Université Montpellier II

Sciences et Techniques du Languedoc

Rapport de Travail Encadré de Recherche

MetaCiv : DU CHASSEUR-CUEILLEUR AUX EMPIRES.

Auteurs :

Bruno YUN
Bruno.yun@etud.univ-montp2.fr
François SURO
Francois.suro@etud.univ-montp2.fr
Lionel FERRAND
Lionel.ferrand@etud.univ-montp2.fr
Alexandre VALIERE
Alexandre.valiere@etud.univ-montp2.fr

Encadré par :
Jacques Ferber, Mathieu Lafourcade

24 mai 2015

Table des matières

Introduction

Par le passé, le monde de l'informatique privilégiait des intelligences artificielles centralisées. Les difficultés rencontrées par l'utilisation d'une telle méthode a engendré la recherche de solutions différentes notamment à travers l'étude de phénomènes sociaux collectif, c'est la naissance des systèmes multi-agents.

Un système multi-agents est un système composé d'un ensemble d'agents, situés dans un certain environnement et interagissant selon certaines relations. Un agent est une entité caractérisée par le fait qu'elle est, au moins partiellement, autonome. Cette méthode est utilisé dans de nombreux domaines : le cinéma, l'industrie, la recherche, etc... Dans le cadre de notre projet de recherche, nous serrons amenés à utiliser et à modifier le logiciel MetaCiv. Cette plateforme permet de simuler des civilisations notamment humaine en utilisant les systèmes multi-agents. Elle a pour but de ne pas seulement être utilisé dans le domaine de l'informatique mais aussi de pouvoir favoriser la recherche dans d'autres domaines comme la sociologie ou l'archéologie.

Ce projet porte sur l'élaboration d'une intelligence artificielle simulant des civilisations proches de celles existantes chez l'humain en utilisant les systèmes multi-agents. Nous avions besoin de créer une nouvelle modélisation grâce au logiciel MetaCiv et dans ce but nous avons fait évoluer ce logiciel. La simulation de phénomènes sociaux humains étant très complexe, nous avons réalisé des modèles basés sur des sociétés qu'on pouvait retrouver dans la préhistoire et l'antiquité. Malgré que ces modèles soient basiques, il nous fallait tout de même modifier le logiciel pour améliorer sa stabilité et ajouter des outils facilitant la création de civilisations. Pendant toute la réalisation de ce projet, nous avons dû travailler en collaboration avec un autre groupe pour leurs fournir les outils nécessaire à la réalisation de leur recherche.

Nous avons réalisés deux modèles ayant pour but de montrer l'émergence de façon différente. Un modèle basé sur les fonctionnalités de groupe et l'attribution aléatoire d'une spécialisation des membres d'une communauté. Un autre basé sur le renforcement d'une compétence à travers la réalisation ou l'échec d'une activité ainsi que l'observation de son environment. Pour permettre le travail de modélisation, nous avons commencé par sécuriser les sauvegardes des modèles puis nous avons ajouté diverses fonctionnalités dont le système de gestion des constantes qui nous permet un paramétrage plus efficace des modèles ainsi que les schéma cognitifs qui permettent de séparer le modèle de comportement des agents de l'environnement unique du modèle.

Dans ce document, nous reviendrons sur les difficultés rencontrés lors de notre initiation à MetaCiv et à ses codes sources. Nous expliquerons l'organisation de notre groupe pendant la réalisation de ce projet et nous conclurons sur le travail effectué ainsi que fournir des pistes d'évolutions futures du logiciel MetaCiv.

1 Présentation de la plateforme MetaCiv

Il existe aujourd'hui de nombreuses plateformes de modélisation de systèmes complexes se basant sur la programmation par agents. Dans l'équipe de travail SMILE, plusieurs plateformes ont déjà été mises au point, Turtlekit [Mic02] et Madkit [GFM00] largement inspirées des plateformes basées sous StarLogo (http ://education.mit.edu/starlogo/). Cela fait bien plus d'une année qu'une plateforme spécifique dénommée MetaCiv à vu le jour. Nous allons rappeler très rapidement ses principes et concepts.

1.1 Principe

MetaCiv est une plateforme de modélisation par agents dédiée la conception de modèles relatifs des systèmes complexes (biologiques, environnementaux, etc...). Elle est basée sur les fonctionnalités de la plateforme Turtlekit déjà existante. Elle bénéficie naturellement des fonctionnalités des plateformes précédentes, notamment du concept Agent Group Role (AGR). De plus, elle intègre les principes dits des quadrants [FST09]

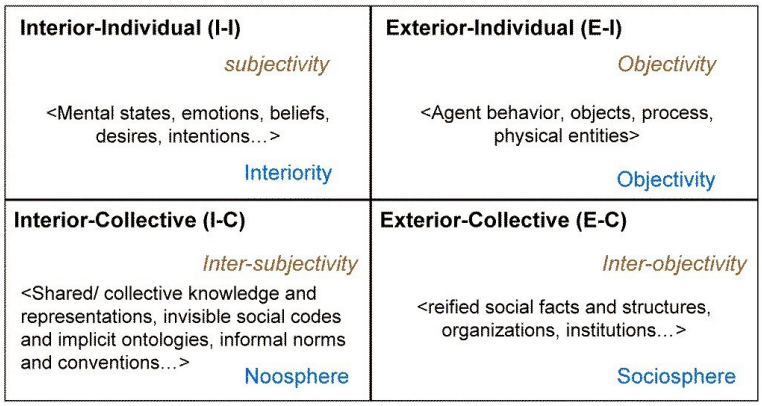

FIGURE 1 – Quadrants.

Un système multi-agents selon le principe des quadrants est perçu comme un ensemble de composants organisés selon deux axe : individuel-collectif d'une part et interiorité-exteriorité d'autre part. Ainsi le quadrant I-I définit ce qui constitue le fonctionnement interne de l'agent, sa manière de penser, ses idées, ses croyances. Le quadrant E-I quand lui, définit l'image que donne l'agent au monde, savoir son comportement, ses actions, ses possessions, son apparence. Ensuite, le quadrant I-C correspond aux idées partagées par un groupe d'individus, les normes et codes s'appliquant ce groupe, les valeurs communes et connaissances partagées. Enfin, le quadrant E-C s'applique à tout ce qui donne une représentation de groupe sociaux, savoir, par exemple, les bâtiments publics, les monuments, les infrastructures. MetaCiv est donc un framework de modélisation de société humaines dans lesquelles l'espace, la culture et la structure sociale jouent un rôle important. On peut ainsi faire une représentation globale de MetaCiv en reprenant les quadrants de MASQ sous la forme MASQ-ML le langage de modélisation de MASQ [FST09].

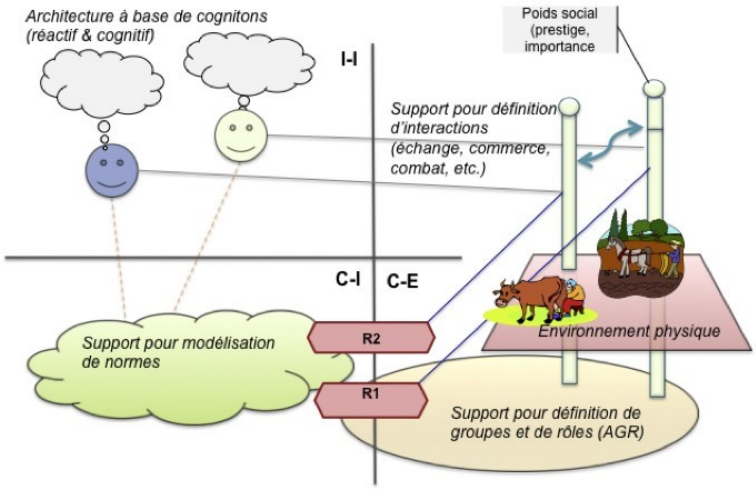

FIGURE 2 – MetaCiv exprimé en MASQ-ML.

1.2 Concepts

Comme vu précédemment MetaCiv est une plateforme de simulation par agents dans laquelle nous postulons que chaque agent possède un cognition . De plus ces agents évoluent au sein d'un environnement évolutif qui impact leurs décisions. Pour cela, dans la plateforme, chaque agent possède un "esprit" qui concentre les perceptions, la mémoire (qui enregistre les actions effectuées ainsi que les influences subies) ainsi qu'une faculté de décision.

La figure des quadrants résume le fonctionnement actuel et les extensions que nous souhaitons y apporter. Tout agent possède donc un esprit qui est un agrégat de cognitons. Il décide d'exécuter un plan, qui est un ensemble d'actions.

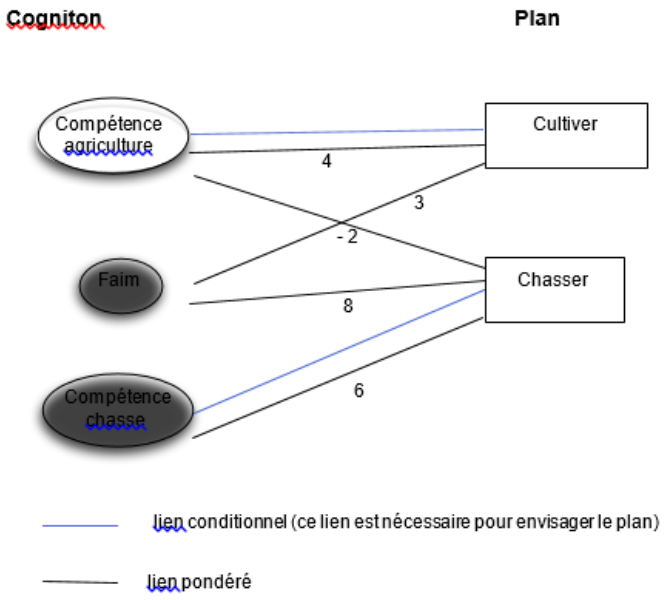

FIGURE 3 – Exemple de prise de décision.

Prenons l'exemple de la figure précédente, dans une société d'agents agriculteurs l'utilisateur prévoit deux plans d'actions possibles, Cultiver et Chasser, les cognitons Compétence Agriculture, Faim, et Compétence chasse, ainsi que les liens entre cognitons et plans. Pour pouvoir exécuter ces plans il faut disposer des cognitons Compétence agriculture ou Compétence chasse : ceci est exprimé sous la forme de liens conditionnels. Les liens pondérés sont rajoutés entre cognitons et plans d'actions pour exprimer l'influence d'un cogniton sur un plan (un poids négatif traduisant un degré d'inhibition et un poids positif un degré de facilitation). Un agent qui possède les trois cognitons de la figure ci-dessus décidera de Cultiver ou Chasser après évaluation des liens pondérés ici Cultiver 7 et Chasser 12. Le choix entre ces deux plans est en fait dépendant d'une randomisation prenant en compte les poids calculés précédemment, ceci afin de pallier les comportements majoritaires. Nous avons abstrait le schéma de fonctionnement de MetaCiv dans la figure suivante, en présentant les concepts existants et ceux qui constitueront les extensions visées.

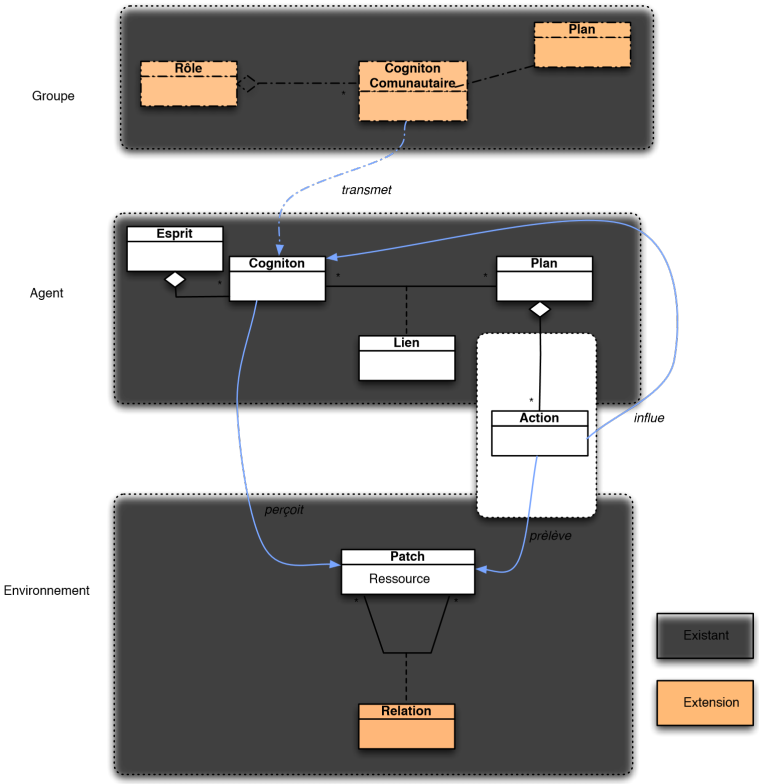

FIGURE 4 – Schéma de fonctionnement de MetaCiv.

Nous retrouvons donc le fait qu'un agent dispose d'un esprit constitué de cognitons. Ceux ci sont liés aux plans que peut exécuter l'agent par des liens conditionnels ou d'influences. Les actions prélèvent des ressources dans les patches et influent sur les cognitons de l'agent. Les perceptions de l'environnement se font aussi grâce aux cognitons. Les extensions sur lesquelles nous allons réfléchir, la lueur de la bibliographie, concernent le niveau groupe et la transmission de la connaissance communautaire d'une part, et d'autre part les relations spatio temporelles au niveau de l'environnement (celui ci ne se cantonnant plus au rôle de support).

2 Modélisations de civilisations

2.1 Introduction

Le but de nos modélisation est de construire un ensemble de modèles de développement de civilisations (IA de développement), allant du chasseur-cueilleur à l'empire. C'est en essayant d'imiter les civilisations humaines et leurs évolutions à travers le temps que l'on sera amené à mettre en valeur les aspects d'émergence dans les systèmes multi-agents. Nous expliciterons les différentes notions nécessaires à la compréhension de nos modélisations. Créer une modélisation de civilisation humaine allant du chasseur-cueilleur à l'empire étant trop complexe, nous avons favorisés la création de deux modèles distincts

qui chercherons à répondre de manière différente à l'émergence. Le premier modèle, "Cueilleur-Artisan-Agriculteur" va chercher à faire fonctionner une transition entre deux âges à travers l'utilisation de notions de groupes alors que le modèle "Nomade-Sédentaire" recherchera à induire une émergence à travers l'environnement dans lequel évoluent les agents.

2.1.1 Les agents

Les agents sont vus sous l'angle *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, interagir avec leur environnement et construire des aménagements. Ils évoluent dans un environnement collectif agrégat de *patches*. Les cognitions vont influencer les plans, de manière pondérée (selon l'importance que l'utilisateur 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 influer sur ses cognitons. De même les actions peuvent directement ou indirectement via les objets ajouter ou influer les cognitons.

2.1.2 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. Les cognitons peuvent se cumuler, et ne sont donc pas des "états" mentaux. En fait, c'est la somme des cognitons qui représente réellement l'état mental de l'agent.

Il existe deux interactions entre un cogniton et un plan. La première rend un plan "accessible" par l'agent et la deuxième influence l'envie (ou la répulsion) d'effectuer un plan.

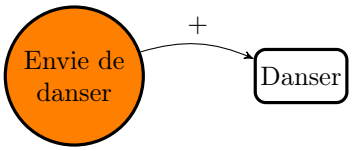

FIGURE 5 – Exemple d'un cogniton renforçant un plan (flèche positive).

Dans l'exemple précédent, on peut remarquer que le cogniton "Envie de danser" influence positivement l'envie de l'agent à effectuer le plan "Danser", mais ce dernier n'étant pas accessible à l'agent, il ne peut l'effectuer. On rajoute alors un lien "conditionnel" (qui peut éventuellement venir du même cogniton) permettant d'activer le plan.

Les cognitons peuvent avoir un "trigger". Il s'agit d'un dispositif permettant d'apparaître ou de disparaître le cogniton en question selon la valeur d'un attribut. Dans le cas contraire, les cognitons sont soit des "Starting cognitons", c'est à dire des cognitons présent dès le début de la simulation, soit des cognitons qui n'apparaissent qu'en réponse d'un plan.

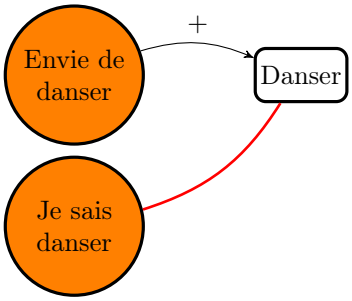

FIGURE 6 – Exemple d'un cogniton renforçant un plan (flèche positive) et d'un deuxième cogniton qui rend le plan accessible (trait rouge).

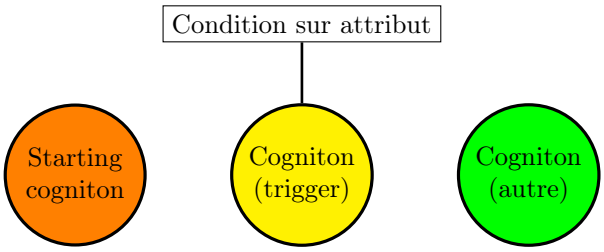

FIGURE 7 – Exemple des différents types de cognitons : Starting cogniton (en orange), cogniton activé par trigger (en jaune) et les cognitons ajoutés par des plans (en vert).

