leur quantité dénonce le poids.

Par exemple, Agriculter est lié conditionnellement au cogniton Agriculteur (qui influence aussi positivement ce plan) ; par contre le cogniton EnvieExplorer influence négativement ce même plan.

Un clic droit sur chaque cogniton donne le détail sur celui-ci (nom, type et pourcentage d'agent qui auront ce cogniton à leur création) et sur les liens le concernant.

*Remarque : le dosage du pourcentage est important dans le modèle et pour la simulation*

Pour accéder à la description des actions d'un plan (clic gauche), si les actions L double cliquer sur elles jusqu'à déploiement complet des actions du plan.

## **Autoplan**

2 actions A\_ChangeAttribute (portant sur sommeil et vie)

## **Plan ConstruireMaison**

L'arbre des actions déployé est représenté sur la figure 18.

```

begin
  if j'ai une maison (n'importe où) then
    if j'ai une maison sur le patch où je suis then
      action interne 1 niveau 1
      j'utilise cette maison (et par suite je gagne 40 de sommeil)
    end
    else
      action interne 2 niveau 1
      je me déplace vers le patch de ma maison la plus proche
    end
  end
  else
    if sur le patch où je suis, existe un quelconque aménagement
    then
      action interne 1 niveau 2
      j'avance aléatoirement d'un pas (le patch étant occupé)
    end
    else
      action interne 2 niveau 2
      je crée une maison sur le patch
    end
  end
end
end

```

**Algorithm 1:** Plan ConstruireMaison

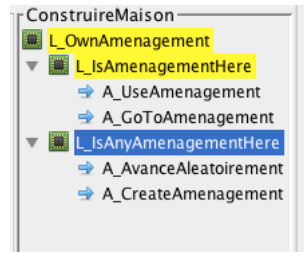


FIGURE 18 – Arbre des actions du plan ConstruireMaison

Cette action permet à l'agent de construire l'aménagement souhaité si celui n'en possède pas déjà un et qu'il n'y a pas d'aménagements sur le patch sur lequel il se trouve. Si le patch est occupé il ira en chercher un libre plus loin. Si l'agent possède déjà cet aménagement il ira l'utiliser.

Les autres actions de constructions sont construites de la même manière.

## 12 Observation de la simulation

Une fois la simulation lancée. Le tick 1 crée la position de départ du peuplement. Le tick 2 crée les agents. Il est possible, à partir du tick 3, de sélectionner un agent (avec un clic gauche) et de l'observer (au moyen d'un clic droit suivi de la sélection du choix *Observe one agent*). Cette fenêtre d'observation se met à jour automatiquement à chaque tick de la simulation. Elle permet ainsi d'observer facilement l'évolution d'un ou plusieurs agents au cours de la simulation.

### 12.1 La fenêtre d'observation d'un agent

Le premier onglet, l'onglet *Mind*, qui s'ouvre automatiquement, représente les cognitons et les plans présents dans l'esprit de l'agent. Il permet d'observer rapidement la répartition des cognitons et des plans dans l'esprit de l'agent (cf. Figure 19).

Cela permet ainsi, au moyen du graphique *Cogni weight*, de voir que, au tick auquel a été prise cette impression d'écran, l'agent observé possédait les quatre cognitons *EnvieExplorer*, *Agriculteur*, *MonChezMoi* et *ProduireAutreChose* avec un poids un peu plus important pour le cogniton *Agriculteur*. De même on peut aussi voir que cet agent est susceptible, à ce tick, d'effectuer les plans *Explorer*, *Construire Outil*, *Construire Maison* et *Agriculter* avec un peu plus de chances pour le plan

