

UNIVERSITÉ MONTPELLIER II

SCIENCES ET TECHNIQUES DU LANGUEDOC



UNIVERSITÉ
DE MONTPELLIER

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

22 mai 2015

Introduction

1 Présentation de la plateforme MetaCiv

1.1 Principe

1.2 Concepts

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 *intérieurité* 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 influencer sur ses cognitons. De même les actions peuvent directement ou indirectement via les objets ajouter ou influencer 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.

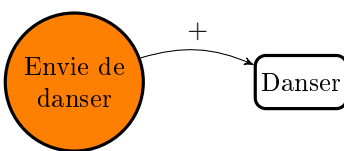


FIGURE 1 – Exemple d'un cogniton renforçant un plan (flèche positive).

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.

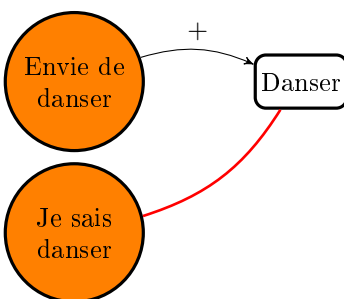


FIGURE 2 – Exemple d'un cogniton renforçant un plan (flèche positive) et d'un deuxième cogniton qui rend le plan accessible (trait rouge).

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.

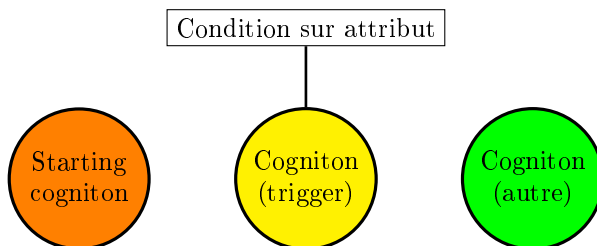


FIGURE 3 – Exemple des différents types de cognitons : Starting cogniton (en orange), cogniton activé par trigger (en jaune) et les cognitons ajoutés par des plans (en vert).

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 *Mêmes* sont des croyances ou comportements culturels transmissibles. 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.

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.

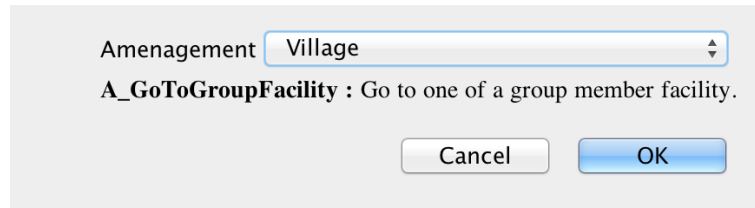


FIGURE 4 – Exemple de l’interface d’édition d’une A_action dans MetaCiv



FIGURE 5 – Exemple de l’interface d’édition d’une L_action dans MetaCiv

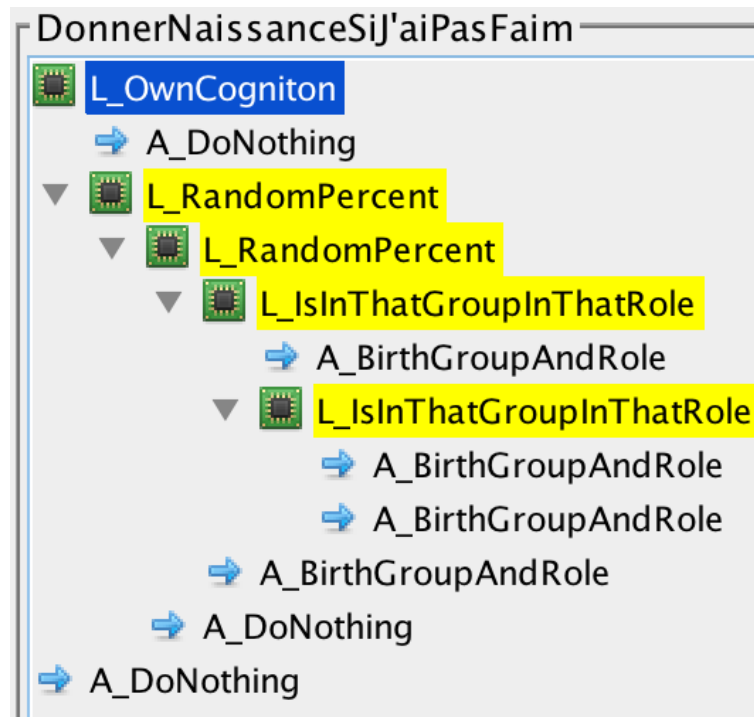


FIGURE 6 – Exemple de l'interface du plan "DonnerNaissanceSiJ'aiPasFaim" dans MetaCiv. Les actions conditionnelles sont repérées par une icône en forme de processeur et les autres actions par une icône en forme de flèche.

2.1.5 Les attributs

2.1.6 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 7 – Exemple de culturon (en marron).

2.2 Le modèle "Cueilleur-Artisan-Agriculteur"

Pour débiter, ce document explique le fonctionnement d'une modélisation ayant pour but de mettre en valeur l'émergence d'une civilisation. Cette implémentation se base sur MetaCiv, framework de mo-

délisation de société humaine utilisant la technique des SMA. Toutes les constantes mentionnées dans ce document 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

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 8 – 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.