2.1.3 Catégorisation des cognitons

Pour mieux structurer l'organisation des cognitons, ceux-ci sont catégorisés en cinq types : *Skills, Traits, Beliefs, Percepts, Mèmes*. 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, change la couleur d'un cogniton dans l'interface de MetaCiv et permet de mieux repérer le type du cogniton.

— 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 pré-requis 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 *Mèmes* sont des croyances ou comportements culturels transmissibles. Ainsi, les mèmes 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 mèmes. Par définition, les mèmes sont transmissibles, et ils sont potentiellement héréditaires.

2.1.4 Les plans et les actions

Un plan est un ensemble de plusieurs actions génériques qui peuvent être partagées par un ou plusieurs plans. Une action peut prendre en entrée des paramètres (groupe, rôle, constante, attribut, nombre, aménagement, objet, etc..). Il existe deux types d'actions :

— Les actions qui modifient les paramètres d'un agent (attributs, position, inventaire, cognitons) ou l'environnement (récolte, construction de routes, etc...). Le nom d'une telle action est de la forme : A_nomDeLAction.
— Les actions conditionnelles sont généralement composées de deux actions internes. Elles exécutent la première action si la condition passée en paramètre est satisfaite et la seconde action sinon. Le nom d'une telle action est de la forme : L_nomDeLAction.

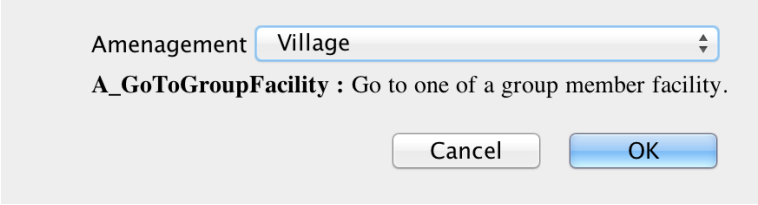

FIGURE 8 – Exemple de l'interface d'édition d'une A_action dans MetaCiv

.

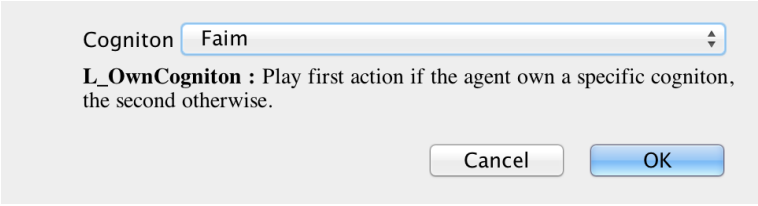

FIGURE 9 – Exemple de l'interface d'édition d'une L_action dans MetaCiv

.

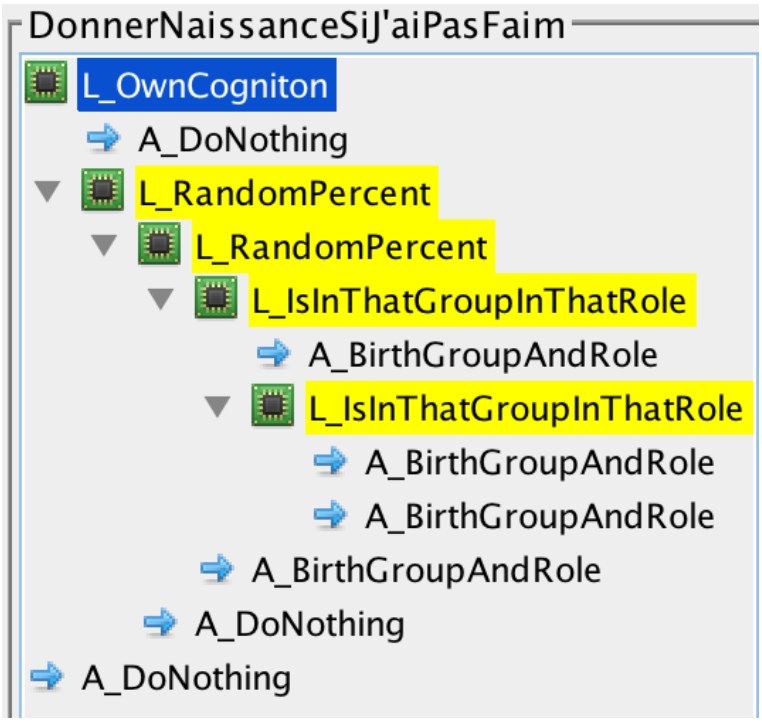

FIGURE 10 – Exemple de l'interface du plan "DonnerNaissanceSiJ'aiPasFaim" dans MetaCiv. Les actions conditionnelles sont repérées par une icône en forme de processeur et les autres actions par une icône en forme de flèche.

2.1.5 Les constantes

Les constantes peuvent être assimilés aux paramètres de la simulation. Elles peuvent être prise en paramètre d'une ou plusieurs actions mais ne peuvent être modifiés que par l'utilisateur du simulateur. La modification d'une constante est prise en compte même si le modèle est en cours d'exécution. Elles servent à modifier plus facilement les paramètres de la simulation dans toutes les actions dans lesquelles elles sont utilisées.

2.1.6 Les attributs

Les attributs sont des variables numériques personnelles à chaque agent. Elles peuvent être modifiés par des actions et leurs valeurs originales sont saisies par l'utilisateur. Elles servent à représenter les caractéristiques d'un agent.

2.1.7 Les groupes et les rôles

Lorsque plusieurs agents se rassemblent pour former un groupe, chaque agent de ce groupe possède un rôle. Chacun de ces rôles est défini de manière générique et ajoute des plans supplémentaires au moyen de nouveaux cognitons dits "culturons" et permettent ainsi la spécialisation de l'agent.

FIGURE 11 – Exemple de culturon (en marron).

2.2 Le modèle "Cueilleur-Artisan-Agriculteur"

Ce modèle va servir à générer une société basée sur la cueillette, l'artisanat et l'agriculture. Elle aura comme particularité d'attribuer un rôle spécifique à chacun des agents dans le but d'"Observer" l'émergence d'un comportement collaboratif au sein même d'une civilisation et ses conséquences sur la population.

Toutes les constantes mentionnées dans ce modèle sont données dans la section "Constantes". Les agents sont initialement au nombre de 81 lors du lancement de la modélisation.

2.2.1 L'environnement du modèle

En premier lieu, l'environnement dans lequel nos agents se déplacent est un espace fermé de $\mathbb{R}^2$. Il existe un total de quatre types de patches présents : Mer, Prairie, Terre Sterile et Forêt. Le modèle étant relativement basique, il n'existe que deux types de ressources disponibles : Les baies et le bois. Le tableau suivant donne le récapitulatif des différents patches et ressources disponibles.

Patch	Ressources		Valeur initiale		Croissance des ressources		Passabilité
Mer	Aucunes		0		0		Non
Terre Sterile	Aucunes		0		0		Oui
Prairie	Baies	Bois	10	1	0.2	0	Oui
Forêts	Baies	Bois	10	2	0.1	0.2	Oui

FIGURE 12 – Tableau récapitulatif des patches et des ressources.

Un patch de terre stérile ne produit pas de ressources alors que les patches Prairie et Forêts produisent en permanence des baies. On pourra remarquer que seules les forêts produisent du bois même si les prairies possèdent une valeur en bois initiale. Un patch peut être récolté dès lors que les ressources en bois ou en baies sont supérieures ou égales à 1. Une ressource "baies" est alors transformée en un objet baie (de même pour le bois).

Le point d'apparition des agents se situe au milieu de l'environnement et plusieurs prairies sont présentes aux alentours. On peut aussi distinguer une grande forêt dans la partie droite de l'environnement.

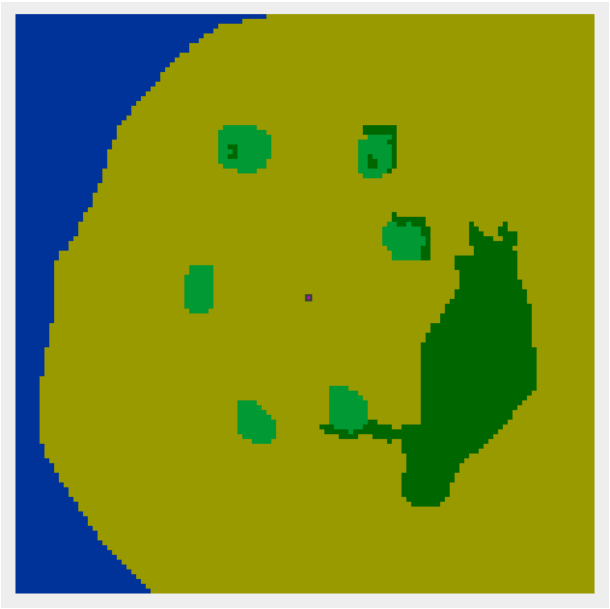

FIGURE 13 – environnement de la modélisation

Il convient de remarquer que lorsque les agents apparaissent dans l'environnement, ils sont éparpillés autour du point d'apparition et non dessus.

2.2.2 Les attributs du modèle

Il existe six attributs dans ce modèle : Vie, Age, CompetenceArtisanat, Energie, CompetenceCueilleur et EsperanceDeVie.

- L'attribut Energie est baissée chaque tour d'une certaine quantité ($BaisseEnergieParTick$), et n'est remontée qu'en mangeant de la nourriture (Baies).
- L'âge d'un individu est augmenté à chaque tick ($AgeParTick$).
- La compétence en artisanat reflette la dextérité d'un individu pour l'artisanat et n'est augmentée qu'après avoir fabriqué divers objets.
- L'attribut "CompetenceCueilleur" reflette l'aptitude d'un individu pour la recherche et la récolte de baies et n'est augmentée qu'après avoir ramené des baies au village.
- La vie d'un individu est utilisée lors des combats entre deux agents. (Non implémentée pour l'instant).
- L'attribut EsperanceDeVie est initialisée dans le "BirthPlan". Ce plan est lancé automatiquement à la création d'un agent et permet l'initialisation de plusieurs paramètres. Cet attribut fixe l'âge à partir duquel l'agent est susceptible de mourir. Afin de donner une espérance de vie propre à chacun des agents, nous générons la fonction de masse de la loi de Poisson de paramètre $k = 20$ et $\lambda = 10$.

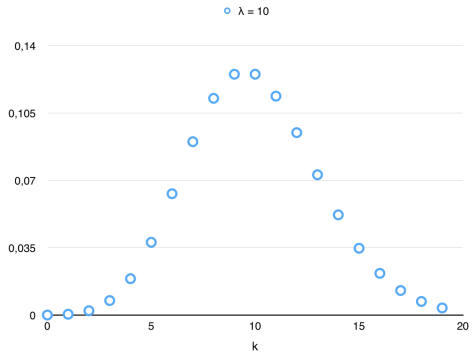

FIGURE 14 – Représentation de la fonction de masse de formule $P(k) = \frac{\lambda^k}{k!}e^{-k}$

On calcule ensuite sa fonction de répartition et on génère une variable suivant cette loi de Poisson en deux étapes :
— On tire un nombre r entre 0 et 1.
— On trouve l'entier i tel que $P(X \leq i-1) \leq r \leq P(X \leq i)$.
Pour avoir une valeur légèrement plus précise, on trouve le nombre approché i' tel que :

$$i' = \frac{r}{\alpha} - \frac{P(X \leq i)}{\alpha} + i \text{ avec } \alpha = P(X \leq i) - P(X \leq i-1).$$

Enfin, pour avoir une espérance de vie plus acceptable, nous posons :

$$EsperanceDeVie = 5 * i' \text{ avec } i' \in [0, 20]$$

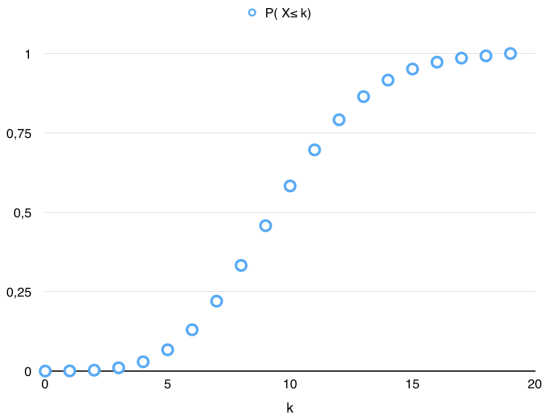

FIGURE 15 – Représentation de la fonction de répartition de la loi de Poisson de paramètre $\lambda = 10$ et $k = 20$.

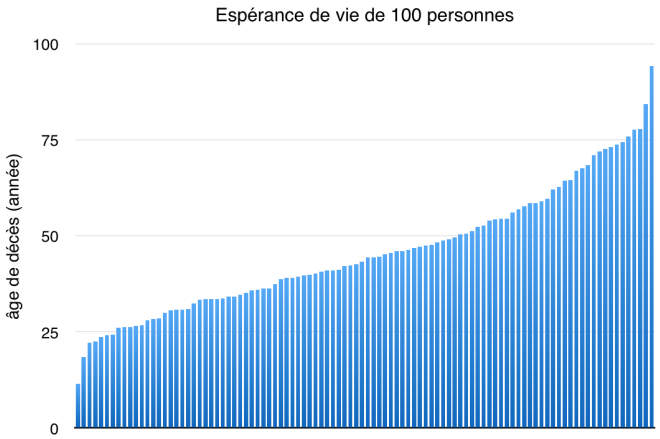

FIGURE 16 – Représentation triée de la valeur de l'attribut "EsperanceDeVie" de 100 agents.

Ainsi, sur les cents agents, trois ont un attribut "EsperanceDeVie" inférieur ou égal à 20, vingt-trois agents ont un attribut "EsperanceDeVie" supérieur ou égal à 60 et le reste des soixante-quatorze agents ont une "EsperanceDeVie" comprise entre 20 et 60.

Attribut	Valeur par défaut
Vie	50
Age	0
CompetenceArtisanat	0
CompetenceCueilleur	0
Energie	100
EsperanceDeVie	Valeur initialisé par le "BirthPlan".

FIGURE 17 – Tableau récapitulatif des attributs d'un agent et de leurs valeurs par défauts.

2.2.3 Les plans par défaut

Lorsque les agents arrivent dans l'environnement, ils possèdent plusieurs plans par défauts. Il en existe quatre : Birthplan, Standard, Ne rien faire et DonnerNaissanceSiJ'aiPasFaim.

— Le plan "Standard" est effectué par chaque agent à chaque tick et gère tout ce qui relève de la physiologie de l'agent. Elle modifie donc les attributs : âge et Energie. Elle est également responsable des tests pour détruire les agents. On peut également remarquer la présence de la ligne "Créer Un Groupe" dans le plan standard, cette ligne crée un groupe et donne à l'agent courant le rôle de "Cueilleur".

— Le plan "Birthplan" qui est lancé à chaque fois qu'un agent est crée ne sert qu'à initialiser l'attribut "EsperanceDeVie".

— Le plan "Ne rien faire" permet aux agents ayant du bois d'augmenter leurs attributs CompetenceEnArtisanat (avec une certaine probabilité).

— Le plan "DonnerNaissanceSiJ'aiPasFaim" permet à un agent de créer un autre agent (avec une certaine probabilité) s'il n'a pas le cogniton "Faim".

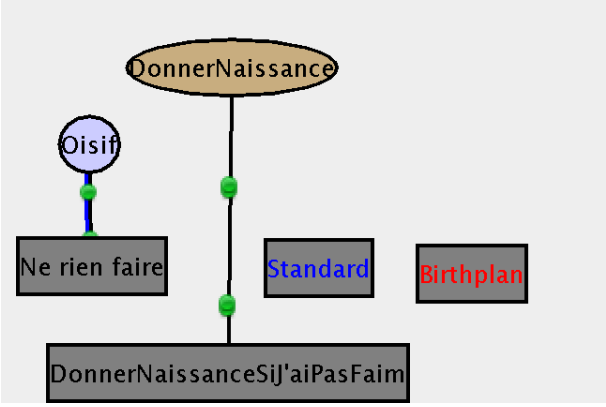

FIGURE 18 – Affichage des plans et cognitons associés dans MetaCiv (les couleurs des cognitons n'indiquent pas le type de cogniton).

Standard

- Si ($Energie = 0$) alors $Mourir$.
- Si (Age > EsperanceDeVie) alors $Mourir$ (avec une certaine probabilité).
- Baisser $Energie$ de $BaisseEnergieParTick$.
- Augmenter Age de $AgeParTick$.
- Créer un groupe et s'attribuer le rôle de *cueilleur* si je n'ai pas de groupe (avec une certaine probabilité).

Ne rien faire

- Si j'ai du bois, fabriquer une sculpture et augmenter $CompetenceEnArtisanat$ de $CompetenceParSculture$(avec une certaine probabilité).