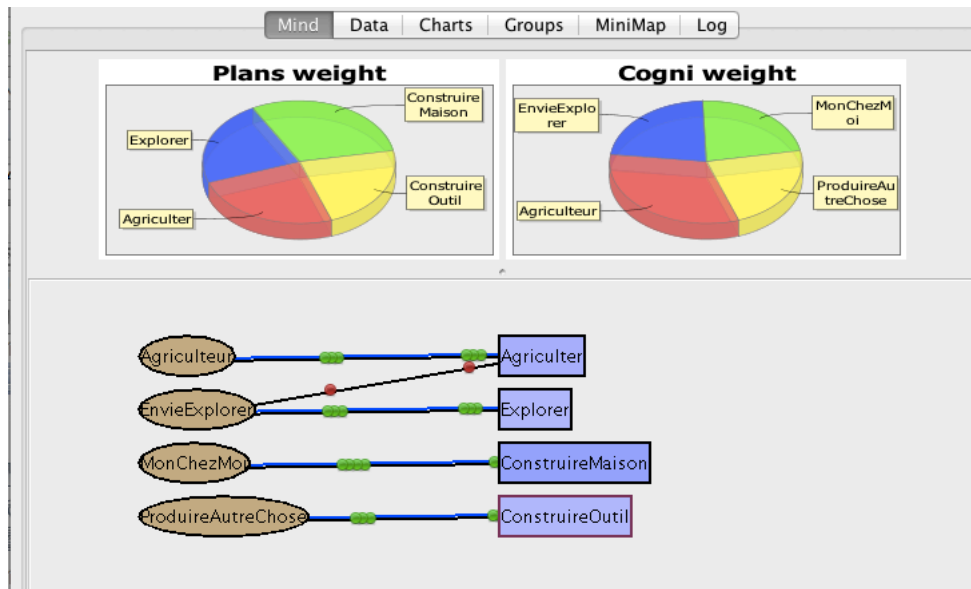


FIGURE 19 – Fenêtre d’observation de la constitution de l’esprit de l’agent

Construire Maison et Agriculteur.

L’onglet *Data* présente une vision globale des données concernant l’agent, à savoir, le poids de ses cognitons, le poids des plans qu’il est susceptible d’effectuer, la valeur de chacun de ses attributs, son inventaire, le plan qu’il effectue au cours de ce tick, ainsi que le type de patch sur lequel il se trouve (cf. Figure 20) ou même les groupes auxquels il appartient.

Dans le cas de cette simulation et à ce tick précis, on peut voir que l’agent observé possède le cogniton Agriculteur, lequel a un poids de 1.45, les autres cognitons présents dans l’esprit de l’agent ont un poids de 1.0 ce qui confirme l’information donnée par le premier onglet, il en est de même concernant les plans. On peut aussi