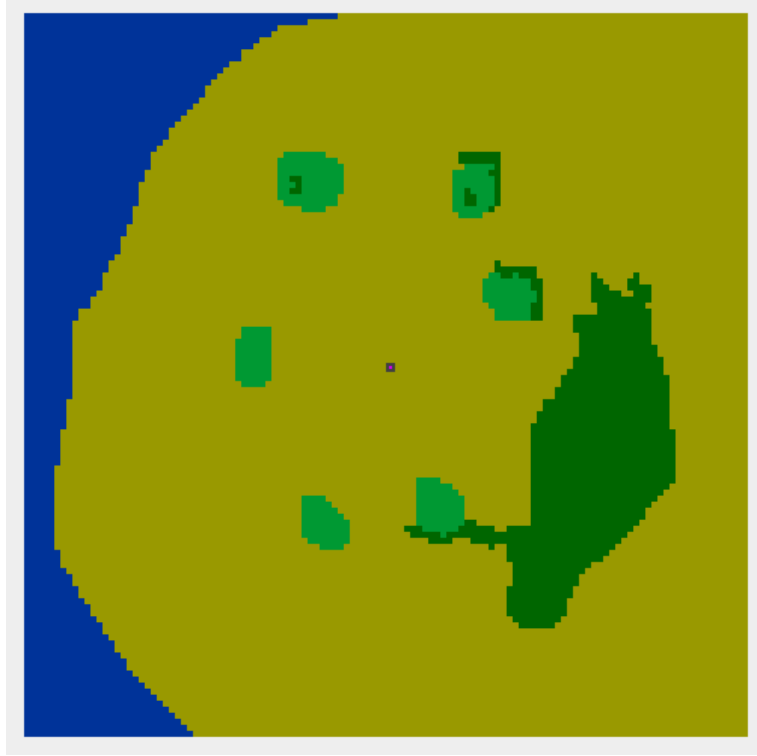


FIGURE 9 – environnement de la modélisation

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

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 reflète la dextérité d'un individu pour l'artisanat et n'est augmentée qu'après avoir fabriqué divers objets.
- L'attribut "CompetenceCueilleur" reflète 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 à chacun des agents, nous générons la fonction de masse de la loi de Poisson de paramètre $k = 20$ et $\lambda = 10$.

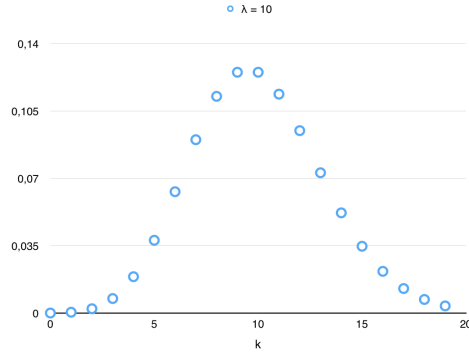


FIGURE 10 – Représentation de la fonction de masse de formule $P(k) = \frac{\lambda^k}{k!}e^{-\lambda}$

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 :

$$EsperanceDeVie = 5 * i'$$

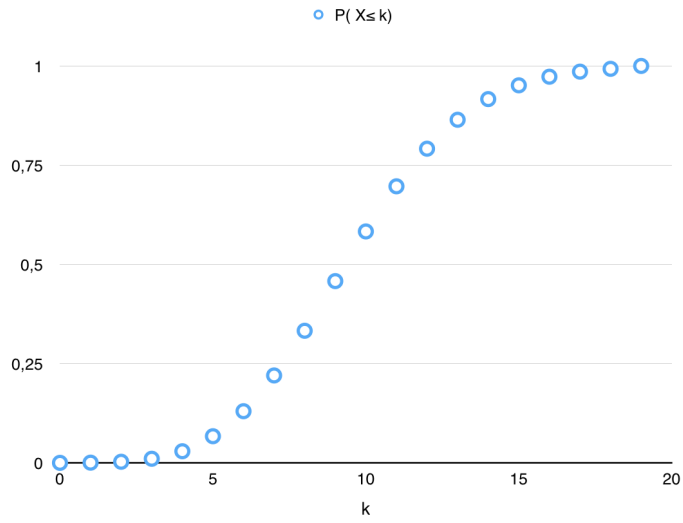


FIGURE 11 – Représentation de la fonction de répartition de la loi de Poisson de paramètre $\lambda = 10$ et $k = 20$.

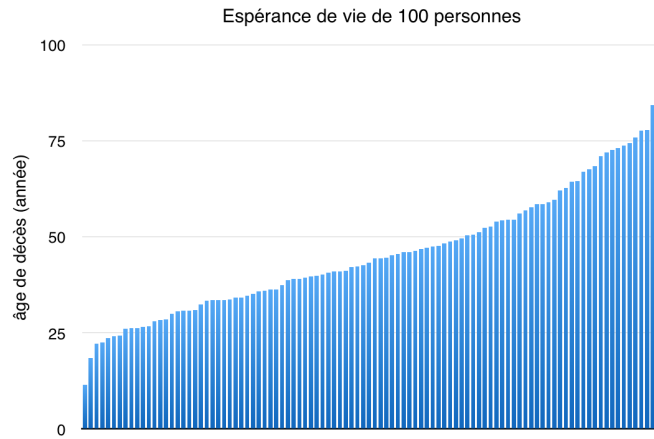


FIGURE 12 – Représentation triée de la valeur de l'attribut "EsperanceDeVie" de 100 agents.

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
CompétenceArtisanat	0
CompétenceCueilleur	0
Energie	100
EsperanceDeVie	Valeur initialisé par le "BirthPlan".

FIGURE 13 – 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 CompétenceEnArtisanat (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".

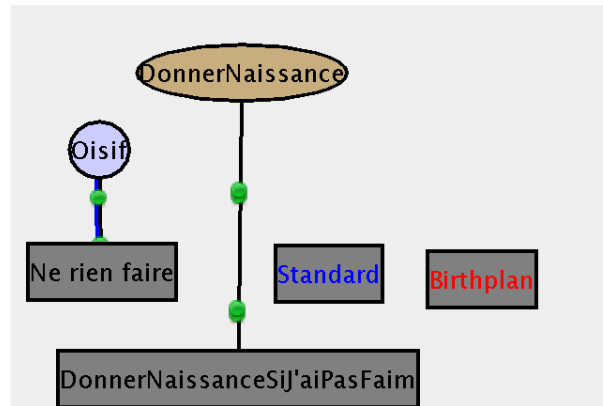


FIGURE 14 – Affichage des plans et cognitons associés dans MetaCiv (les couleurs des cognitons n’indiquent pas le type de cogniton).