+

Oisif

Birthplan

Initialiser l'attribut EsperanceDeVie

DonnerNaissanceSiJ'aiPasFaim

- Si je possède le cogniton "Faim", alors ne rien faire.
- Sinon donner naissance à un agent (avec une certaine probabilité)
  - Avec une certaine probabilité, lui donner le même rôle.
  - Sinon lui donner le rôle de Cueilleur.

+

Donner Naissance

FIGURE 19 – Figure des plans "Ne Rien Faire", "Standard", "DonnerNaissanceSiJ'aiPasFaim" et "Birthplan".

On remarquera que les plans "Birthplan" et "Standard" n'ont pas besoin de cognitons pour être effectués. Le plan "Ne rien faire" est accessible dès le lancement de la simulation puisqu'il est activé par le cogniton "Oisif". Même si le plan "DonnerNaissanceSiJ'aiPasFaim" est influencé, il n'est pas activé.

En plus de ces deux plans, nous avons aussi plusieurs plans qui permettent de gérer la $Faim$ mais aussi le rapatriement des $Baies$ au village : "Consommer", "AllerChercherAuVillage" et "RamenerBaiesAuVillage".

— Le plan "Consommer" utilise une baie de l'inventaire de l'agent pour remonter ses vies s'il en possède ou augmente le poids du cogniton "FaimEtRienDansMonSac" dans le cas contraire.
— Le plan "AllerChercherAuVillage" ramène l'agent au village du groupe pour récupérer des baies. Si des baies sont présentes dans l'inventaire du village, il baisse le poids du cogniton "FaimEtRienDansMonSac".
— Le plan "RamenerBaiesAuVillage" ramène l'agent au village du groupe afin qu'il y dépose son surplus de baies. Une fois cette opération effectuée, son cogniton "TropDansMonSac" est baissé.

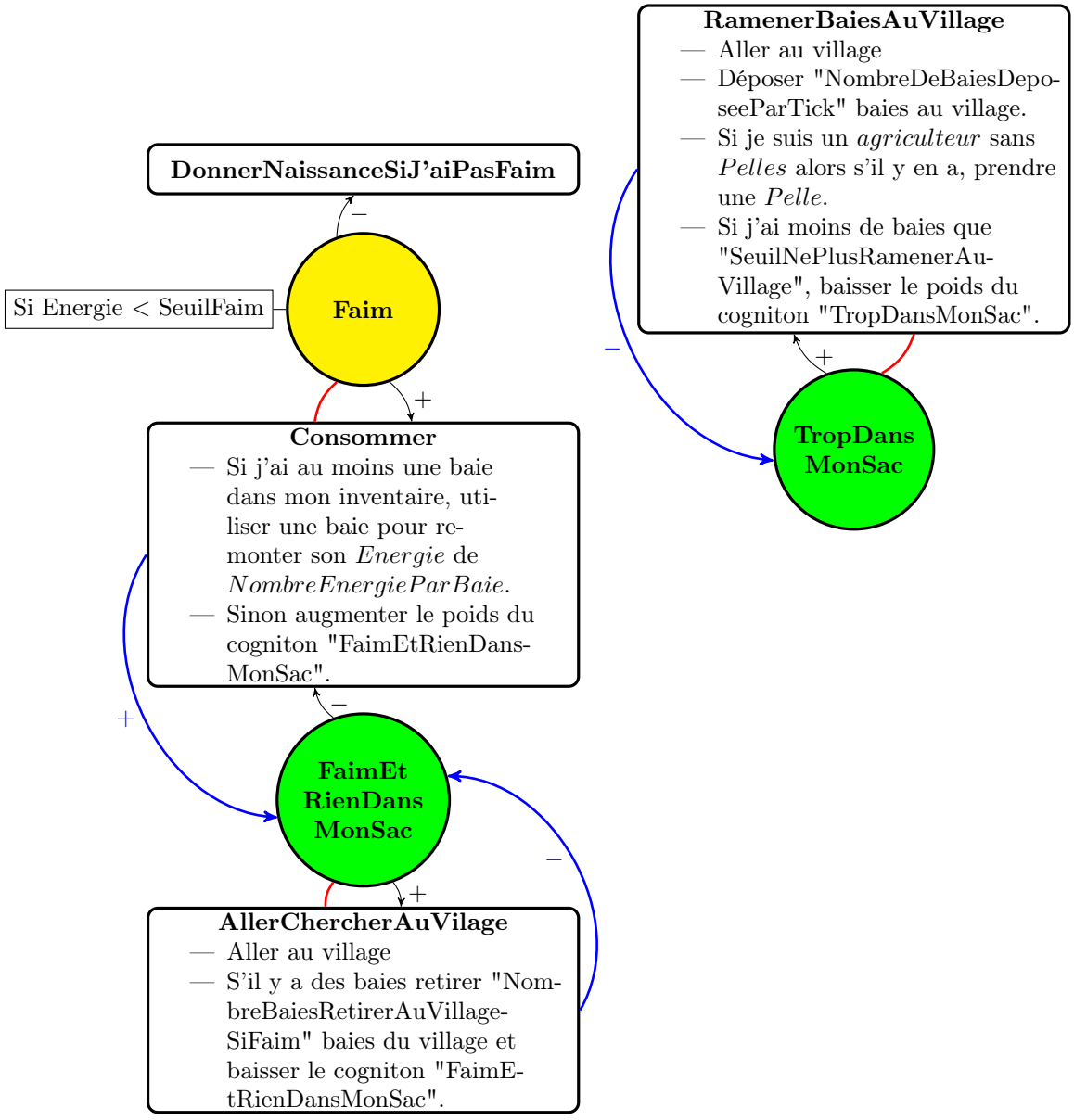

FIGURE 20 – Figure des plans "DonnerNaissanceSiJ'aiPasFaim", "Consommer", "AllerChercherAuVillage" et "RamenerBaiesAuVillage".

On remarquera que le plan "RamenerBaiesAuVillage" permet aux agriculteurs de récupérer des pelles au village (s'il y en a) lorsque ceux-ci viennent déposer des baies.

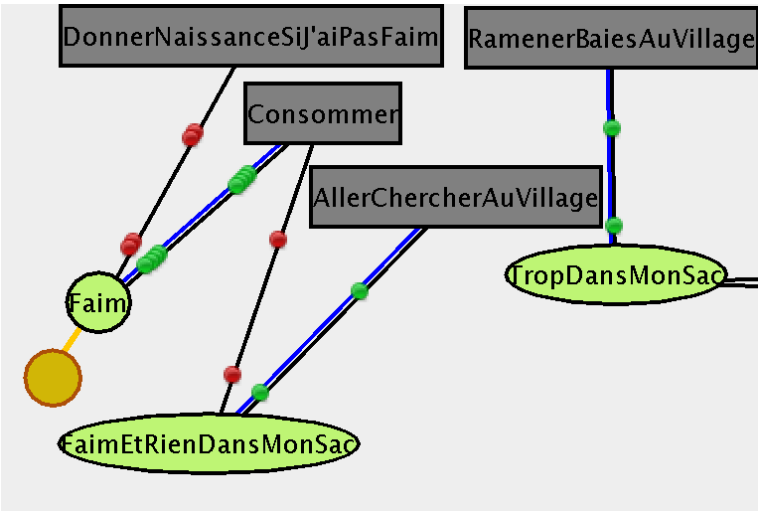

FIGURE 21 – Représentation des cognitons et plans associés dans MetaCiv.

2.2.4 Les rôles du modèle

Le cueilleur

Lorsqu'un groupe est créé, le premier individu de ce groupe est un *cueilleur*. La fonction principale de cet agent est bien sûr de cueillir mais aussi de recruter d'autres agents n'ayant pas de groupe et de construire le village du groupe (s'il n'y en a pas déjà un). Il possède également la capacité de devenir un *Agriculteur* ou un *Artisan*.

Le cueilleur possède donc cinq plans supplémentaires :

- Le plan "Cueillir" permet aux cueilleurs de partir à la recherche des baies et influence le cogniton "TropDansMonSac" pour le rapatriement des baies. Il est activé par le culturon "AllerChercherDesBaies".
- Le plan "recruter" permet aux agents de former des groupes. Il est activé par le culturon "Recruter".
- Le plan "Construire" permet aux agents de construire leur village, s'ils n'en possèdent pas. Il est activé par le culturon "ConstruireVillage". Une fois le village construit, le cogniton "PasBesoinDeVillage" est ajouté et inhibe le plan "Construire".‘
- Le plan "DevenirAgriculteur" est activé par le cogniton "DevenirAgriculteur'" mais n'est influencé que par le cogniton "BonCueilleur", lui-même activé que lorsque l'agent à son attribut "CompetenceCueilleur" supérieure ou égale à la constante "SeuilBonCueilleur". Ainsi, un cueilleur n'est susceptible de devenir un agriculteur que lorsque les conditions sont satisfaites.
- Le plan "DevenirArtisan" reprend le principe du plan "DevenirAgriculteur".

On remarquera que le plan "DonnerNaissanceSiJ'aiPasFaim" qui était influencé par le cogniton "DonnerNaissance" n'est activé que maintenant par le culturon "ReproduireCueilleur".

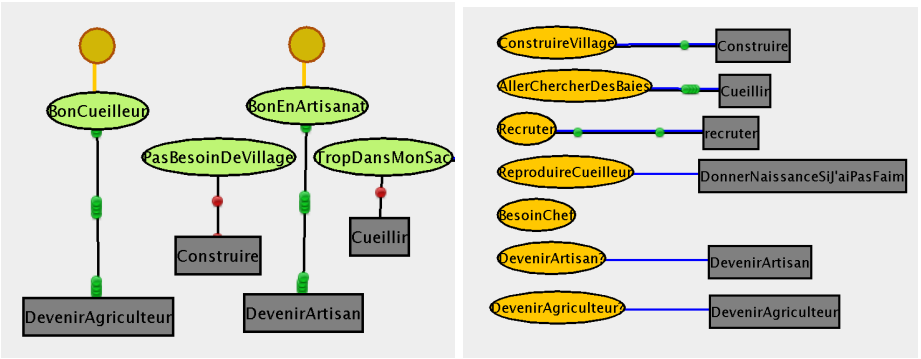

FIGURE 22 – Représentation des plan, culturons et cognitons du rôle cueilleur dans MetaCiv.

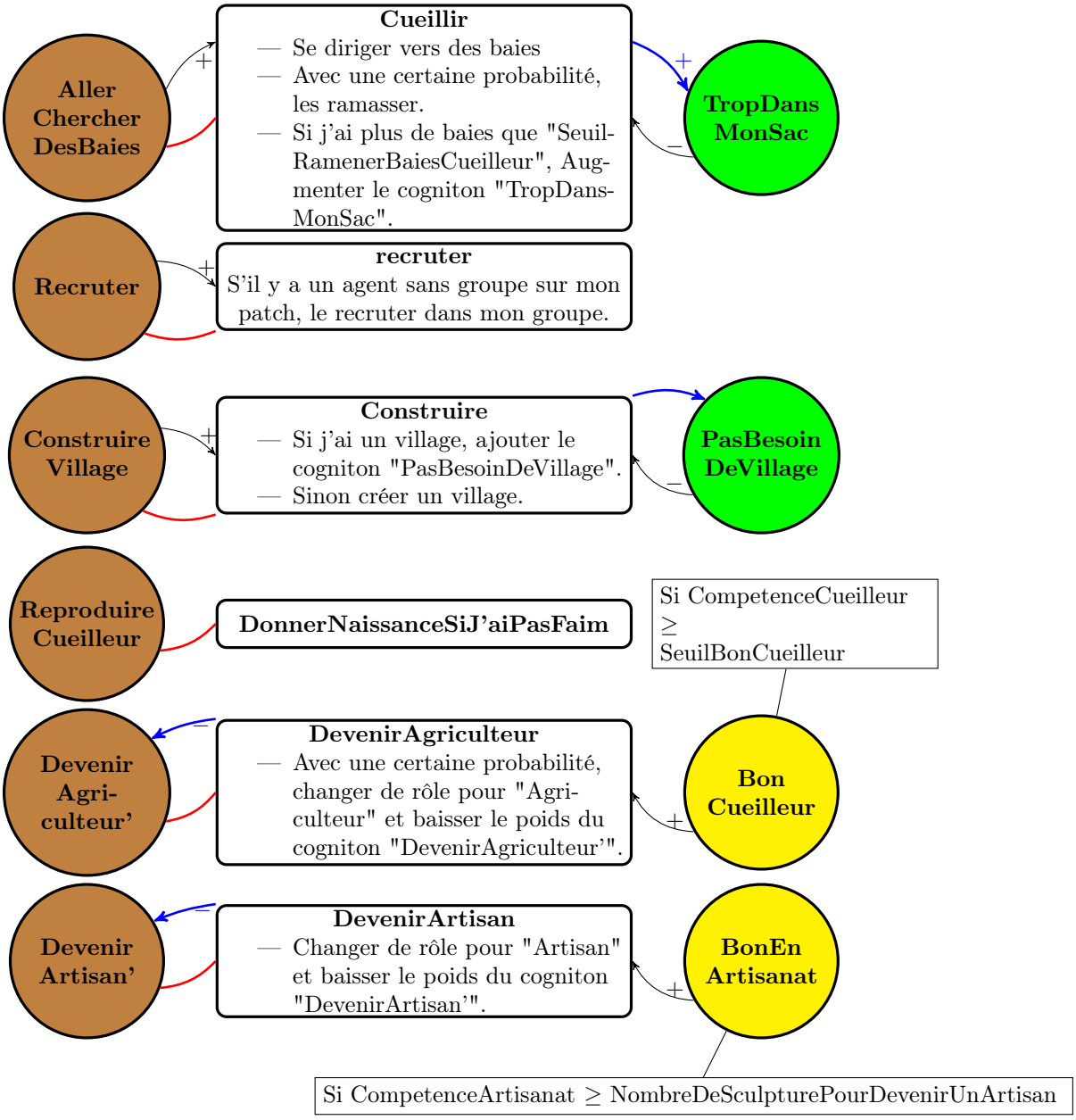

FIGURE 23 – Figure des plans d'un cueilleur et des interactions avec les cognitons associés.

Les artisans

Les artisans sont des agents qui construisent leurs huttes et partent à la recherche de bois pour construire des outils (ici des pelles) pour augmenter la production des agriculteurs. En contre-partie,

ceux-ci reçoivent des baies supplémentaires du village. Ils possèdent quatre plans qui leurs sont associés : ChercherBois, RamenerPelleAuVillage, ConstruirePelle et ConstruireHutte.

— Le plan "ChercherBois" consiste à se déplacer vers les patches les plus riches en bois et à transformer cette ressource en objet bois.
— Le plan "RamenerPelleAuVillage" ramène l'agent au village pour y déposer la ou les pelles que l'agent possède. Il retire ensuite un bonus de baies en contre-parti et baisse le cogniton "JaiUnePelle".
— Le plan "ConstruirePelle" fait construire une pelle à l'agent s'il possède au moins NombreDeBoisPourFaireUnePelle unités de bois dans son inventaire. S'il dispose d'une pelle, le cogniton "JaiUnePelle" est augmenté.
— Si l'agent possède une hutte, le cogniton "PasBesoinHutte" est ajouté sinon une hutte est construite sur un emplacement libre.

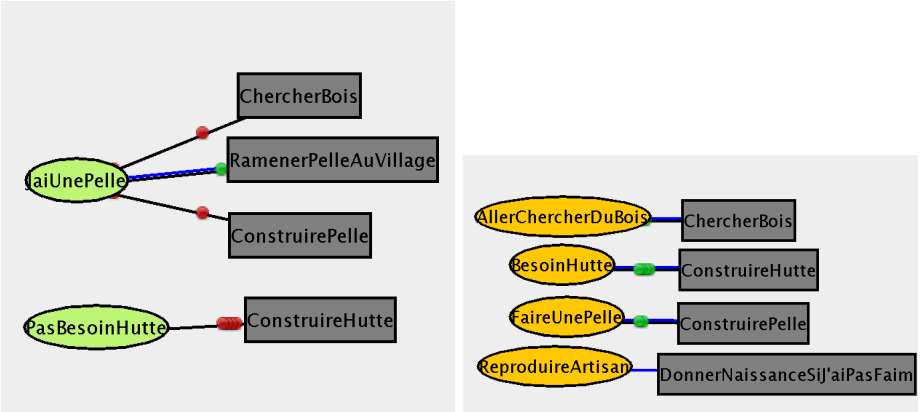

FIGURE 24 – Représentation des plans, culturons et cognitons du rôle artisan dans MetaCiv.