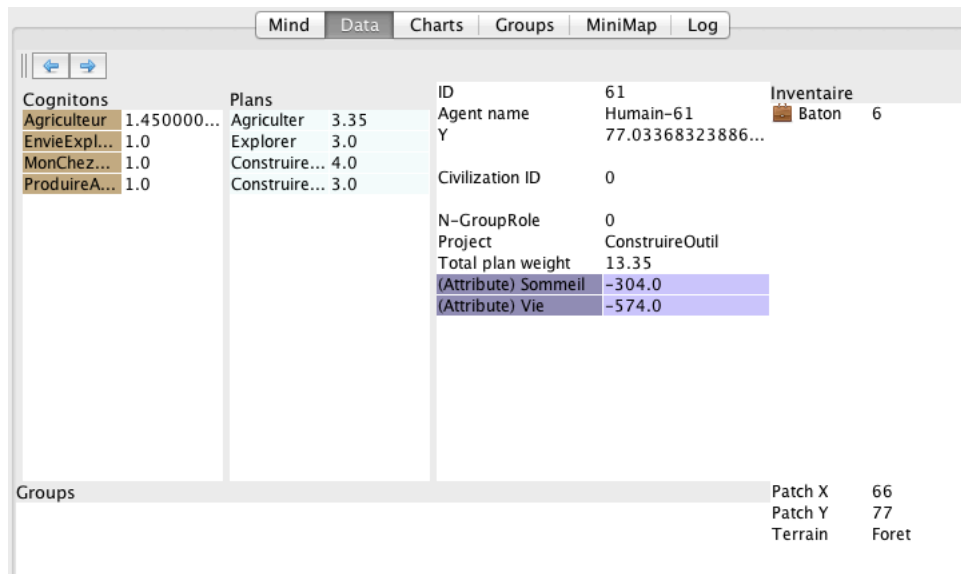


FIGURE 20 – Fenêtre d’observation de l’agent

constater que l’agent possède 6 objets Bâton dans son inventaire et que son plan actuel est Construire\_Outil. Il est donc prévisible qu’au prochain tick, l’agent crée un outil à partir de ces bâtons. De plus on observe que l’agent possède ces deux attributs Vie et Sommeil largement en négatif, ce qui est normal, car rien n’a été prévu, dans cette simulation, lors de la création de la cognition de l’agent, pour pousser ce dernier à consommer de la nourriture ou à aller se reposer si ses attributs atteignent un seuil critique. Il aurait été possible de le faire en définissant un *trigger* portant sur l’attribut voulu et déclenchant l’apparition d’un cogniton influençant très positivement un plan consistant à augmenter la valeur de cet attribut. Enfin cet onglet nous permet



de constater que l'agent n'appartient à aucun groupe et qu'il se situe sur un patch de type Forêt.

L'onglet *Charts* contient deux graphiques représentant, pour le premier, la répartition des attributs de l'agent, et le second, un aperçu rapide de la valeur de chaque plan de l'agent. Passer la souris sur chaque composante des graphiques permet d'obtenir plus d'informations (cf. Figure 21).

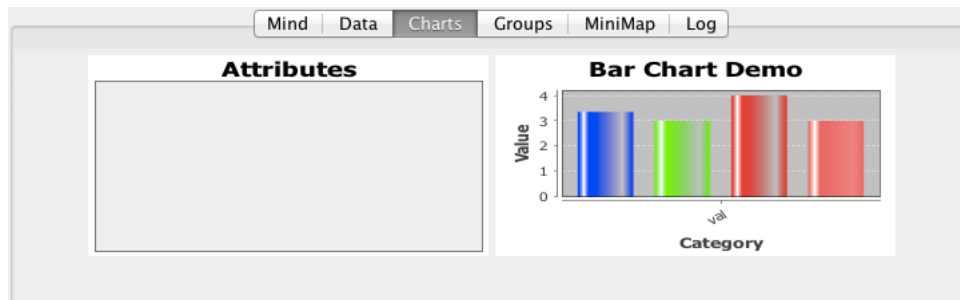


FIGURE 21 – Fenêtre d'observation des attributs et plans de l'agent

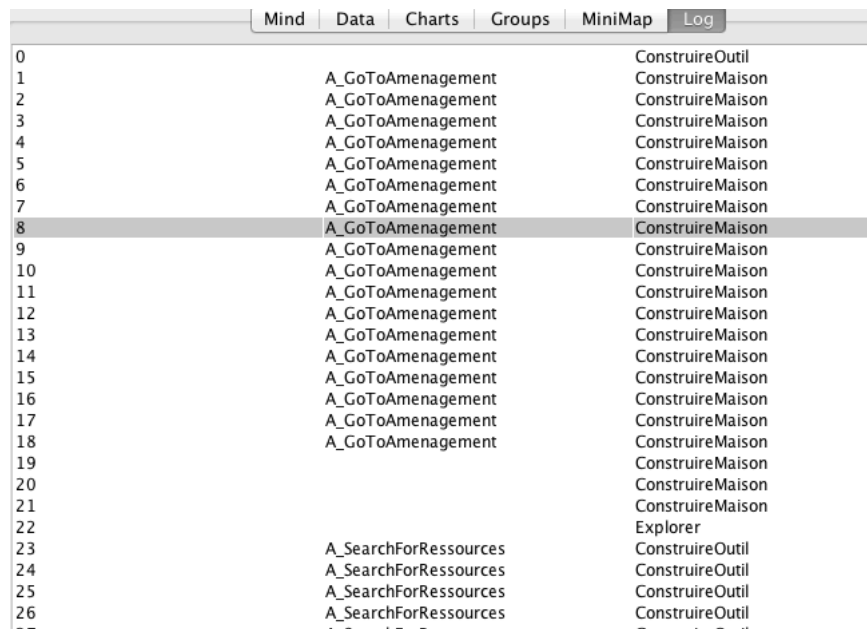
Cet onglet permet ainsi de confirmer que les valeurs des attributs de l'agent sont négatives car elles n'apparaissent pas dans le graphique et nous montre le poids de chaque plan susceptible d'être effectué par l'agent sous forme d'un diagramme. Pour savoir quel plan est représenté par quelle couleur il suffit de passer la souris au-dessus de celui-ci.