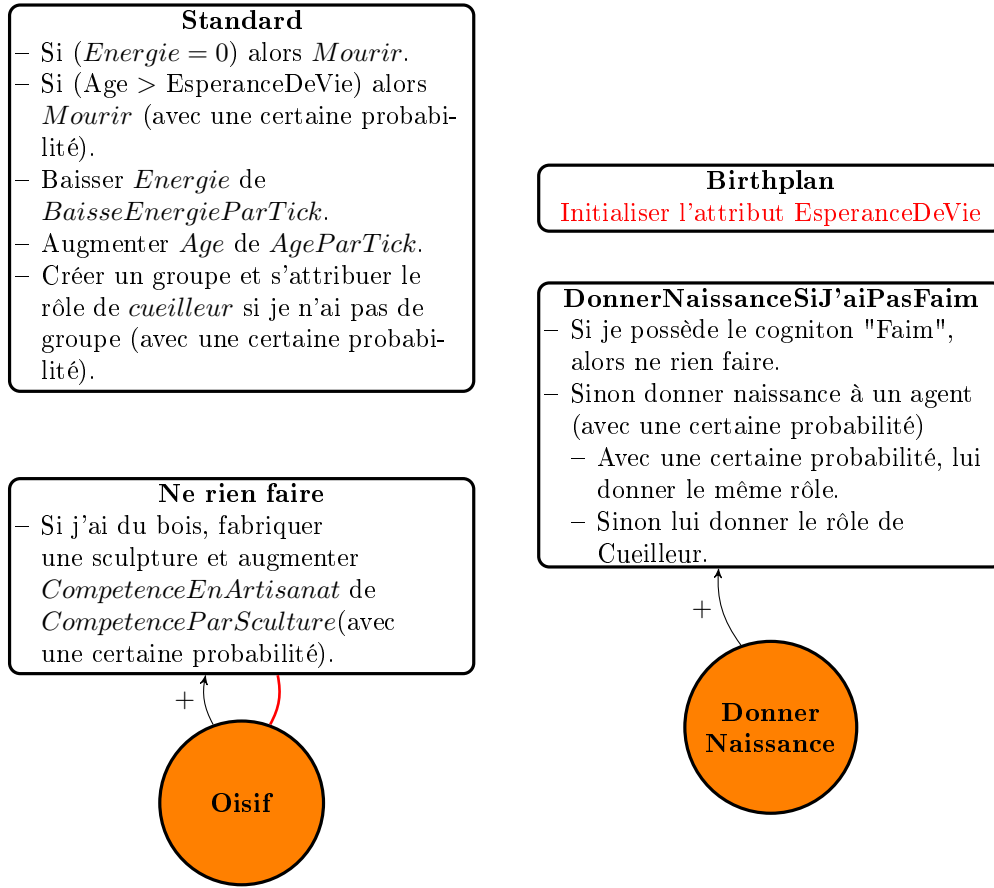


FIGURE 15 – Figure des plans "Ne Rien Faire", "Standard", "DonnerNaissanceSiJ'aiPasFaim" et "Birthplan".

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.

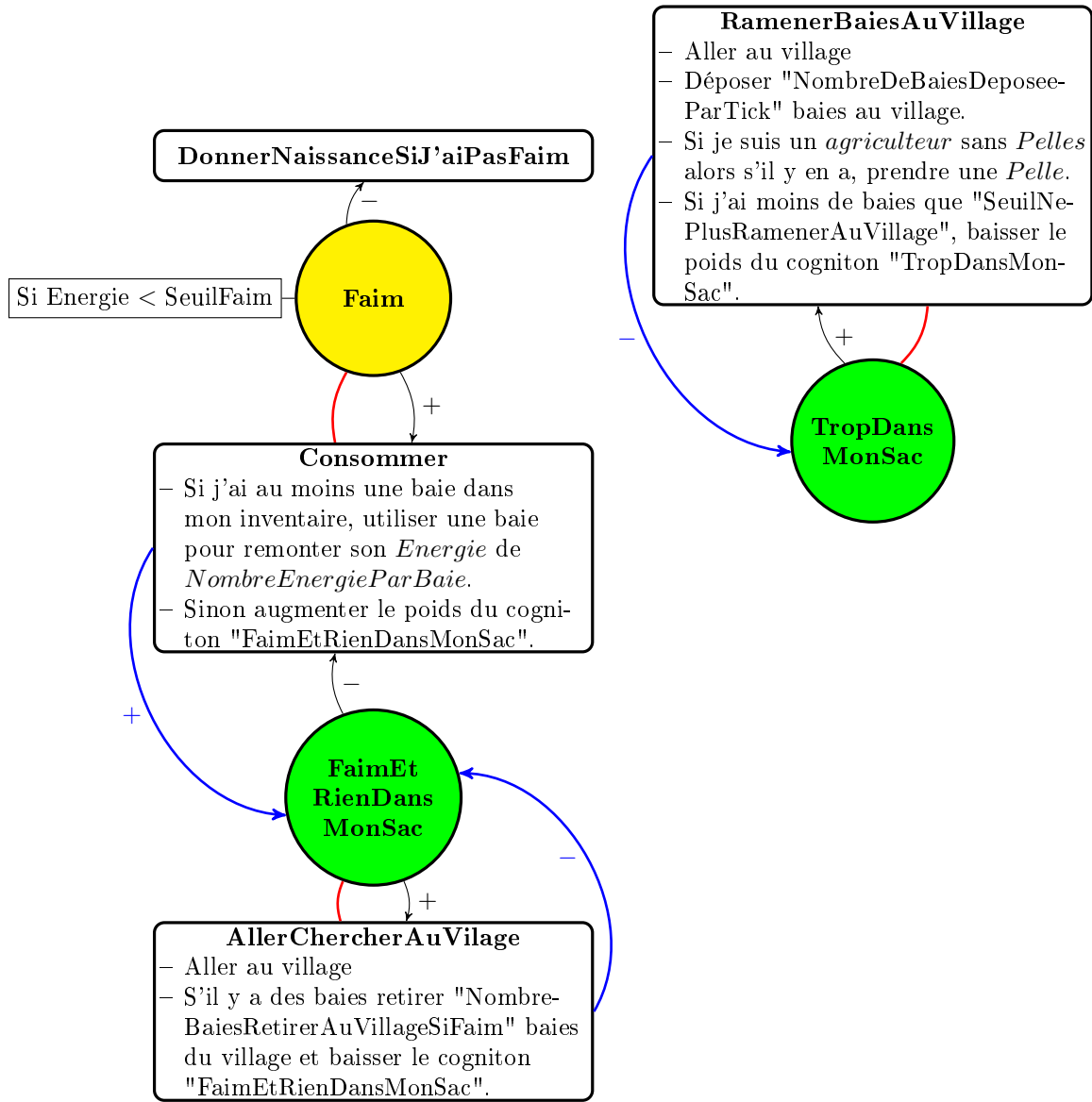


FIGURE 16 – Figure des plans "DonnerNaissanceSiJ'aiPasFaim", "Consommer", "AllerChercherAuVillage" et "RamenerBaiesAuVillage".

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.