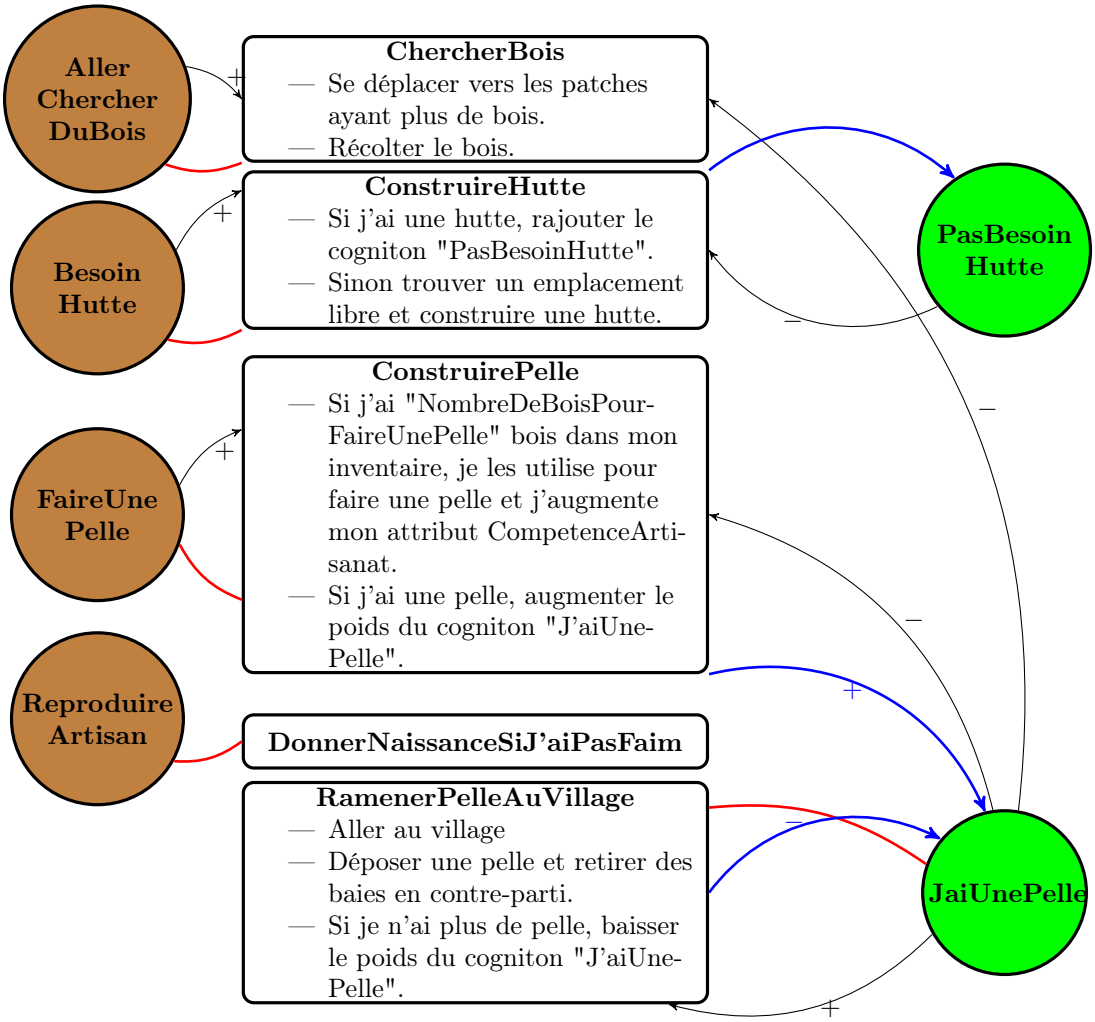

FIGURE 25 – Figure des plans d'un artisan et des interactions avec les cognitons associés.

Les agriculteurs

La seule fonction des agriculteurs est de récolter des baies dans leurs champs ou de construire un champs s'ils n'en possèdent pas. Ils ont donc un seul plan : Agriculter. Si l'agent n'a pas d'aménagement "Champs", il choisi un emplacement libre pour en construire un. Sinon, il se positionne sur son champ pour récolter "NombreDeBaiesRecolteesParTick" baies s'il n'a pas de pelles et "NombreDeBaiesSupplementairesPelles" baies s'il en a une. Lorsque l'agent possède plus de "SeuilRamenerBaiesAgriculteur" baies dans son inventaire, le poids du cogniton "TropDansMonSac" est augmenté.

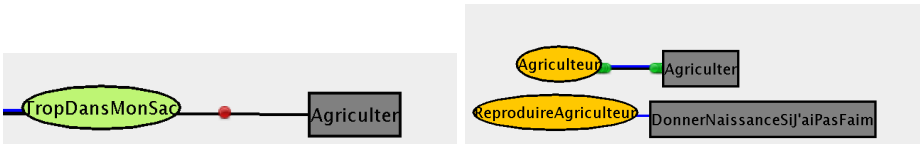

FIGURE 26 – Représentation des plan, culturons et cognitons du rôle agriculteur dans MetaCiv.

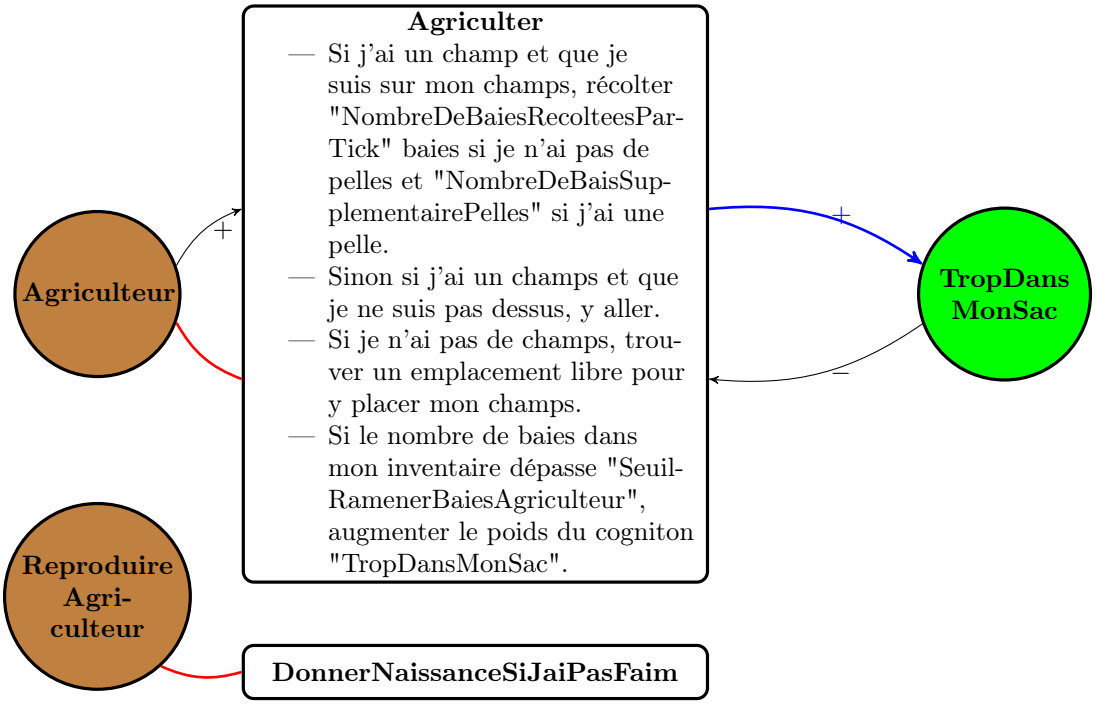

FIGURE 27 – Figure des plans d'un agriculteur et des interactions avec les cognitons associés.

2.2.5 Les aménagement

Les aménagements sont des marques déposées sur des patches et sont repérés par leurs positions. Ils ont un inventaire qui permet aux agents de déposer ou retirer des objets à l'intérieur. Le tableau suivant récapitule les aménagements de la modélisation.

Aménament	Apparence	Types d'agents qui utilisent l'aménagement	Utilisation de l'inventaire
Village		Tout types d'agents	Oui
Hutte		Artisans	Non
Champs		Agriculteurs	Non

FIGURE 28 – Tableau récapitulatif des aménagements.

2.2.6 Les étapes de la modélisation

Au début de la modélisation, les agents sont dispersés autour du point d'apparition. Ils sont immobiles puisqu'ils ne possèdent aucun plan permettant le mouvement (on peut facilement l'ajouter).

FIGURE 29 – Situation initiale

La création des groupes et des villages

Puis, grâce à leurs plans "Standard", certains agents fondent leur groupe en partant à la recherche d'autres agents. Bien sûr, lorsque les agents détectent qu'ils n'ont pas de village, ils en construisent un.

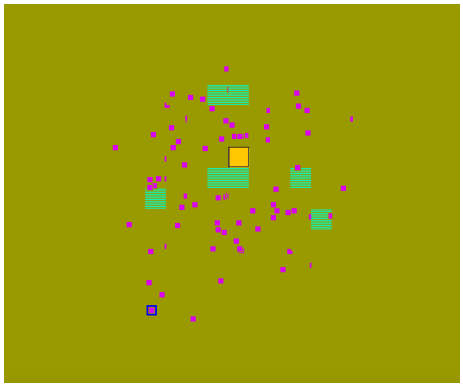

FIGURE 30 – Fondement des groupes et création des villages

La cueillette

Les cueilleurs ayant construit leur village, ils partent alors à la recherche de ressources dans les zones avoisinantes pour remplir le village en ressources. Ceux n'ayant pas réussi à en trouver survivent en se fournissant au village.

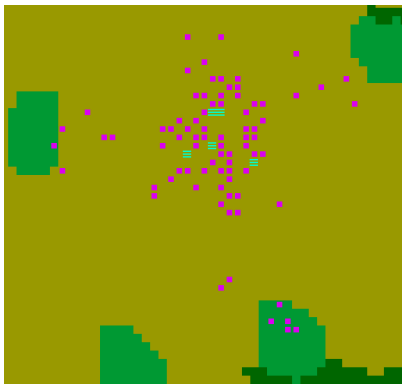

FIGURE 31 – La recherche de nourriture

2.2.7 L'apparition de l'agriculture

Lorsque des cueilleurs ramènent des ressources au village, leurs attributs "CompetenceCueilleur" augmente et à partir d'un certain seuil, ils deviennent des agriculteurs. Leur rôle se réduit alors à récolter des baies dans leurs champs, puis les ramener au village.

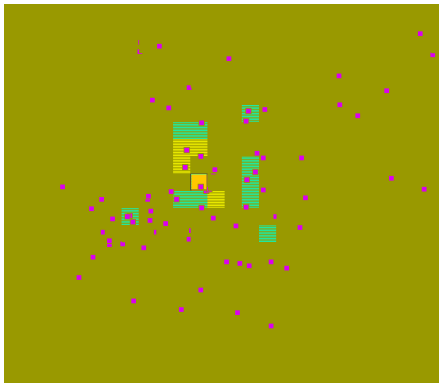

FIGURE 32 – Apparition des premiers champs et agriculteurs

L'apparition des artisans

Certains *Cueilleurs* ont la possibilité de se convertir en *Artisan*. Leur rôle est de récolter du bois et de revenir à leurs huttes pour produire des outils, ici, des pelles. Ils ramènent ensuite ces pelles au villages pour que les agriculteurs puissent s'en servir.

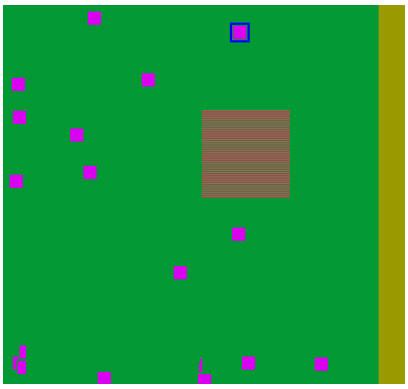

FIGURE 33 – Un artisan (entouré en bleu) proche de sa hutte.

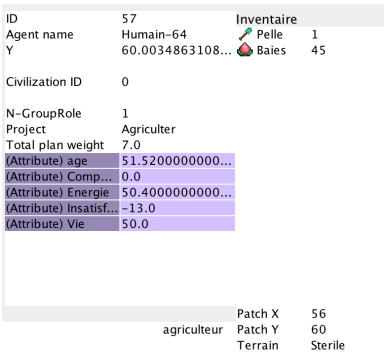

FIGURE 34 – Exemple d'un agriculteur ayant récupéré une pelle au village.

2.2.8 L'évolution de la population selon les âges.

L'évolution de la population avec la constante "ChancesAvoirDesEnfants" égale à 0.5

Dans cette partie, la reproduction des agents ne produit que des cueilleurs. On effectue une première simulation qui ne contient que des cueilleurs.

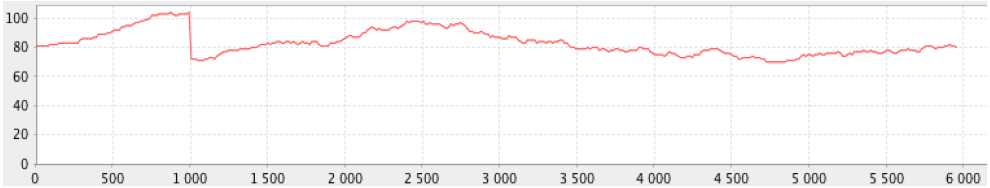

FIGURE 35 – Graphe représentant l'évolution de la population en fonction du temps dans une civilisation ne contenant que des cueilleurs.

On rajoute ensuite la possibilité que certains cueilleurs deviennent des artisans.

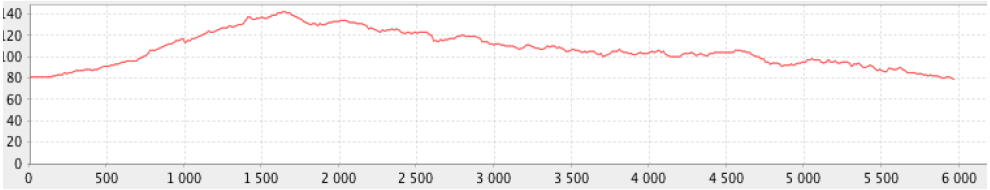

FIGURE 36 – Graphe représentant l'évolution de la population en fonction du temps dans une civilisation ne contenant que des cueilleurs et des artisans.

Puis on rajoute la possibilité qu'un cueilleur devienne un agriculteur.

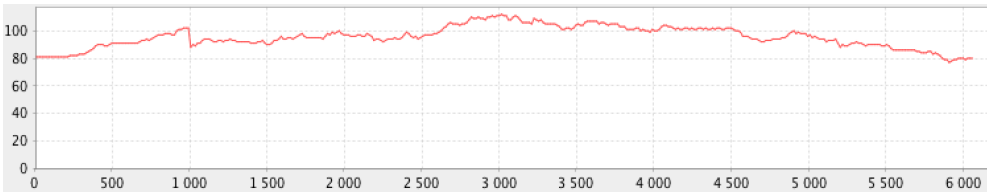

FIGURE 37 – Graphe représentant l'évolution de la population en fonction du temps dans une civilisation contenant des cueilleurs, artisans et agriculteurs.

Après observation de plusieurs graphes, on conclu que les graphes précédents ne donnent pas de résultats significatifs car la population est trop peu nombreuse et le taux de reproduction trop faible. Une modification de la constante "ChancesAvoirDesEnfants" pourrait potentiellement résoudre ce problème.

L'évolution de la population avec la constante "ChancesAvoirDesEnfants" égale à 1.

Dans cette partie, la reproduction des agents ne produit que des cueilleurs et la constante "ChancesAvoirDesEnfants" a été fixé à 1. On remarque après plusieurs lancements que cette modification de constante permet d'avoir des graphes plus significatifs et plus homogènes entre chaque simulation. On effectue une première simulation qui ne contient que des cueilleurs.

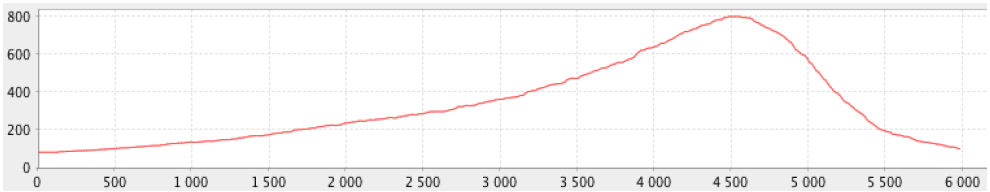

FIGURE 38 – Graphe représentant l'évolution de la population en fonction du temps dans une civilisation ne contenant que des cueilleurs.

La population des cueilleurs augmente considérablement par rapport aux simulations précédentes. Elle croît jusqu'au tick 4500 puis décroît. En effet, la population augmentant dans un environnement pauvre en ressource génère l'apparition du cogniton "Faim" dans l'esprit des agents, les empêchant ainsi de se reproduire ce qui explique la baisse de population.

On rajoute ensuite la possibilité que certains cueilleurs deviennent des artisans.

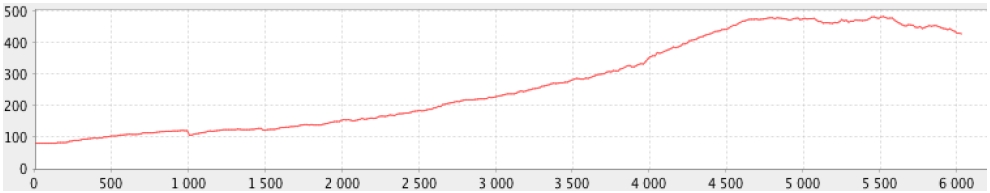

FIGURE 39 – Graphe représentant l'évolution de la population en fonction du temps dans une civilisation ne contenant que des cueilleurs et des artisans.