L'onglet *Groups* permet d'avoir une représentation des groupes auxquels appartient l'agent observé, ainsi que son rôle dans ces groupes. L'agent n'appartenant à aucun groupe cet onglet ne nous apprendra rien dans

cette simulation, dans d'autres simulations nous pourrions observer les rôles de l'agent observé dans les différents groupes auxquels il appartiendrait.

L'onglet *Minimap* permet d'avoir un zoom sur la position de l'agent sur la carte.

Enfin, l'onglet *Log* permet d'avoir un récapitulatif des plans et des actions effectués par l'agent à chaque tick (cf. Figure 22).



Tick	Action	Result
0		ConstruireOutil
1	A_GoToAmenagement	ConstruireMaison
2	A_GoToAmenagement	ConstruireMaison
3	A_GoToAmenagement	ConstruireMaison
4	A_GoToAmenagement	ConstruireMaison
5	A_GoToAmenagement	ConstruireMaison
6	A_GoToAmenagement	ConstruireMaison
7	A_GoToAmenagement	ConstruireMaison
8	A_GoToAmenagement	ConstruireMaison
9	A_GoToAmenagement	ConstruireMaison
10	A_GoToAmenagement	ConstruireMaison
11	A_GoToAmenagement	ConstruireMaison
12	A_GoToAmenagement	ConstruireMaison
13	A_GoToAmenagement	ConstruireMaison
14	A_GoToAmenagement	ConstruireMaison
15	A_GoToAmenagement	ConstruireMaison
16	A_GoToAmenagement	ConstruireMaison
17	A_GoToAmenagement	ConstruireMaison
18	A_GoToAmenagement	ConstruireMaison
19	A_GoToAmenagement	ConstruireMaison
20	A_GoToAmenagement	ConstruireMaison
21	A_GoToAmenagement	ConstruireMaison
22		Explorer
23	A_SearchForRessources	ConstruireOutil
24	A_SearchForRessources	ConstruireOutil
25	A_SearchForRessources	ConstruireOutil
26	A_SearchForRessources	ConstruireOutil
27	A_SearchForRessources	ConstruireOutil

FIGURE 22 – Fenêtre récapitulative des actions de l'agent

Nous pouvons ainsi voir que les derniers agissements de l'agent consistaient à aller se reposer à sa maison puis à aller chercher de quoi fabriquer des outils.

## 12.2 La fenêtre d'observation générale

Il est aussi possible d'observer le comportement général des agents au moyen des onglets *Agents*, *Options*, *Performances*, *Tableau de bord* et *Charts* présentés précédemment.

L'onglet *Agents* correspond à l'onglet *Data* présenté précédemment à ceci près qu'il est muni de deux flèches avant et arrière permettant de facilement passer de l'observation d'un agent à un autre.

L'onglet *Options* permet de définir les préférences d'affichage de la fenêtre *WorldViewer*. Il est muni de plusieurs options :

- *Affichage des plans* : permet de n'afficher que les agents effectuant un certain plan à sélectionner dans la liste déroulante. Les agents n'effectuant pas ce plan seront minimisés et ne seront visibles qu'en cas de zoom.
- *PheroMap* : permet de colorer les patches non plus en fonction de leur type de terrain mais en fonction de la valeur de la ressource sélectionnée pour chaque patch. Plus la ressource est présente sur le patch, plus ce dernier sera coloré en bleu.
- *ShowSpecificGroupAndRole* : permet de colorer les agents en fonction de leur appartenance à un groupe.

L'onglet *Performances* quand à lui affiche les ressources matérielles utilisées par la simulation.

L'onglet *Tableau de bord* permet, en cliquant sur l'icône *Mettre à jour les données* représentée par une double disquette, de visualiser pour chaque cogniton, le pourcentage d'agents présents dans la simulation le possédant.

Enfin l'onglet *Charts* permet de voir l'évolution de la moyenne des poids de chaque plan dans l'esprit des agents composants la simulation, de même pour les attributs.