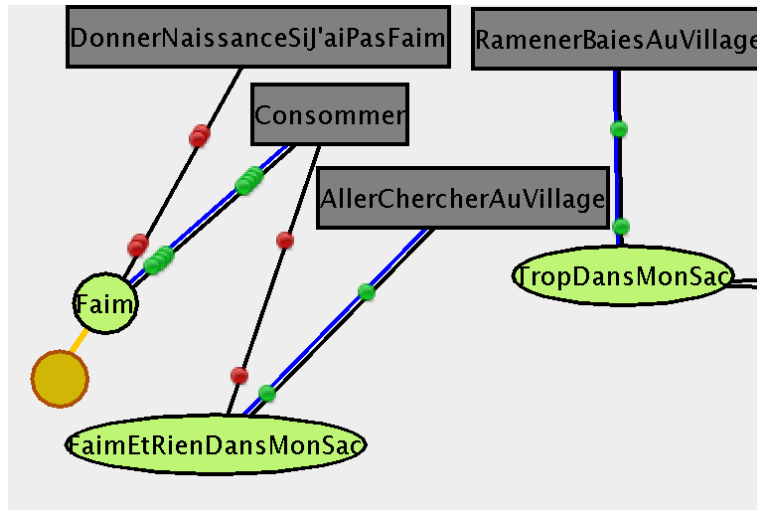


FIGURE 17 – Représentation des cognitons et plans associés dans MetaCiv.

2.2.4 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 a son attribut "CompétenceCueilleur" 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".

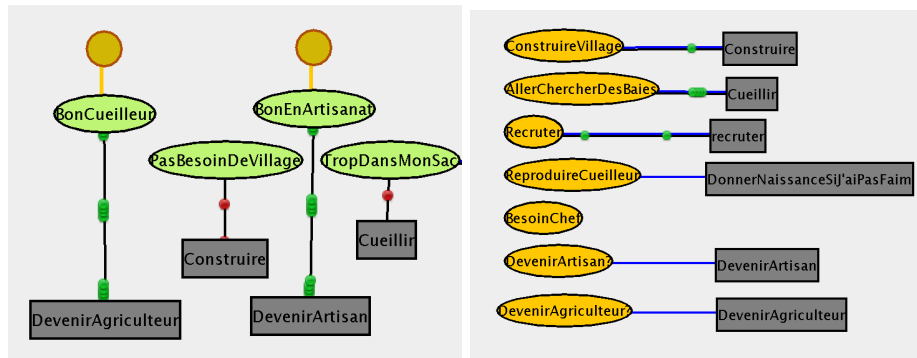


FIGURE 18 – Représentation des plan, culturons et cognitons du rôle cueilleur dans MetaCiv.

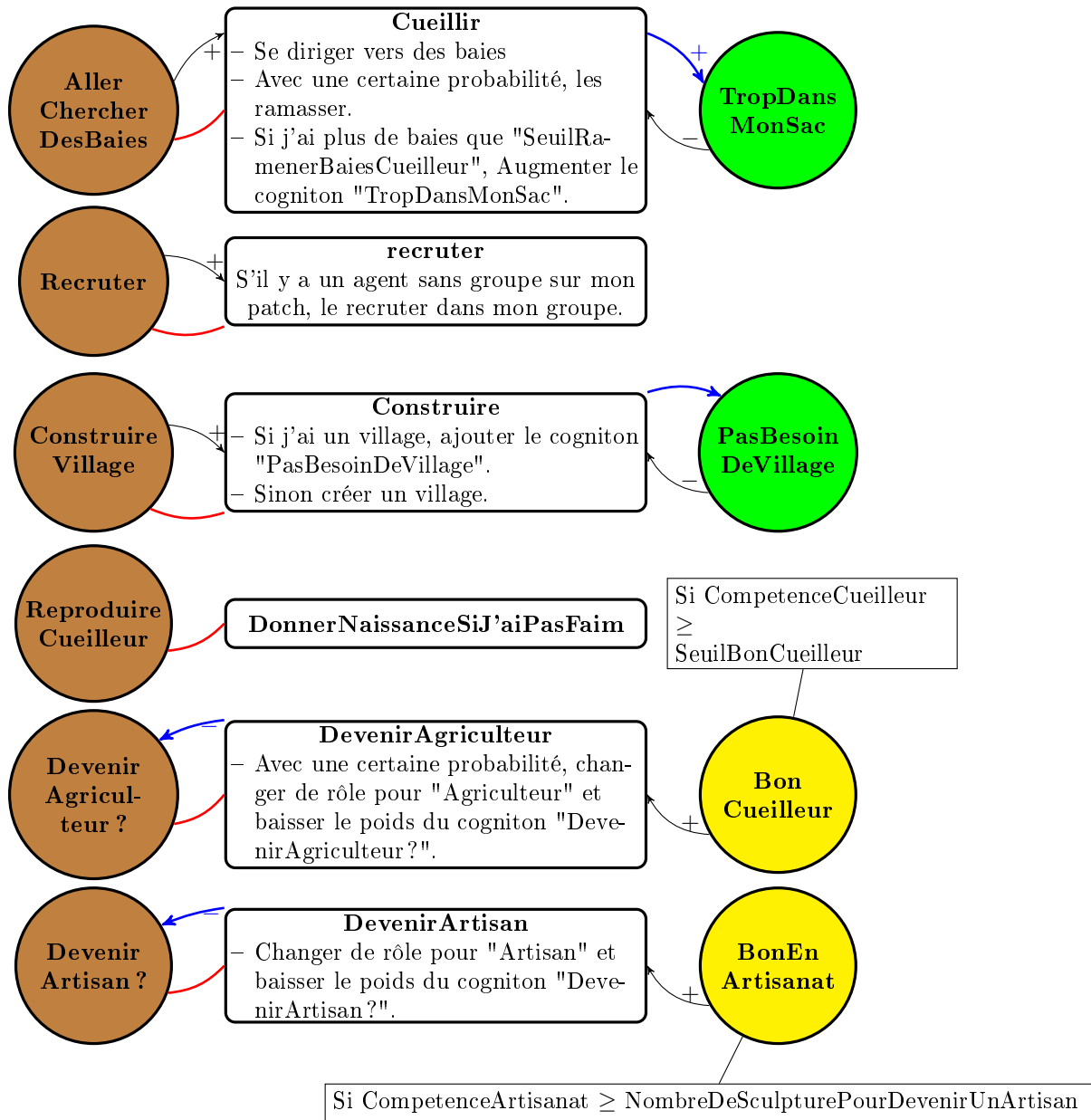


FIGURE 19 – Figure des plans d'un cueilleur et des interactions avec les cognitons associés.