On remarque qu'au lieu de diminuer à partir du tick 4500, la courbe du nombre d'agents par rapport au tick se stabilise. En fait, les artisans ayant ramenés des pelles au villages constituent une réserve de baies qui leurs permettent de survivre même lorsque la nourriture vient à manquer. Même si les autres agents meurent de famine, les artisans ayant encore des baies peuvent se reproduire et créer d'autres cueilleurs puisqu'ils ne possèdent pas du cogniton "Faim". La mortalité des cueilleurs et l'apparition de nouveaux agents explique la stabilisation de la courbe après le tick 4500.

Puis on rajoute la possibilité qu'un cueilleur devienne un agriculteur.

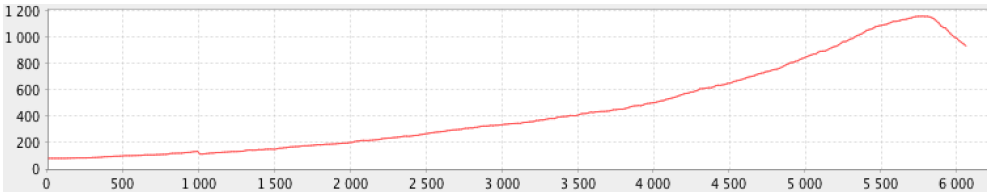

FIGURE 40 – Graphe représentant l'évolution de la population en fonction du temps dans une civilisation contenant des cueilleurs, artisans et agriculteurs.

On remarque que la diminution de la population due à la famine a été déplacée vers le tick 5700. Les agriculteurs apportant de la nourriture même lorsque l'environnement est vide permet à la population de survivre plus longtemps. Cependant, une fois que ces agriculteurs meurent, le nombre d'agriculteur n'est pas renouvelé. En effet, la condition pour qu'un cueilleur devienne un agriculteur est qu'il augmente son attribut "CompetenceCueilleur" et cela est difficile lorsque l'environnement est vide.

Une solution serait alors d'introduire le fait qu'un agent ait une probabilité que ses enfants aient le même rôle que lui, cela règlerait hypothétiquement le problème du manque d'agriculteurs.

L'évolution de la population avec la constante "ChancesAvoirDesEnfants" égale à 1 et l'héritage des rôles.

Dans cette partie, les agents ont une probabilité (ChancesEnfantMemeRole) d'avoir un enfant ayant le même rôle. On effectue une première simulation qui ne comporte que des cueilleurs.

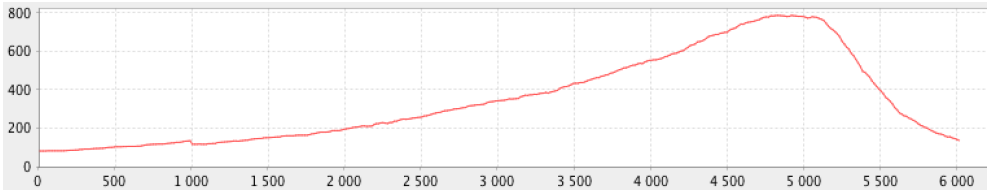

FIGURE 41 – Graphe représentant l'évolution de la population en fonction du temps dans une civilisation ne contenant que des cueilleurs.

On ne remarque aucune différence avec la simulation précédente, ce qui est normal puisque le fait que la reproduction des cueilleurs donne des cueilleurs était déjà présente. On rajoute ensuite la possibilité que certains cueilleurs deviennent des artisans.

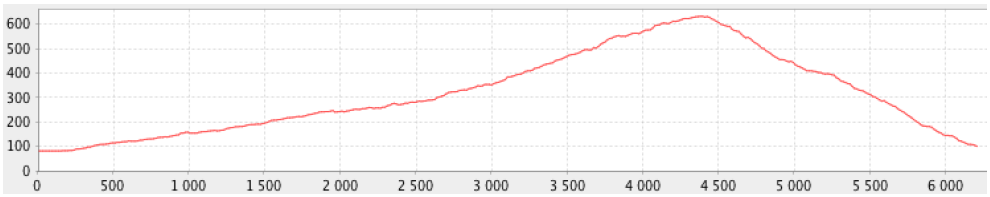

FIGURE 42 – Graphe représentant l'évolution de la population en fonction du temps dans une civilisation ne contenant que des cueilleurs et des artisans.

On remarque que ce qui semblait être un équilibre à partir du tick 4500 n'existe plus. La possibilité qu'un artisan se reproduise et donne un artisan implique une augmentation du nombre d'artisans, ces derniers étant en trop grand nombre n'ont pas de quoi composer leurs réserves de baies. Toute la population est touchée par la famine, même les artisans ce qui explique la baisse générale de population. Il convient donc de rajouter la possibilité qu'un cueilleur devienne un agriculteur.

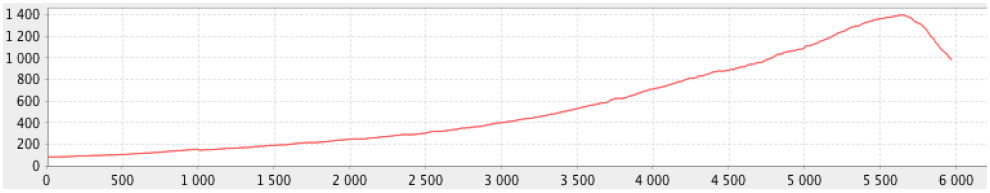

FIGURE 43 – Graphe représentant l'évolution de la population en fonction du temps dans une civilisation contenant des cueilleurs, artisans et agriculteurs

Ici encore, on ne remarque pas de changements majeurs par rapport à la modélisation précédente mise à part une augmentation de la population (1400 au lieu de 1200). En fait, les conditions d'apparition d'un agriculteur étant trop faible, il n'en existe que très peu (trois à quatre) par simulation et leur âges varie généralement entre 30 à 50 pour un attribut "EsperanceDeVie" de 40 à 60. Même avec un pourcentage d'avoir des enfants possédant le même rôle, le nombre d'agriculteur augmente difficilement et la population n'a plus de quoi se nourrir ce qui génère une baisse globale de la population.

2.2.9 Les constantes de la modélisation

SeuilRamenerBaiesCueilleur	50
SeuilRamenerBaiesAgriculteur	50
SeuilNePlusRamenerDePellesAuVillage	0
SeuilNePlusRamenerAuVillage	30
SeuilEnergiePourMourir	0
SeuilConsommationDeBaies	1
SeuilBonCueilleur	5
SeuilFaim	50
NombreEnergieParBaies	8
NombreDeSculpturesPourDevenirUnArtisan	5
NombreDeSculpturesAjoutee	1
NombreDePelleAvantDeRamenerAuVillage	1
NombreDePelleAjoute	1
NombreDeChefParVillage	1
NombreDeBoisParSculpture	1
NombreDeBoisPourFaireUnePelle	5
NombreDeBaiesRecolteesParTick	5
NombreDeBaiesRecolteesAvecPelle	10
NombreDeBaiesDeposeesParTick	10
NombreBaiesRetireesAuVillageSiFaim	6
CompetenceParSculpture	1
CompetenceParPelle	1
ChancesEnfantMemeRole	30
ChancesAvoirDesEnfants	1
BaisseEnergieParTick	-0.1
AgeParTick	0.02

FIGURE 44 – Tableau récapitulatif des constantes de la modélisation.

2.2.10 Conclusion

On constate que le modèle est non seulement cohérent mais permet également d'"Observer" des comportement de groupe. L'émergence de certains individus ayant différents rôles permet à toute la société de survivre plus longtemps. C'est en attribuant de façon permanent un rôle à un individu que l'on est capable d'obtenir une civilisation efficace. Cette modélisation a un niveau de paramétrage élevé qui laisse peu de place au hasard. En effet, on constate que puisque les agents sont enfermés dans leurs plans, il y a peu d'hésitation dans ce qu'il y a a faire pour que la société fonctionne. Du coup, elle est pratique pour "Observer" un comportement voulu et est très sensible à la modification de ses paramètres.

2.3 Le modèle "Nomade-Sédentaire"

Dans ce modèle, on réalise une émergence de civilisation à travers une société dépendante de son environnement. L'objectif est de montrer qu'une société peut à travers l'observation et l'adaptation à son

environnement survivre, s'adapter et évoluer. Pour ce faire, on a décidé de simuler une société nomade sans chefs et très dépendante de son environnement, on devra réussir a faire naitre le besoin de faire de l'agriculture une fois que la population devient trop élever pour être soutenu par un système nomade. La spécialisation des agents sera basé sur le renforcement de leurs cognitons à travers l'expérience de la réalisation ou de l'échec de leurs plans.

2.3.1 L'environnement

L'environnement dans lequel nos agents se déplacent est un espace fermé de $\mathbb{R}^2$. Il y a en tout 6 types de patches présents : Mer, Prairie, Forêt, Littoral et Montagne. Le modèle étant relativement basique, il n' y a que trois types de ressources disponibles : Les baies, le poisson et le gibier. Le tableau suivant donne le récapitulatif des différents patches et des ressources disponibles.

Patch	Ressources		Valeur initiale		Croissance des ressources		Passabilité
Mer	Aucunes		0		0		Non
Montagne	Aucunes		0		0		Oui
Prairie	Baies	Gibier	10	1	0.2	0	Oui
Forêts	Baies	Gibier	10	2	0.1	0.2	Oui
Littoral	Poisson		10		0.2		Oui

FIGURE 45 – Tableau récapitulatif des patches et des ressources.

Il y a trois type de patch qui sont capable d'offrir des ressources. Mais chaque type de patch produit une ressource en priorité. Ce qui forcera les agents a optimiser leur méthodes de récolte en fonction de leur environnement.Un patch ne peut être récolter seulement si les pheromone de la ressource pressente sur le patch sont supérieur a 1, si c'est le cas alors cette pheromone est transformer en un objet. Les baies en baies, les poissons en poissons et les gibiers en viandes.

FIGURE 46 – environnement de la modélisation Nomade-Sédentaire

Le point d'apparition des agents se situe au milieu d'une foret. Les agents pour optimiser leur récolte de ressources devront se mettre très rapidement a la chasse.

On peut aussi noter que lorsque les agents apparaissent dans l'environnement, ils sont éparpillés autour du point d'apparition et non dessus.

2.3.2 Les attributs

Il y a pour l'instant six attributs dans ce modèle : Faim, Age, Charisme, Santé.

— L'attribut Faim est baissé chaque tour d'une certaine quantité ($BaisseFaimParTick$), et n'est remonté qu'en mangeant de la nourriture (Baies ou Viande). Cet attribut va nous servir a simuler la faim chez nos agents.

— L'âge d'un individu est augmenté à chaque tick ($AgeParTick$). Dans cette simulation l'age d'un agent est d'environ 1000 tick.

— L'attribut "Charisme" comme son nom l'indique reflète le charisme d'un individu. Certain humain ayant naturellement une intelligence social et un charisme naturel cela nous servira lors des phase migratoire de nos agents. Cet attribut sera initialise dans le "Birth Plan".

— L'attribut "Santé" reflète l'état physique d'un individu. Lorsque un agent est blesse, affame ou bien vieux l'attribut "Santé" sera diminuer et sera une composante importante de déterminer la vie ou la mort d'un agent.

Attribut	Valeur par défaut
Faim	300
Santé	20
Age	0
Charisme	Valeur initialisé par le "BirthPlan" entre 0 et 5.

FIGURE 47 – Tableau récapitulatif des attributs d'un agent et de leurs valeurs par défauts.

2.3.3 Les plans par défaut

Lorsque les agents arrivent dans l'environnement, ils possèdent plusieurs actions par défauts. Il y en a quatre : Birth Plan, Auto Plan, Cueillir et Chasser, Ramener Ressource ainsi que Manger.

- Le plan "Birthplan" est le plan qui est lance a chaque fois qu'un agent est cree. Dans cette simulation le Birthplan va surtout servir a initialiser les attribut qui son obtenu a la naissance tel que le Charisme.
- Le plan "Auto Plan" est effectuée par chaque agent à chaque tick et gère tout ce qui relève de la physiologie de l'agent. Il modifie donc les attributs : age et faim. Il va aussi servir a réguler la population en décidant a chaque tick si un agent doit mourir ou enfanter un nouvelle agent. L'Auto Plan va aussi être charger de la perception de l'agent et détecter dans quel type d'environnement l'agent se trouve.
- Le plan Cueillir va permettre au agent de récolter des baies.
- Le plan Chasser va permettre au agent de chasser du gibier pour de la viande.
- Le plan Ramener Ressource va permettre au agent de rapporter les ressources qu'ils ont sur eux a l'aménagement le plus proche.
- Le plan Manger est le plan qui va augmenter l'attribut de faim lorsque c'elle ci est trop basse. Pour manger l'agent mangera une ressources de type nourriture qu'il possédé. S'il n'en possédé il ira en chercher a l'aménagement le plus proche.

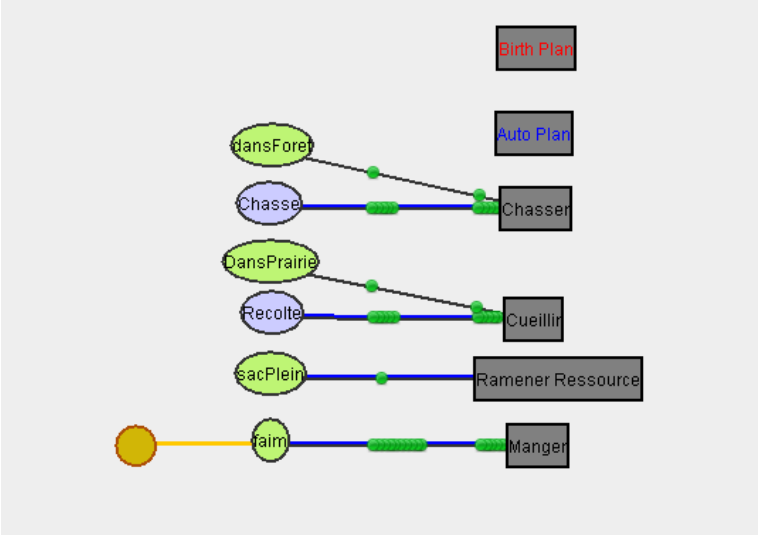

FIGURE 48 – Représentation des cognitons et plans par default associés dans MetaCiv.

On remarquera que les plan "Birthplan" et "Standard" n'ont pas besoin de cognitons pour être effectués. Les plan "Cueillir",Chasser, Ramener Ressource sont accessible dès le lancement de la simulation puisqu'ils sont activé par les cognitons "Chasse", "Recolte" et "SacPlein".

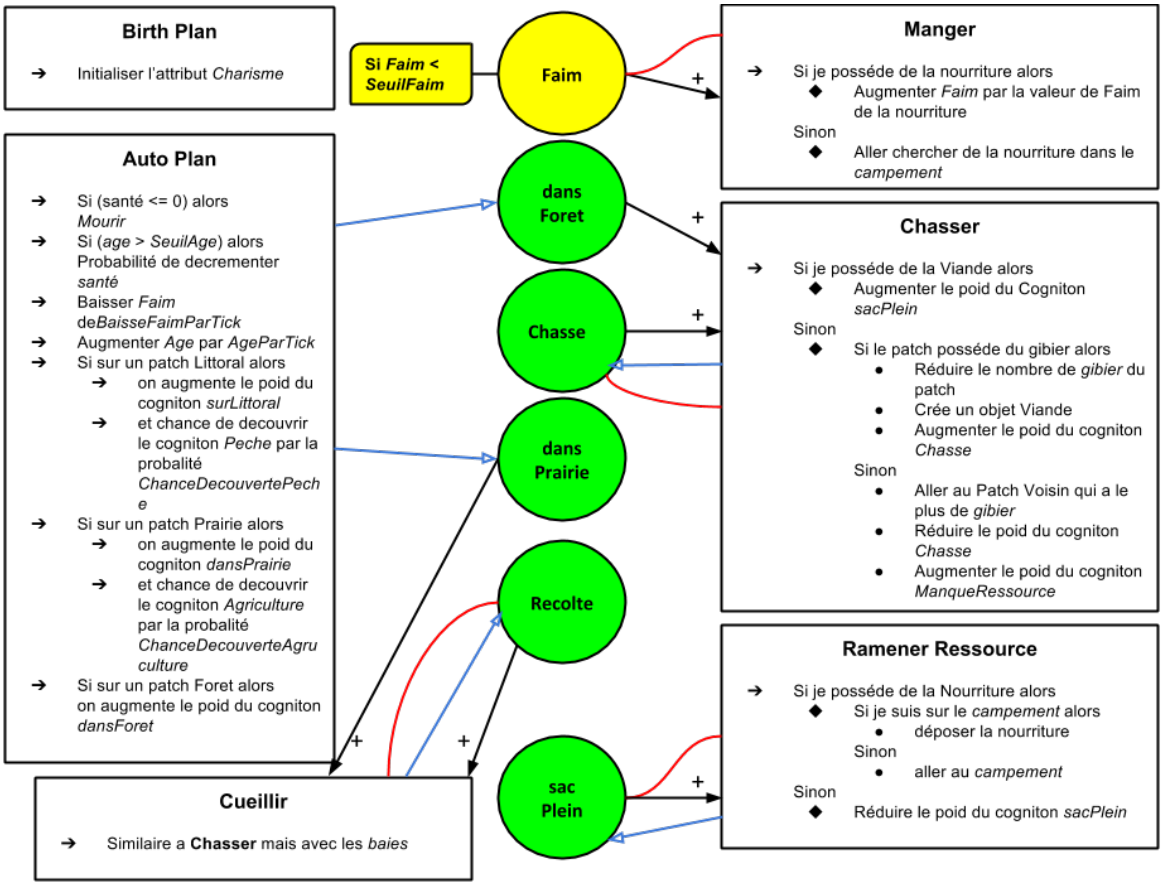

FIGURE 49 – Schéma des cognitons et plans présent chez un agent au lancement de la simulation.