2.2.5 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.

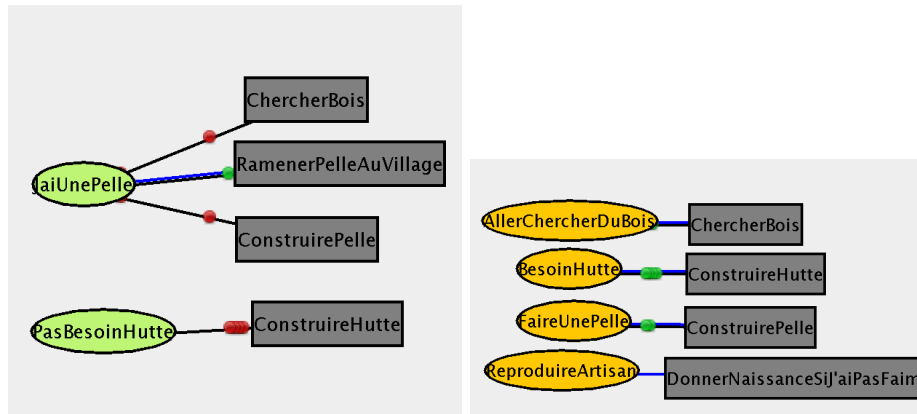


FIGURE 20 – Représentation des plans, culturons et cognitons du rôle artisan dans MetaCiv.

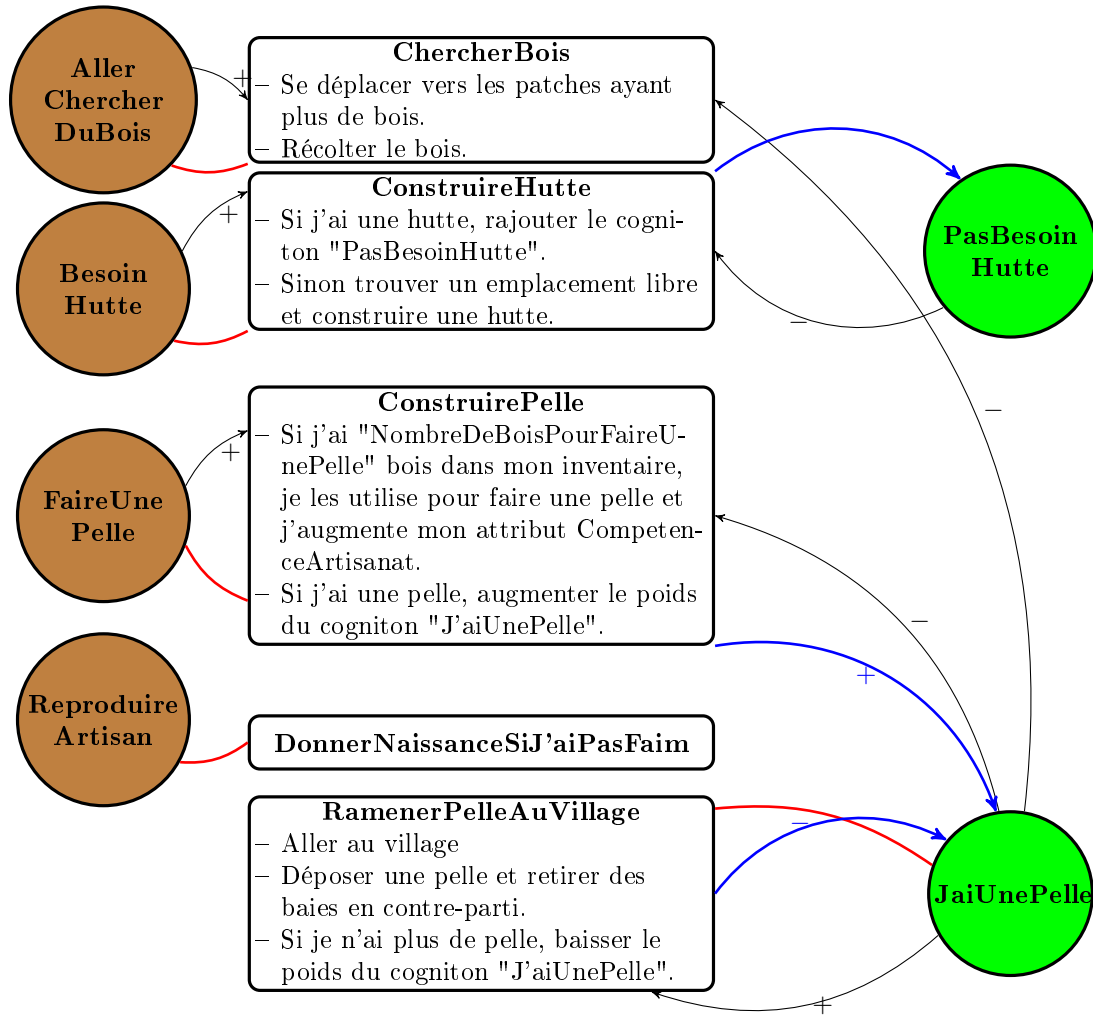


FIGURE 21 – Figure des plans d'un artisan et des interactions avec les cognitons associés.

2.2.6 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.

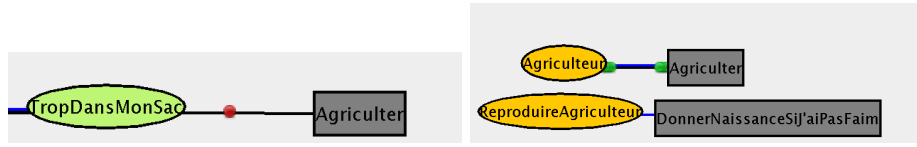


FIGURE 22 – Représentation des plan, culturons et cognitons du rôle agriculteur dans MetaCiv.

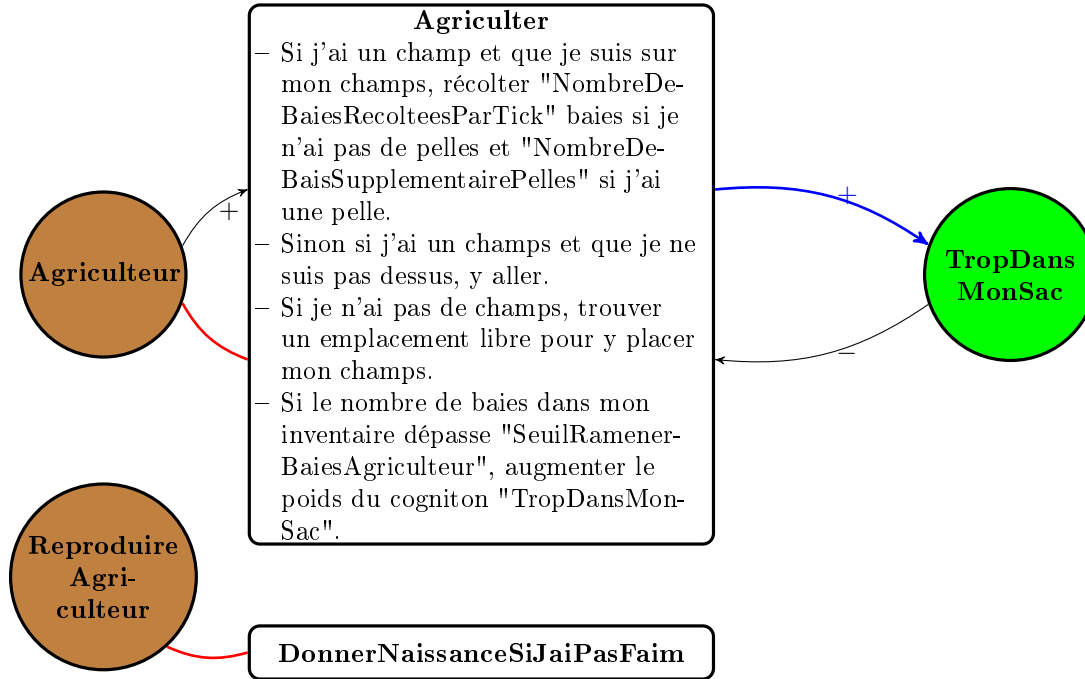


FIGURE 23 – Figure des plans d'un agriculteur et des interactions avec les cognitons associés.

2.3 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.




Aménagement	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 24 – Tableau récapitulatif des aménagements.

2.4 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).

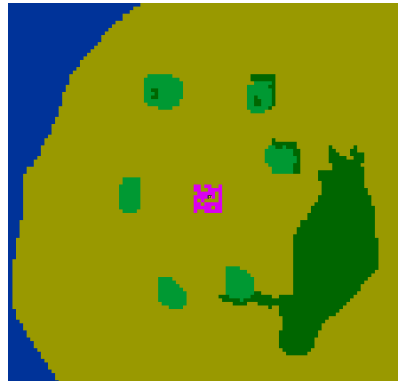


FIGURE 25 – Situation initiale

2.4.1 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.

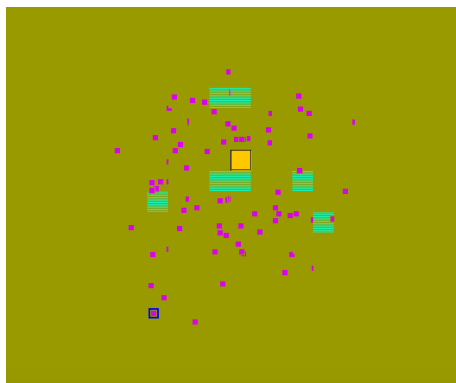


FIGURE 26 – Fondement des groupes et création des villages

2.4.2 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 27 – La recherche de nourriture

2.4.3 L'apparition de l'agriculture

Lorsque des cueilleurs ramènent des ressources au village, leurs attributs "CompétenceCueilleur" 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.

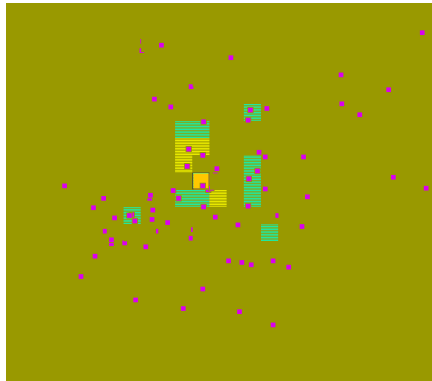


FIGURE 28 – Apparition des premiers champs et agriculteurs

2.4.4 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.

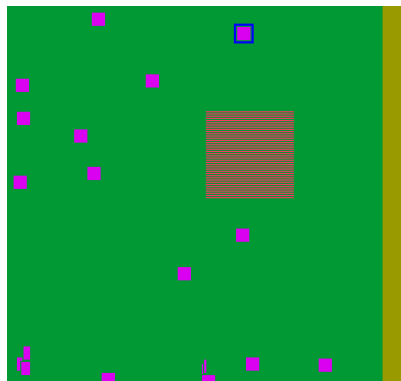


FIGURE 29 – Un artisan (entouré en bleu) proche de sa hutte.

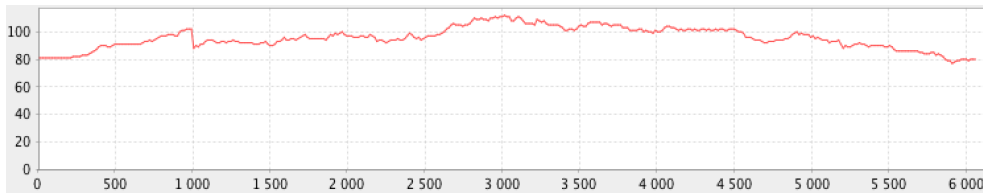


FIGURE 33 – Graphe représentant l'évolution de la population en fonction du temps dans une civilisation contenant des cueilleurs, artisans et agriculteurs.

Après observation de plusieurs graphes, on conclut 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.

2.5.2 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 fixée 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.

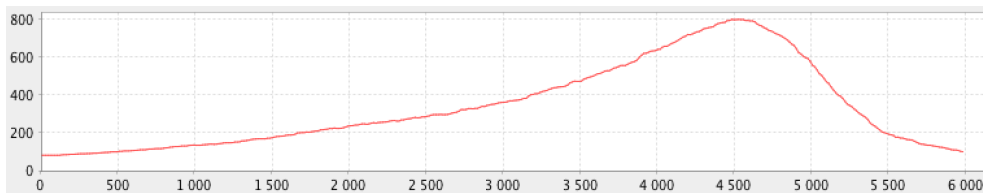


FIGURE 34 – Graphe représentant l'évolution de la population en fonction du temps dans une civilisation ne contenant que des cueilleurs.

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.