On remarquera cette foi que le cogniton "Faim" a un trigger qui permettra d'enclencher l'action lorsque l'agent en a vraiment besoin.

En plus de ces Plan, nous avons aussi plusieurs plan qui permettent de gérer l'émergence de nouvelle compétence et la migration de la communauté lorsque les ressources autour de leur campement commence a manquer.

— Le plan "Pêcher" va permettre au agent de pêcher des poissons qui est elle aussi une ressource que les agents peuvent utiliser pour se nourrir.
— Le plan "Agriculture" va permettre au agent de faire de l'agriculture. Crée des fermes ainsi que de récolter les ressources produite par ces fermes.
— Le plan "Déménager" qui va permettre au agent de lancer une élection pour savoir si la communauté est favorable pour déplacer le campement. Si ce vote est positif alors un groupe de nomade est crée. Nous présenterons comment le vote est effectuer lors de la présentation du groupe.

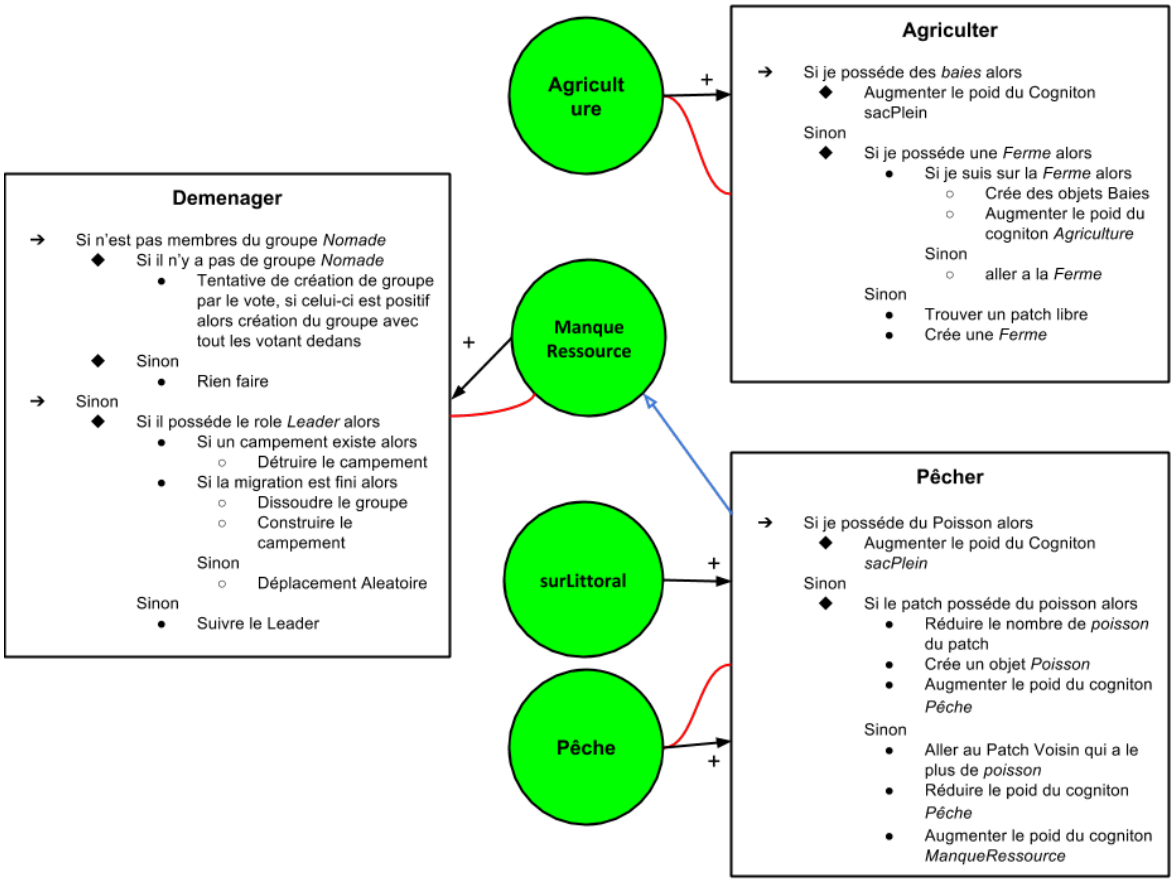

FIGURE 50 – Schéma des cognitons et plans possible autre que les default chez un agent.

On remarquera que le plan "Pêcher" et "Agriculture" sont bien relier a un cogniton pourtant ont ne les a pas nommer lors des plan originaux. En effet ces plan vont être découvert. Cette découverte s'effectue dans le "Auto Plan", lorsque un agent se déplace sur un littoral il a une chance de découvrir la compétence permet "peche" activant son cogniton du même nom. A la découverte de ce Cogniton le plan est activer et utilisable par l'agent. Il en va de même pour "Agriculture" lorsque un agent se trouve sur un patch de type prairie.

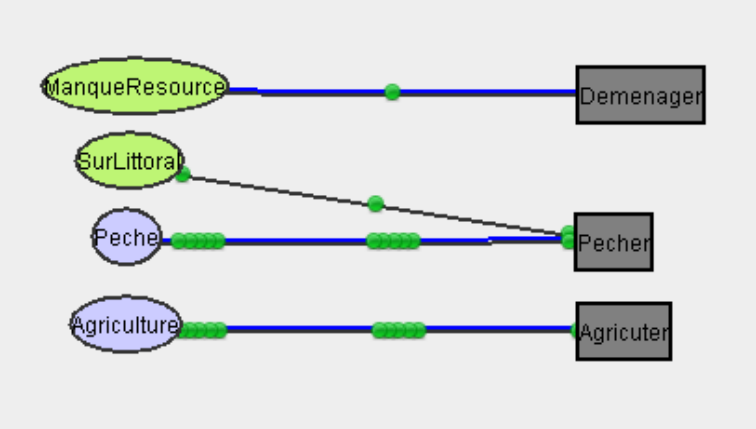

FIGURE 51 – Représentation des cognitons et plans associés dans MetaCiv pour la Pêche, agriculture et déménager.

2.3.4 Les groupes - Les Nomades

Lorsque plusieurs agents ont besoin de réaliser une taches a plusieurs ou bien ont besoin d'obtenir un rôle spécifique dans la communauté ont peut utiliser l'outil de groupe. Dans le cas de cette modélisation nous avons voulu limiter le plus possible l'utilisation de ces groupes pour "Observer" comment les agents aller se spécialiser eux même en fonction de leur environnement. Il y a tout de même une action de groupe qui nécessite cette notion. Lorsque le groupe décide qu'il faut déplacer le campement.

"Leader"

Dans les societes nomade il a exister beaucoups de modele d'organisation different par le passer et aujourd'hui encore ou 1,4% de la population mondiale serait nomade.

Dans le cas de cette modelisation nous avons decider d'avoir une organisation diriger vers une societer sans veritable chef mais plutot des individue qui se demarque en proposant des initiatives et en fonction de leur qualite (charisme, competence) les autres membres de la tribu vont decider de les suivres ou pas.

On a vu que lorsque un agent enclanche le plan demenager il va y avoir un vote. Celui-ci est la pour decider si la tribu deplace le campement ou pas. Pour ce faire le vote se deroule comme suit :

- L'agent demande a tout les agents situer dans un radius autour de sa position si ils veulent le rejoindre ou pas.
- Le calcul pour chaque votant est le suivant :
  - On récupéré le cogniton de type "compétence" le plus élever du votant et on la compare avec ce même cogniton chez le demandeur. Si le cogniton du demandeur est plus élevé alors le bonus du vote est incrémenter de la constante voteBonusValue.

$$\text{Si } highestSkillVoter < skillAsker \text{ alors}$$

$$bonus = bonus + voteBonusValue;$$

— Le vote porte sur une perception de l'environnement, ici du manque de Nourriture. On ajoute donc le poids du cogniton qui est tester, "ManqueResource", a une variable bonusCogniton.

$$bonusCogniton = ManqueResourceWeight;$$

— On ajoute la valeur d'un attribue au bonus, ici dans cette modélisation se sera le Charisme du demandeur.

$$bonus = bonus + CharismeValue;$$

— On lance un des entre 0 et 15.

— On compare le jet de des a l'addition des deux deux bonus. Si celui ci est plus petit on comptabilise un vote positif.

$$\text{Si } dice < (bonus + bonusCogniton) \text{ alors}$$

$$vote + +;$$

— Lorsque chaque agent dans le radius a été interroge on procède au comptage des voix positive. Si elle représente plus de la moitié de la population interroger alors on considère l'élection remporter.

— Le groupe est créé.

Lorsque le groupe est crée, le demandeur du vote devient le leader du groupe obtiens le rôle de "leader". La fonction principale de cet agent est de récupérer les stocks de ressources restante et de guider son groupe vers un nouveaux lieu.

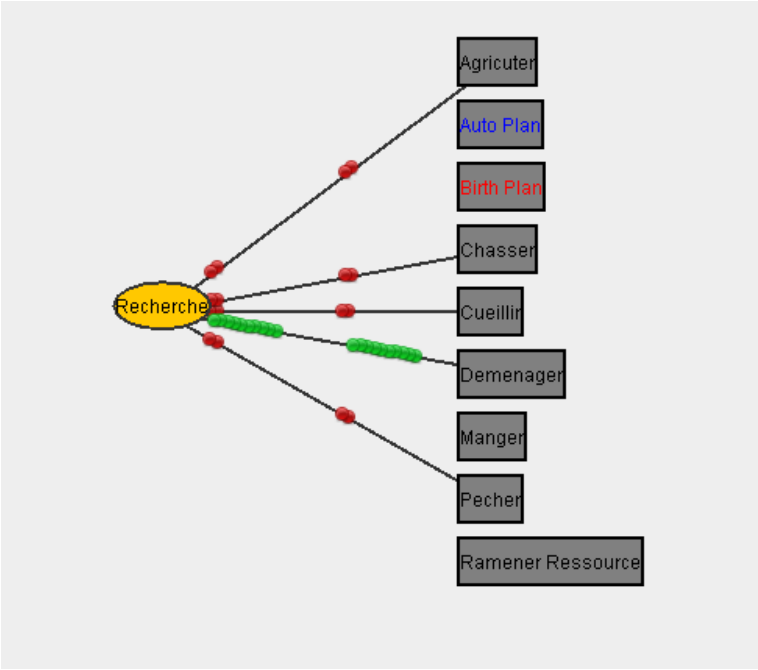

FIGURE 52 – Représentation du culturon Leader et plans associés dans MetaCiv.

Les "Follower"

Lorsque l'élection est réussite les votant rejoignent aussi le groupe, mais en tant que "follower". Les "follower" ont un rôle relativement simpliste. Leur seul objectif étant de suivre le leader et de se maintenir en vie.

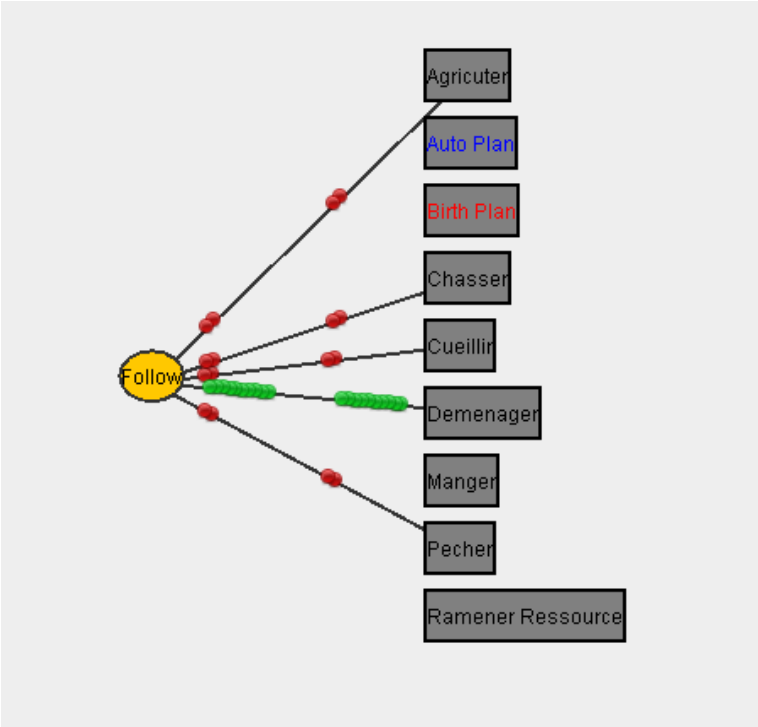

FIGURE 53 – Représentation du culturon Follower et plans associés dans MetaCiv.

2.3.5 Les aménagement

Les aménagements sont des marques déposées sur des patches et sont repérés par leurs positions. Ils ont un inventaire qui permet aux agents de déposer ou retirer des objets à l'intérieur. Le tableau suivant récapitule les aménagements de la modélisation.

Aménament	Apparence	Types d'agents qui utilisent l'aménagement	Utilisation de l'inventaire
Campement		Tout types d'agents	Oui
Ferme		Agriculteurs	Non

FIGURE 54 – Tableau récapitulatif des aménagements de la modélisation Nomade-Sédentaire.

2.3.6 Les étapes de la modélisation

Au début de la modélisation, les agents sont dispersés autour du point d'apparition. Ils sont immobiles puisqu'ils ne possèdent aucun plan permettant le mouvement (on peut facilement l'ajouter).

FIGURE 55 – Situation initiale

La récolte de ressource

Une fois que la communauté a été initialiser, puisque les agents possèdent les compétences nécessaire, ils se mettents dirrectement a faire de la cueillette et de la chasse.

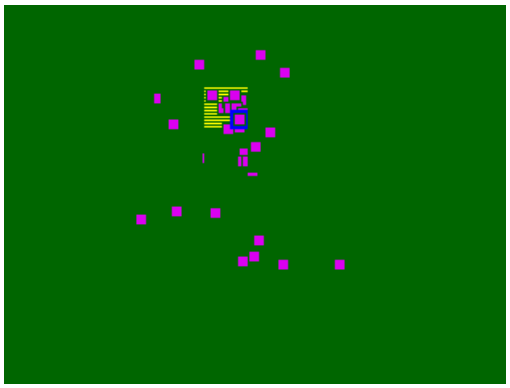

FIGURE 56 – La récolte de ressource

Le manque de ressources, le début de la migration

Les agants ayant de plus en plus de mal a trouver de resources leur perception du manque de resources augmante et par consequent les chances que le vote pour determiner le demenagement du camps soit accepter. Le leader detruit alors le campement en prennant toute les ressources.

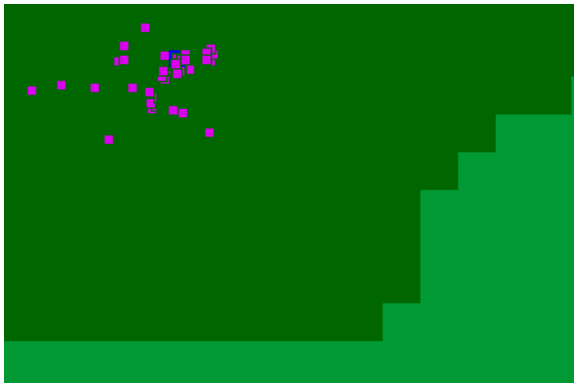

FIGURE 57 – Le début de la migration

La migration

Le leader se déplace aléatoirement sur la carte a la recherche d'un nouvelle endroit ou refaire leur campement. Pendant ce temps les autres agents suivent et font de leur mieux pour survivre.

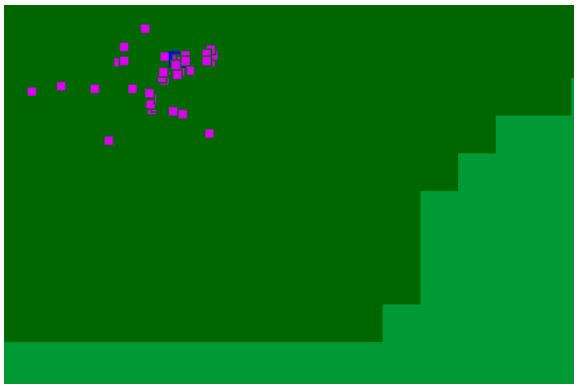

FIGURE 58 – La migration

Fin de migration

Les agents ayant fini leur migration un campement est a nouveau construit. Leur manque de ressource a disparu et recommence a nouveaux leur récolte de ressources pour pouvoir se nourrir et faire suffisamment de stock pour survivre la prochaine migration.