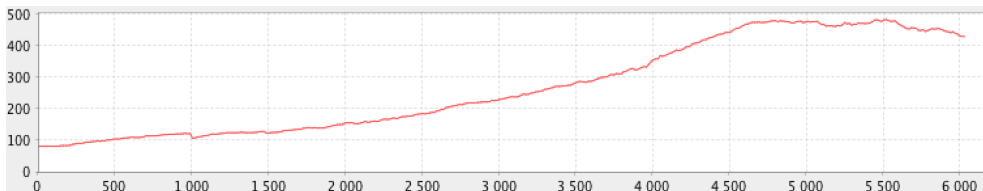


FIGURE 35 – Graphe représentant l'évolution de la population en fonction du temps dans une civilisation ne contenant que des cueilleurs et des artisans.

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.

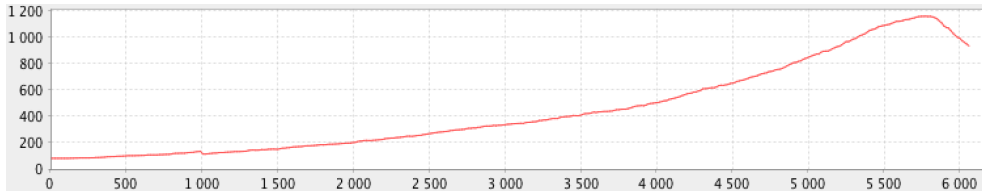


FIGURE 36 – Graphe représentant l'évolution de la population en fonction du temps dans une civilisation contenant des cueilleurs, artisans et agriculteurs.

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 "CompétenceCueilleur" 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 réglerait hypothétiquement le problème du manque d'agriculteurs.

2.5.3 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.

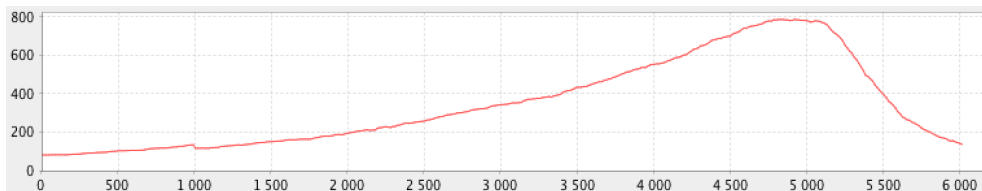


FIGURE 37 – Graphe représentant l'évolution de la population en fonction du temps dans une civilisation ne contenant que des cueilleurs.

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.

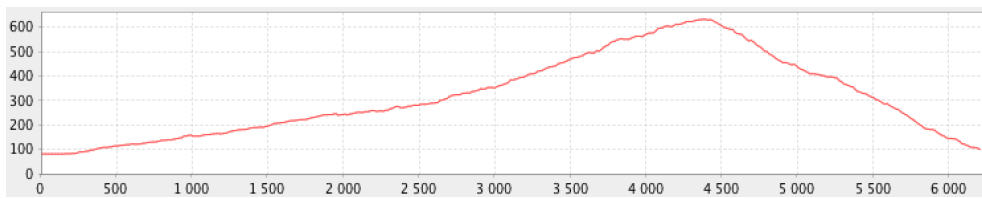


FIGURE 38 – Graphe représentant l'évolution de la population en fonction du temps dans une civilisation ne contenant que des cueilleurs et des artisans.

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.

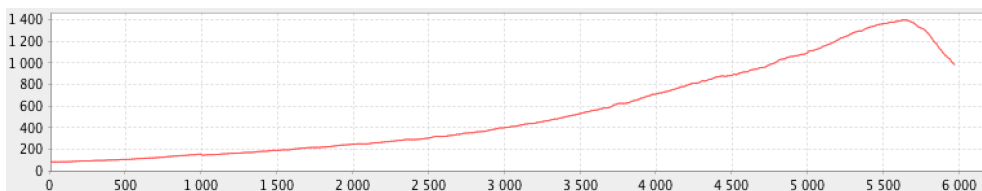


FIGURE 39 – Graphe représentant l'évolution de la population en fonction du temps dans une civilisation contenant des cueilleurs, artisans et agriculteurs

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.

2.6 Impressions sur l'utilisation de MetaCiv

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 cognitions 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 : cognitions, 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.

2.7 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
NombreDeSculpturesAjoutée	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 40 – Tableau récapitulatif des constantes de la modélisation.

2.8 Le modèle "Nomade-Sédentaire"

2.9 Observations

3 Le simulateur

3.1 Introduction

le programme MetaCiv , logiciel de modélisation de civilisation , est un projet qui a été développé par des groupes successifs d'étudiant. Développé autour de turtlekit et madkit , le programme possède une base solide , mais les ajouts successifs de fonctionnalités sans respect de normes et de méthodes on conduit à un logiciel instable et un code source chaotique.

nous avons essayé , lors de l'ajout de fonctionnalités , de produire un code plus clair , en séparant bien chaque chose (éviter les fonctions de plusieurs centaines de ligne), en utilisant une approche plus

objet et en utilisant quelques design patterns.

3.2 Fonctionnalités

3.2.1 Schéma cognitif

besoins : - 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 des espèces différentes, par exemple l'évolution des populations d'animaux sauvages à l'arrivée de l'homme - de manière générale , manipuler ces schémas cognitifs comme des éléments du modèle et non plus comme le modèle lui-même (cloner le schéma pour essayer différentes solutions , importer des schémas d'autres modèles ..., le tout en conservant un environnement défini)

pour ce faire nous avons crée une classe SchémaCognitif et nous y avons déporté tout ce qui relève du comportement des agents.