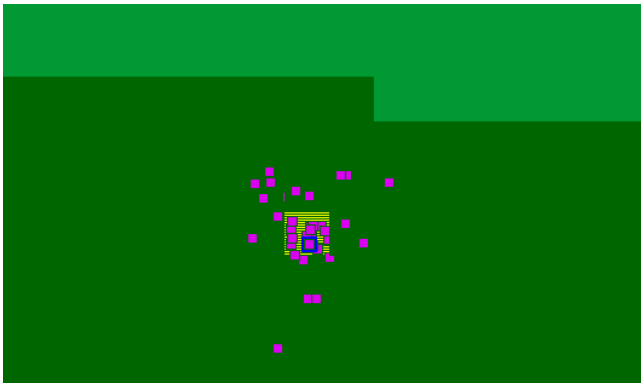

FIGURE 59 – Fin de migration.

Découverte de l'agriculture

Un agent découvre l'agriculture. Son manque personnel de ressource diminue, malgré cela les migrations vont continuer quelque temps. Mais puisqu'il manque moins souvent de nourriture il a plus de chance de se reproduire souvent et par conséquent puisque ses enfants auront la compétence d'agriculture il y aura de plus en plus d'agriculteur. Plus y aura d'agriculteur moins de chance le vote de migration sera accepter et celle-ci arrêteront.

FIGURE 60 – Découverte de l'agriculture.

Debut de village

On observe que la population a exploser mais que les agents étant tous regrouper autour du campement font que toutes ces fermes ressemble a un début de village.

2.3.7 Les constantes de la modélisation

ChanceDecouvertePeche	0.1
ChanceDecouverteAgriculture	0.001
SeuilFaim	50
SeuilAge	1000
voteBonusValue	1
NombreEnergieParBaies	2
NombreEnergieParViandes	15
BaisseFaimParTick	-0.1
AgeParTick	0.02

FIGURE 61 – Tableau récapitulatif des constantes de la modélisation.

2.3.8 Conclusion

Une fois encore, on constate que le modèle est cohérent. Il permet d'observer rapidement des comportements de groupe, mais sa vrai particularité est l'observation de l'adaptabilité de l'agent à son environnement. En effet, le renforcement de cogniton par l'expérimentation des agents d'une façon un peu similaire à ce que l'on peut retrouver dans les colonies de fourmis permet d'aborder l'émergence d'une façon différente. En effet, chaque individu étudie lui même son environnement et c'est le regroupement de l'esprit de chaque individualité qui va créer la transition de civilisation et permettre à la communauté de survivre plus longtemps. Cette modélisation à un niveau de dépendance à son environnement très élevé qui laisse beaucoup de place au hasard. En effet, on constate que puisque les agents font leur choix en fonction de leur environnement, la même civilisation mise dans deux contextes différents auront deux profils d'évolution totalement différentes. Par conséquent, ce genre de modélisation est pratique pour "Observer" l'impact que peut avoir un environnement sur le fonctionnement d'un système multi agent, et dans ce cas là d'une civilisation humaine.

2.4 Observations

3 Le simulateur

3.1 Introduction

MetaCiv est un projet qui a été développé par des groupes successifs d'étudiants. Développé autour de Turtlekit et Madkit, le programme possède une base solide, mais les ajouts successifs de fonctionnalités sans avoir établit de normes et de méthodes au préalable a conduit à un logiciel instable et un code source peu compréhensible.

Nous avons essayé de produire un code plus clair lors de l'ajout de nos fonctionnalités en produisant un code réutilisable avec une approche plus orienté objet et en utilisant quelques design patterns.

3.2 Fonctionnalités implémentés

3.2.1 Le schéma cognitif

Un schéma cognitif est une nouvelle classe qui englobe tout ce qui relève du comportement des agents. L'implémentation d'une telle classe permet d'introduire plusieurs nouvelles fonctionnalités :

- Mettre en concurrence plusieurs civilisations ayant un fonctionnement différent dans le cadre d'un jeu ou d'une évaluation de modèles.
- Modéliser les relations entre ces différentes civilisation, par exemple l'évolution des populations d'animaux sauvages à l'arrivée de l'homme.
- Modéliser différents comportements au sein même d'un modèle tout en conservant un environment défini.

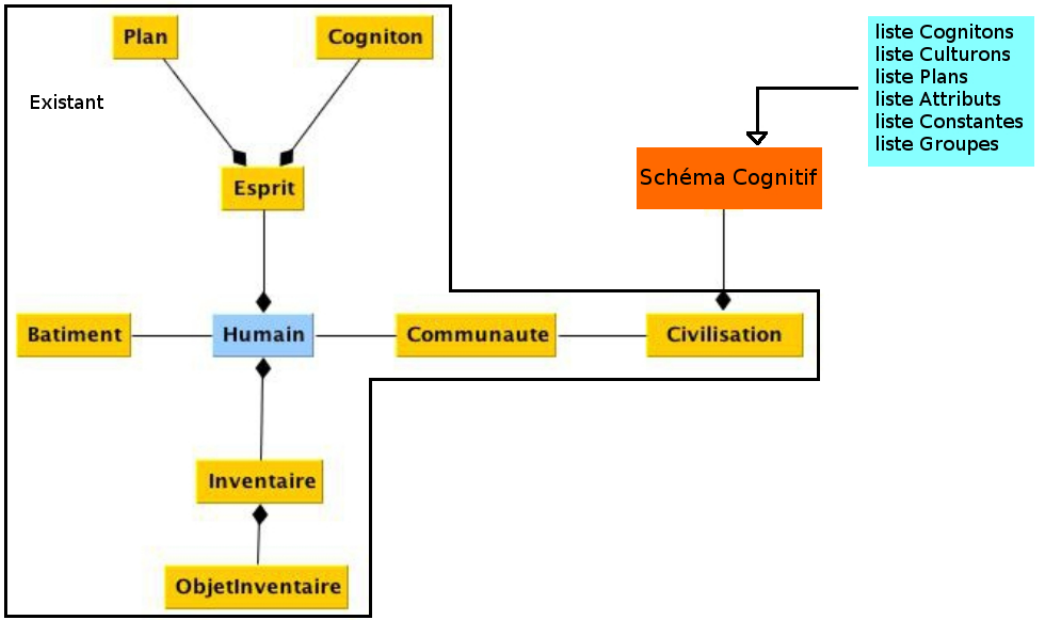

FIGURE 62 – Diagramme simplifié du programme avec l'introduction du schéma cognitif.

Pour implémenter cette architecture de schéma cognitifs, nous avons procédés en trois étapes :

— Nous avons changé le code des accesseurs de la classe "Civilisation" pour qu'ils renvoient les données du schéma cognitif contenu dans la civilisation. C'est à dire que que la classe "Esprit" d'un agent fait toujours appel à la classe "Civilisation" pour s'alimenter en plans et cognitons.

```
public ArrayList<NPlan> getPlans(){
    return cerveau.getPlans();
}
```

FIGURE 63 – Exemple d'un accès à la liste des plans disponibles. "Cerveau" est la référence vers le schéma cognitif

— Nous avons mis au point ce que nous appelons des fabriques pour permettre la gestion de ces schémas. Ces fabriques sont des classes à mi-chemin entre le pattern fabrique abstraite et singleton. Elles génèrent des objets schémas cognitifs et civilisation par des méthodes statiques et les conservent dans une liste statique. Nous pouvons maintenant créer de nouveaux objets, les charger à partir d'un répertoire, les sauvegarder et les cloner en les renommant .

— Nous avons modifié l'interface de MetaCiv pour rajouter les barres d'outils permettant de gérer l'édition des schémas cognitifs.

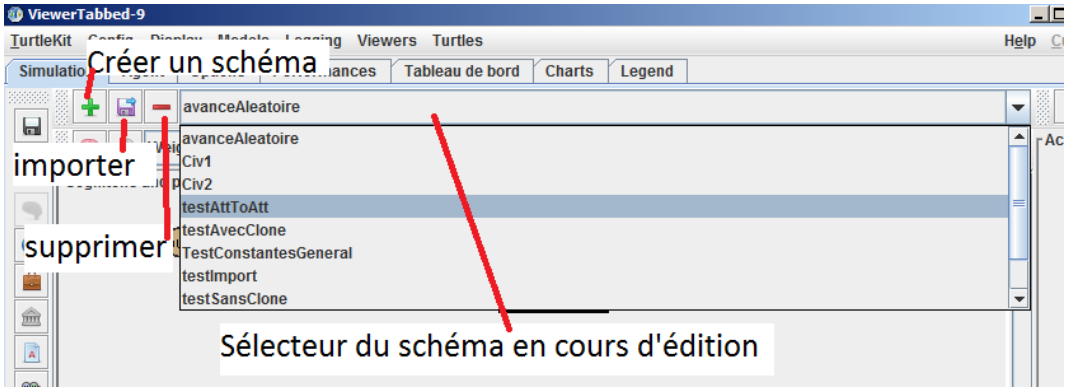

FIGURE 64 – Barre d'outils permettant de sélectionner, créer, importer ou supprimer un schéma cognitif.

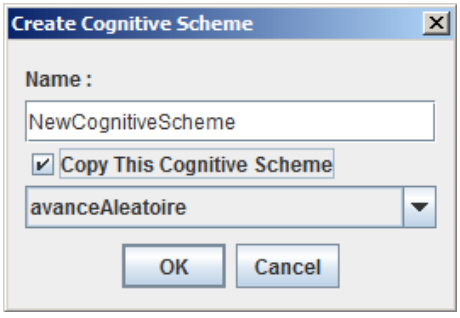

FIGURE 65 – Fenêtre de création d'un nouveau schéma cognitif, possibilité de cloner un schéma existant en le renommant

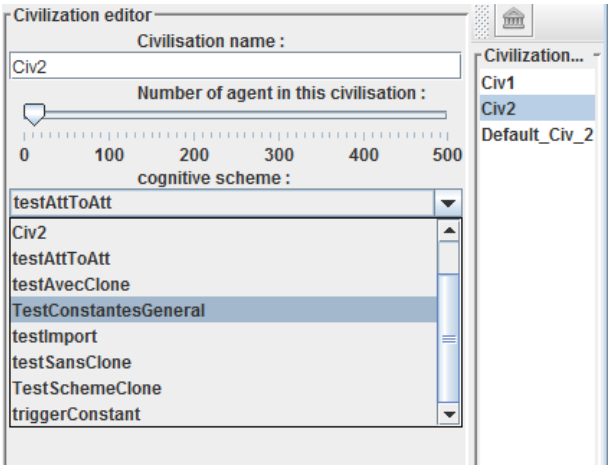

FIGURE 66 – Ajout d'un sélecteur du schéma cognitif à utiliser dans l'interface de gestion des civilisations.

3.2.2 Les constantes

Les constantes sont un ensemble de paramètres numériques de la simulation. Elles peuvent être prise en paramètre d'une ou plusieurs actions mais ne peuvent être modifiés que par l'utilisateur du simulateur. Elles permettent de :

- Centraliser les constantes utilisés à plusieurs endroits dans les plans pour simplifier le travail de réglage du modèle.
- Prendre en compte une mise à jour de la valeur des constantes pendant l'exécution de la simulation.
- Choisir entre l'utilisation des constantes et l'utilisation d'une valeur unique dans une action.
- Assurer la rétro-compatibilité des actions tout en donnant la possibilité de les adapter avec un minimum de changements.

Pour mettre en place les constantes, nous avons introduit le design pattern "Observer". Nous avons crée une classe "MCConstant" qui hérite de "Observable" et qui contient la valeur numérique de la constante, cette classe est manipulée directement par l'interface de gestion des constantes.

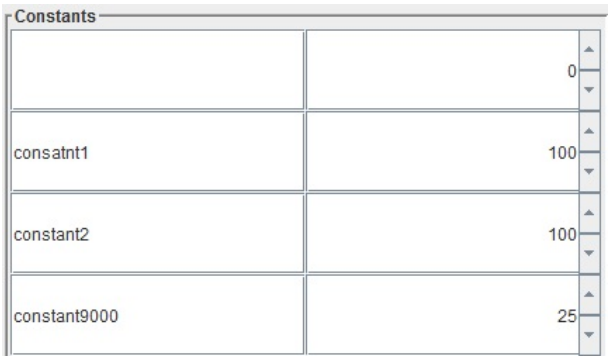

FIGURE 67 – Interface de gestion des constantes, agit directement sur les variables de type MCConstant

Nous avons ensuite crée les classes "MCDoubleParameter" et "MCIntegerParameter" qui implémentent l'interface "Observer". Ces classes sont utilisés dans le code des actions et remplacent les variables double et int. Elles sont initialisés à l'aide de la méthode "loadParam" de la classe "Action", dont toutes les actions héritent.

```
public MCDoubleParameter loadDoubleParam(OptionsActions option)
{
    double val = (Double) option.getParametres().get(0);
    MCConstant Const = null;

    if(option.getParametres().size() > 1
            && option.getParametres().get(1).getClass().equals(String.class)
            && option.testStringSubType("Constant", 1)
            &&sc.getConstantByName((String)option.getParametres().get(1))!=null)
    {
        Const = sc.getConstantByName((String)option.getParametres().get(1));
        val = Const.getValue();
    }

    MCDoubleParameter ret = new MCDoubleParameter(val, Const);
    return ret;
}
```

FIGURE 68 – Méthode d'initialisation. On peut voir l'initialisation de la valeur numérique à la première ligne, puis si une constante à été spécifié, la valeur est changé pour celle de la constante et le lien "Observable-Observer" est mis en place.

Pour ne pas avoir à dupliquer les actions entre celles qui utilisent les constantes et celles qui utilisent les valeurs directes, nous proposons systématiquement l'utilisation d'une constante dans l'interface pour tout les paramètres de type numériques. Si l'action n'a pas été mise à niveau, la constante sera simplement ignorée et la valeur directe sera prise en compte ce qui assure la rétro-compatibilité.

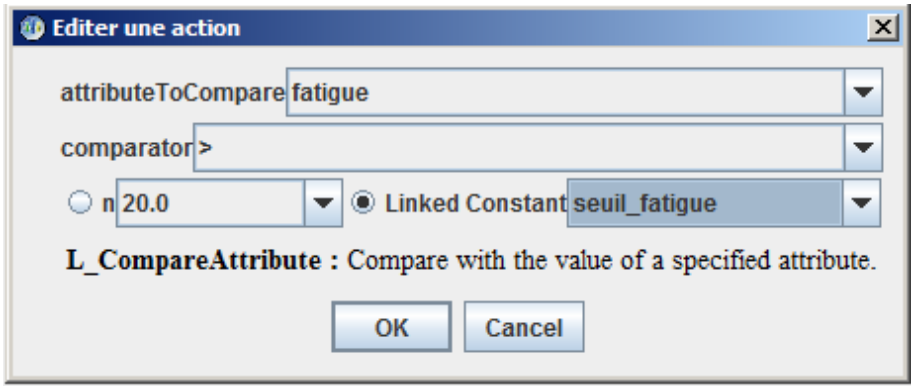

FIGURE 69 – Interface de d'édition d'une action, on peut choisir l'utilisation de la valeur directe ou d'une constante à l'aide du bouton radio, le menu déroulant "linked constant" propose toutes les constantes du schéma cognitif courant.

Nous avons étendu ce principe aux triggers

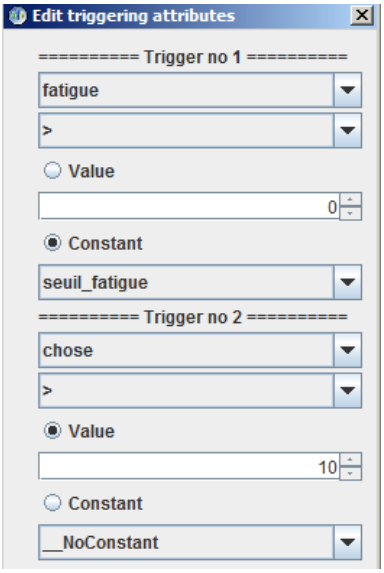

FIGURE 70 – interface de paramètrage des triggers d'un cogniton. le premier utilise une constante , le second une valeur directe

3.2.3 Sauvegarde incrémentale

besoins :

— restaurer une sauvegarde corrompue.

— ne pas solliciter d'actions supplémentaires de la part de l'utilisateur en cas de fonctionnement normal.

— ne pas saturer le disque dur.

Avant d'écraser les fichiers avec la nouvelle sauvegarde , on effectue une copie des fichiers d'origine dans un dossier backup situé a coté du dossier du modèle. Le nombre de sauvegardes ainsi conservées est limité par une constante, la sauvegarde la plus ancienne est remplacée par la nouvelle.