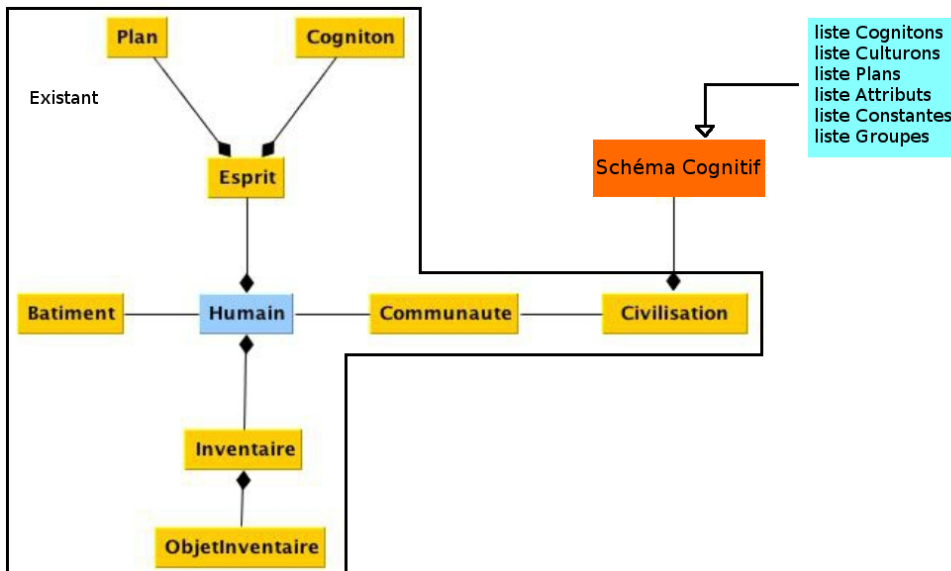


FIGURE 41 – Diagramme simplifié du programme

l'esprit fait toujours appel à la civilisation pour s'alimenter en plans et cognitons , mais nous avons changé le code des accesseurs pour qu'ils renvoient les données du schéma cognitif contenu dans la civilisation

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

FIGURE 42 – exemple d'un accès à la liste des plans disponibles, cerveau est la référence vers le schéma cognitif

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

enfin , nous avons crée les barres d'outils pour gérer l'édition des schémas cognitifs

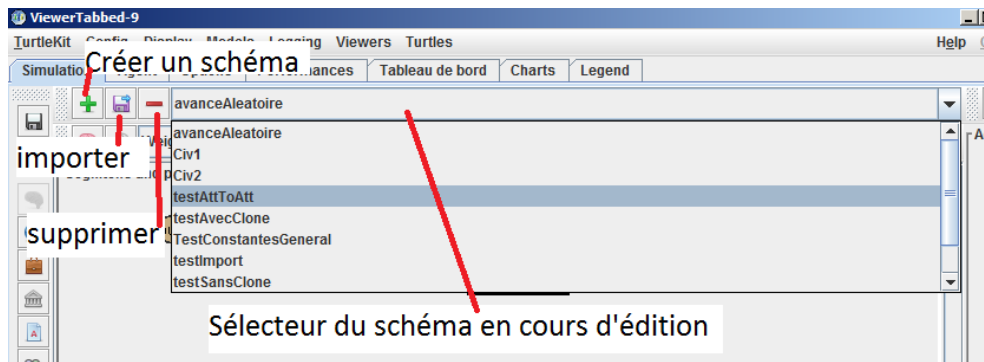


FIGURE 43 – barre d'outil permettant de sélectionner le schéma cognitif en cours d'édition , avec des options de création , d'import et de suppression

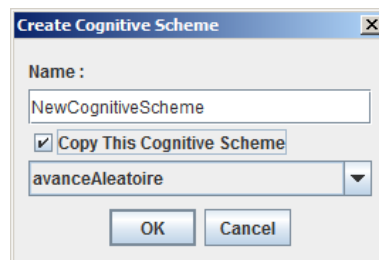


FIGURE 44 – fenêtre de création d'un nouveau schéma cognitif , possibilité de cloner un schéma existant en le renommant

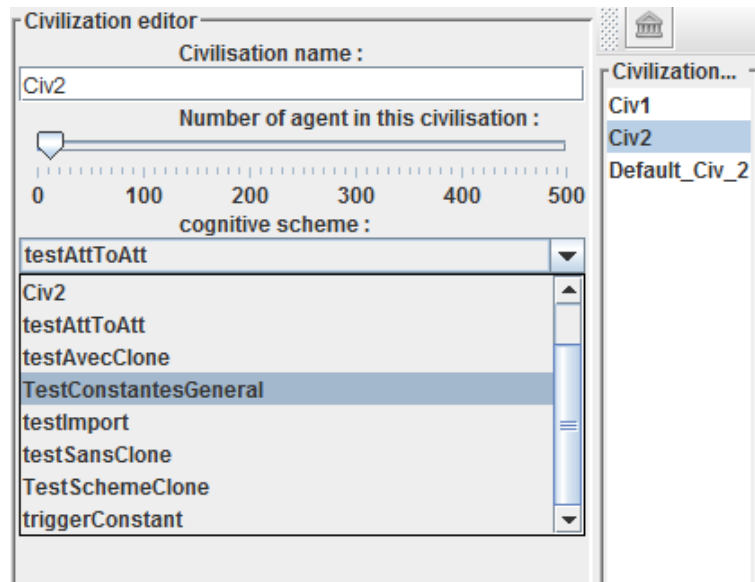


FIGURE 45 – ajout d’un sélecteur du schéma cognitif à utiliser dans l’interface de gestion des civilisations

3.2.2 Constantes

besoins : - centraliser les constantes utilisés à plusieurs endroits dans les plans pour simplifier le travail de réglage du modèle

3.3 Stabilisation du noyau

au cours de l’avancée des modélisateurs , quelques dizaines de bugs ont été découverts. Nous vous en présentons quelques uns en précisant le disfonctionnement constaté , sa raison et la correction effectuée

gestion des plans : au chargement , certains plans sont ajoutés deux fois (mais la simulation fonctionne quand même) -> un test toujours vrai dans le chainage des plans cause l’ajout en double -> nettoyage du code , le problème apparait clairement et est résolu (note : amélioration sensible des performances)

gestion des triggers : la modification des triggers ne semble pas avoir d’effet -> présence de deux listes représentant les triggers : une est utilisé par les agents , l’autre par l’interface. -> regroupement en une seule liste , modification de la liste au lieu de détruire et reconstruire (permet la modification des valeurs pendant l’exécution)

certain cognitons sont présent en double , ce qui fausse le calcul de l’action à effectuer -> les Starting cognitons sont ajoutés en plus de la liste de tous les cognitons qui les contient déjà -> sécurisation de la liste des cognitons d’un agent à travers un accesseur , si le cogniton est déjà présent , l’ajout est ignoré

la sauvegarde sous windows corrompt certain modèles (pas sur mac) -> pas de vérification des caractères interdits dans le nom des éléments qui sont utilisés pour nommer le fichier correspondant -> utilisation de URLEncoder sur toutes les chaines servant comme nom de fichier

3.4 Autre

3.5 Conclusion

4 Gestion de projet

5 Conclusion et perspectives