3.2.4 Les defines

Les defines sont un ensemble de classes ("DefinePath" et "DefineConstant") qui permettent de centraliser les valeurs constantes utilisées dans le code du programme afin de faciliter les modifications éventuelles. Les classes defines contiennent des variables statiques qui sont appelées à différents endroits du programme.

```
public class DefinePath {

    // A COMPLETER
    // Penser a supp de Configuration
    public static String pathToRessource = Configuration.pathToRessources;

    /* Schemas Cognitifs */

    public static final String schemasCognitifs = "schemasCognitifs";
    public static final String attributes = "attributes";
    public static final String constants = "constants";
    public static final String cognitons = "cognitons";
    public static final String plans = "plans";
    public static final String groups = "groups";
    public static final String cloudCognitons = "cloudCognitons";

    public static String pathToSchemas = pathToRessource + "/" + schemasCognitifs;
    public static final String pathToAttributes = "/" + attributes;
    public static final String pathToConstants = "/" + constants;
    public static final String pathToCognitons = "/" + cognitons;
    public static final String pathToPlans = "/" + plans;
    public static final String pathToCloudCognitons = "/" + cloudCognitons;
    public static final String pathToGroups = "/" + groups;

    /* Simulation */
    public static final String simulation = "simulation";
    public static final String actions = "actions/civilisation/individu/plan/action";
    public static final String amenagements = "amenagements";
    public static final String civilisations = "civilisations";
    public static final String effects = "effects";
    public static final String environnements = "environnements";
```

FIGURE 71 – Capture partielle de la classe DefinePath qui liste les chemins d'accès aux ressources

3.2.5 Autres fonctionnalités implémentées

— Permettre la suppression d'éléments des cognitons, culturons, plans.
— Ajout d'un bouton qui permet une fermeture propre du programme (sans laisser de "processus zombie").
— Barres de scroll pour les listes d'éléments (attributs, constantes, actions d'un plan...).

3.3 Fonctionnalités envisagés

3.3.1 Init-plan

Un "Init-plan" serait un plan qui s'exécuterait au premiers instants de la simulation, pour chaque agent. Une des applications serait de modifier l'âge de départ des agents de manière aléatoire pour représenter le fait que la population de départ a déjà vécu un certain temps et est composée d'individus d'ages différents.

3.3.2 Hiérarchie de cognitons

La hiérarchie de cognitons est un nouveau système de prise de décision qui se base sur l'ancien système mais rajoute la notion de MetaPlan. Un MetaPlan est un plan qui conduit à l'évaluation d'un nouveau groupe de cognitons. A chaque fois qu'un agent cherchera à prendre une décision, un MetaPlan spécial noté "racine" sera lancé.

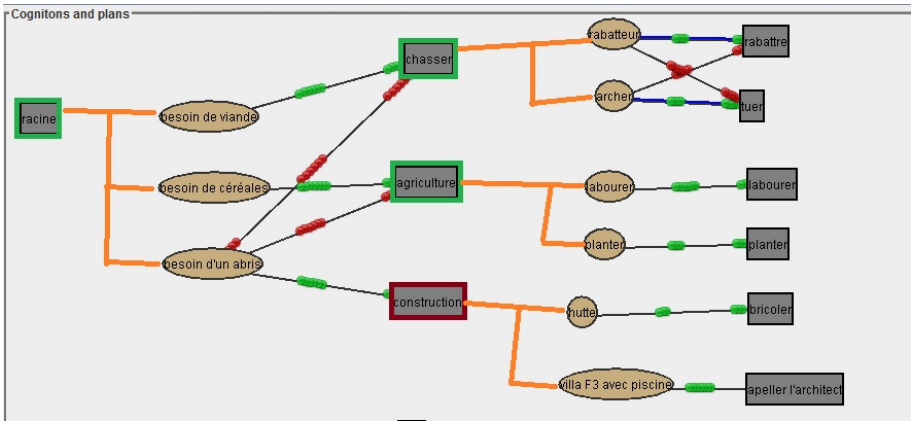

FIGURE 72 – Application à une répartition en sous rôles : si le plan spécial "chasser" est décidé, l'agent n'évaluera plus que les cognitons "rabatteur" et "archer".

Dans l'exemple précédent, on remarque que les cognitons et les plans peuvent se représenter sous forme d'un arbre. Les plans sont les feuilles, le MetaPlan "racine" est la racine de l'arbre et les cognitons et Metaplans sont les noeuds.

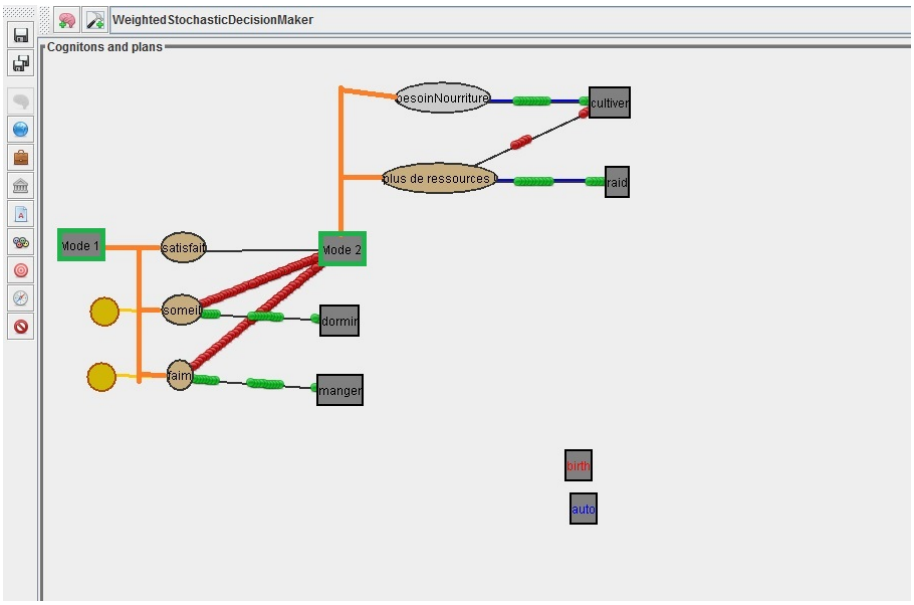

FIGURE 73 – Application système de subsomption : les besoins essentiels sont évalués à la première étape comme la faim ou le sommeil, si ces besoins sont satisfaits, le cogniton satisfait prends le dessus et déclenche le MetaPlan "Mode 2" qui évalue des actions plus complexes

3.3.3 Objets uniques et empilables

Il serait intéressant de séparer les objets en deux catégories :
— Les objets uniques qui possèdent des attributs comme l'usure de l'objet, l'attaque, la défense, etc..
— Les objets dits "stackables" (litt. "empilable") qui resteront toujours identiques, par exemple les ressources alimentaires.

On remarquera que pour l'instant tous les objets sont pour l'instant "stackables" et possèdent un attribut "qualité".

3.4 Stabilisation du noyau

Dans cette partie, nous vous présenterons quelques dysfonctionnements, leurs raisons et les corrections effectuées.

Problème	Gestion des plans : au chargement, certains plans sont ajoutés deux fois (mais la simulation fonctionne quand même)
Raison	Un test toujours vrai dans le chainage des plans et cause l'ajout d'un double du plan.
Correction	Nettoyage du code, le problème apparait clairement et est résolu (note : amélioration sensible des performances

FIGURE 74 – Exemple de correctif sur la gestion des plans.

Problème	Gestion des triggers : la modification des triggers ne semble pas avoir d'effet.
Raison	Présence de deux listes représentant les triggers : une est utilisé par les agents, l'autre par l'interface.
Correction	Regroupement des deux listes en une seule, modification de la liste au lieu et reconstruction (permet la modification des valeurs pendant l'exécution)

FIGURE 75 – Exemple de correctif sur la gestion des triggers.

Problème	Esprit de l'agent : certain cognitons sont présents en double, ce qui fausse le calcul de l'action à effectuer.
Raison	Les starting cognitons sont ajoutés en plus de la liste de tous les cognitons qui les contient déjà.
Correction	Sécurisation de la liste des cognitons d'un agent à travers un accesseur, si le cogniton est déjà présent, l'ajout est ignoré.

FIGURE 76 – Exemple de correctif sur l'Esprit des agents.

Problème	Sauvegarde : la sauvegarde sous windows corrompt certain modèles (pas sur mac).
Raison	Pas de vérification des caractères interdits dans le nom des éléments qui sont utilisés pour nommer le fichier correspondant.
Correction	Utilisation de "URLencoder" sur toutes les chaines servant comme nom de fichier.

FIGURE 77 – Exemple de correctif sur la gestion des sauvegardes.

Au total , c'est plusieurs dizaines de bugs qui ont été corrigés.

3.5 Conclusion

MetaCiv est un logiciel puissant qui repose sur de bonnes bases. Par exemple, le système d'injection d'actions personnalisé sous forme de code java est très bien conçu et le moteur du simulateur est fiable. Cela encourage à l'enrichissement du projet à de nouvelles fonctionnalités pour l'avenir. En revanche, le code des interfaces est très désorganisé dû aux ajouts des différents groupes qui se sont succédés (dont le notre) et devrait être repris avec de bonnes méthodes. Du point de vue de l'utilisation, l'interface manque de sécurité et de retour utilisateur en cas de commande invalide, ce qui causait bien souvent la corruption du modèle sans que l'on comprenne pourquoi. En effet, régler ce problème de retour d'information à l'utilisateur est une étape très importante pour pouvoir amener MetaCiv à un plus grand public.

4 Gestion de projet

La gestion de projet a joué un rôle important dans le développement des nouvelles technologies au cours du 20^{ieme} et 21^{ieme} siècle. Si important qu'au vu du très grand nombre d'échecs de projet que dans les années 1990, il fût décidé de rechercher des méthodes différentes aux méthodes traditionnelles, comme la méthode dite de "Waterfall", pour gérer des projets informatiques. C'est la naissance de la méthode Agile et de son manifeste qui décrit les douze principes et quatre valeurs a respecter.

Nous commencerons par introduire la méthode Agile pour ensuite décrire le déroulement du projet et ce que nous avons fait pour respecter le plus possible les bonnes pratique demander. Finalement, nous expliquerons les outils que nous avons utilisé pour faciliter la gestion de ce projet et donnerons nos conclusions quand a la réussite ou non de notre gestion.

4.1 La méthode Agile

La méthode agile repose sur des cycles de développements itératifs et adaptatifs en fonction des besoins évolutifs du client. Elle permet notamment d'impliquer l'ensemble des membres du groupe ainsi que les responsables du sujet dans le développement du projet. Cette méthode permet généralement de mieux répondre aux attentes du client en un temps limité (en partie grâce à l'implication de ce dernier) tout en permettant au groupe d'acquérir de nouvelles compétences. Cette méthode constitue donc un gain en productivité pour les deux côtés, ainsi qu'une flexibilité accrue.

Pour ce projet nous avons opté pour un sprint entre chaque réunion de groupe avec les tuteurs. MétaCiv étant déjà existant, dès les premières réunions nous avons pu présenter des modélisations, pointer du doigt certains problèmes, s'organiser pour les résoudre et redéfinir les objectifs soulevés.

4.2 Déroulement du projet

Le sujet de ce projet portait sur la modélisation de société humaine à travers l'utilisation de systèmes multi-agent et tout particulièrement l'utilisation du logiciel MetaCiv. A cela venait s'ajouter l'obligation de devoir collaborer avec une autre équipe travaillant aussi sur l'évolution de MetaCiv.

La gestion de plusieurs équipes pouvant vite devenir très difficile, nous avons décidé avec la participation de Monsieur Ferber d'adopter l'utilisation de la méthode Agile. Cette méthode a été très efficace et nous a permis grâce a des sprints d'une à deux semaines de nous permettre de montrer les avancés du projet et de discuter de ses besoins. Lors de nos réunions hebdomadaires ou bi-mensuel , le groupe de modélisation présentait son avancé et les problèmes rencontrés puis avec les conseils de Mr Ferber et Mr Stratulat , nous mettions au point des solutions qui permettaient de pousser la modélisation plus loin.

Nos premières expériences de modélisation se soldaient souvent par des modèles corrompus et donc à des heures de travail perdus. Nous avons donc été amené à la décision d'utiliser une partie de nos ressources afin de rendre le programme plus stable. Dans un premier temps, les quatre membres devaient participer au développement des modèles de civilisation, mais le projet a évolué dû des contraintes logiciels. François Suro est devenu responsable de la maintenance de MetaCiv et de l'ajout de nouvelles fonctionnalités nécessaires à la Modélisation. Pendant ce temps, le reste des membres s'occupaient de l'objectif premier de notre projet, la modélisation. En parallèle, Lionel Ferrand était également chargé de la partie coordination et gestion de notre logiciel et plateforme de versionning GitHub.

Une fois cette étape atteinte, excluant tout risque de perte de travail, nous avons conservé cette séparation des tâches pour continuer la correction des nombreuses instabilités de MetaCiv. Avec un besoin croissant d'évolution dans le logiciel, il nous fallait donc un outil pour facilement faire le relai entre les équipes et avoir un suivis efficace des tâches à accomplir. Nous avons décidé d'utiliser Trello, un logiciel de Post-It en ligne que nous expliquerons plus en detail plus tard.

Les ambitions du projet ont donc été revu à la baisse tout en maximisant le travail qui était faisable avec les moyens et le temps que l'on disposait.

En somme, le développement a été conduit pour les besoins de l'équipe de modélisation. De plus, les avancées du logiciel que nous réalisions servaient également au deuxième groupe. Mais alors, que la partie développement avait prise une place plus importante que prévu nous avons aussi pensé à la pérennité du projet en fournissant un code plus clair, en séparant bien chaque chose et en évitant les fonctions de plusieurs centaines de ligne sans nécessité, maximisant les bonnes pratiques des approches plus orienté objet et en utilisant des design patterns.

4.3 Trello

Pour nous assister dans l'organisation des tâches à faire, nous avons utilisé un outil de gestion de projet en ligne, Trello. Ce programme est basé sur une organisation des projets en planches listant des cartes (tâches à faire, sorte de Post It virtuel). Les cartes sont assignables à des utilisateurs et sont

mobiles d'une planche à l'autre, traduisant leur avancement. Ainsi, nous avons pu organiser le travail à faire en catégorie, pour connaître les tâches restantes et indiquer les fonctionnalités que l'on souhaitait voir, tout en ayant la possibilité de cibler le destinataire du message tout étant visible par tous.

4.4 GitHub

Pour le développement des avancées de MetaCiv, nous avons utilisé GitHub. GitHub est un service web pour les projets de développement de logiciels qui utilisent le système de contrôle de révision Git. Un système de contrôle de révision est la gestion des modifications apportées aux documents, programmes informatiques et autres collections d'informations. Les changements sont généralement identifiés par un code numérique ou alphabétique, appelé le "numéro de révision", "niveau de révision", ou simplement «révision». Par exemple, un ensemble initial de fichiers est "révision 1". Lorsque le premier changement est effectué, l'ensemble résultant est "révision 2", et ainsi de suite. Chaque révision est associée à un horodatage et à la personne qui fait le changement. Les révisions peuvent être comparés, restaurés, et avec certains types de fichiers, fusionnées. Il est un outil obligatoire pour les équipes de développement aujourd'hui.

5 Perspectives d'évolution

MetaCiv permet de modéliser un grand nombre de situation (interactions entres molécules, sociétés, etc...) si l'on est capable de coder les actions nécessaires et la gestion des plans par des cognitons est une alternative originale à la modélisation des modèles par la subsomption. Voici quelques remarques que nous avons eu tout au long de notre modélisation :

- L'ajout des constantes est un atout majeur pour l'utilisateur facilitant ainsi la modification des paramètres de la simulation. On peut ainsi voir en temps réel l'impact d'un paramètre sur la simulation.
- La sauvegarde automatique des dernières versions fonctionnelles d'un modèle permet d'économiser énormément de temps lorsque l'on travaille sur une modélisation. En effet, il n'est pas rare de changer un paramètre, sauvegarder et de ne plus être capable de lancer son modèle, le rendant alors inutilisable.
- L'interface de gestion des agents est un outil agréable à utiliser, on voit directement les paramètres importants de chaque agent : cognitons, plans, attributs, groupe. Il n'est cependant pas assez efficace si l'on veut "Observer" un type d'agent particulier dans une grande population.
- Il serait intéressant de pouvoir sauvegarder la modélisation à un instant donné et ainsi pouvoir la relancer avec différents paramètres. En effet, lorsque l'on modélise une société avec plusieurs niveaux technologiques, on perd énormément de temps à repasser le début du modèle pour "Observer" l'impact d'un changement.
- La construction d'un plan est très intuitive, on imbrique des actions les unes dans les autres. Les descriptions des actions sont claires et précises. Il n'en reste que construire un plan ou le modifier prend beaucoup de temps puisqu'il faut à chaque fois sélectionner l'action dans un menu déroulant. Une autre interface serait à envisager.

MetaCiv possède de grande possibilités d'évolution, nous sommes conscient que l'ajout de nouvelle fonctionnalité ne doit pas rendre le logiciel plus complexe mais le rendre plus simple et facile d'utilisation pour le grand public.

6 Conclusion