Analyse et conception du leadership et de l'utilisation de la mémoire dans le simulateur PARIS

Raphaël Martin

2 

Institut **S**upérieur d'**I**nformatique
de **M**odélisation et de leurs **A**pplications

Complexe des Cézeaux

BP 125 – 63173 AUBIERE CEDEX

Institut **N**ational de la **R**echerche
Agronomique

URH - RAP
Theix
St Genès-Champanelle

RAPPORT DE STAGE 2ème ANNEE

Analyse et conception du leadership et de l'utilisation de la mémoire dans le simulateur PARIS

Tome I

Présenté par : Raphaël MARTIN
Responsable du stage : Laurent PEROCHON
Bertrand DUMONT
Tuteur ISIMA : Claude MAZEL
Lieu de stage : INRA Clermont-Ferrand Theix

Du 5 Avril au 23 Septembre 2004

Institut Supérieur d'Informatique
de Modélisation et de leurs Applications

Complexe des Cézeaux

BP 125 – 63173 AUBIERE CEDEX

Institut National de la Recherche
Agronomique

URH - RAP
Theix
St Genès-Champanelle

RAPPORT DE STAGE 2ème ANNEE

Analyse et conception du leadership et de l'utilisation de la mémoire dans le simulateur PARIS

Tome I

Présenté par : Raphaël MARTIN
Responsable du stage : Laurent PEROCHON
Bertrand DUMONT
Tuteur ISIMA : Claude MAZEL
Lieu de stage : INRA Clermont-Ferrand Theix

Du 5 Avril au 23 Septembre 2004

Remerciements

Je tiens à remercier tout d'abord M. Laurent Pérochon pour son aide en matière d'analyse, ses conseils et sa sympathie tout au long du stage. Je remercie aussi M. Bertrand Dumont qui m'a expliqué lors de nombreuses réunions les mécanismes de mémorisation et leur modélisation.

Je remercie également M. Claude Mazel ainsi que M^{me} Christine Force pour leur intérêt et leurs idées.

Enfin je remercie l'INRA de Theix pour son accueil ainsi que mes camarades stagiaires pour leur gentillesse.

Table des figures et illustrations

Table des abréviations

FGEP	: Fonctionnement et Gestion de l'Ecosystème Prairial
IHM	: Interface Humain Machine
INRA	: Institut National de la Recherche Agronomique
ISIMA	: Institut Supérieur d'Informatique, de Modélisation et de leurs Applications
JVM	: Java Virtual Machine
LIMOS	: Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes
ModVege	: Modèle Végétal
MVC	: Model / View / Controller
PARIS	: PAsture Ruminants Interactions Simulator
RAPA	: Relation Animal/Plantes et Aliments
Rs	: Compartiment Reproducteur sec (tiges et épis secs)
RTF	: Rich Text Format
Rv	: Compartiment Reproducteur vert (tiges et épis verts)
SDD	: Structure De Données
UML	: Unified Modelling Language – Langage de modélisation unifié
URH	: Unité de Recherche sur les Herbivores
Vs	: Compartiment Végétatif sec (gaines et feuilles sèches)
Vv	: Compartiment Végétatif vert (gaines et feuilles vertes)

Glossaire

Abscission	: Fonction qui détermine la quantité de matière sénescente qui sort du système chaque jour.
Biomasse	: Quantité de matière végétale (fourrage) disponible sur une cellule.
Cellule	: Une cellule est l'entité végétale et spatiale élémentaire.
cfg	: Extension de fichier de configuration au format texte pouvant être édité par des éditeurs basiques et capable de supporter des lignes de commentaires.
Croissance	: Fonction déterminant la quantité de biomasse qui sera produite chaque jour par le compartiment Vv.
csv	: Extension de fichier au format texte utilisant le séparateur point virgule.
Défolier	: Action de prélever une quantité de biomasse d'une cellule.
Doxygen	: Logiciel libre permettant la génération automatique de documentations à partir d'une nomenclature de commentaire.
Faciès	: Ensemble unique représentant une communauté végétale de quelques espèces dans des proportions fixées.
Graphviz	: Utilitaire permettant de générer automatiquement des diagrammes UML à l'aide de Doxygen.
Grégarisme	: Relation sociale au sein d'un troupeau de ruminants qui pousse les animaux à rester groupés.
Litière	: Fonction calculant le flux d'abscission des compartiments sénescents.
Montaison	: Fonction simulant le développement de la phase reproductrice des espèces végétales.
Leadership	: Relation sociale au sein d'un troupeau qui pousse les animaux à suivre un leader lors des déplacements longs
Sénescence	: Fonction simulant les mécanismes de vieillissement naturel des tissus végétaux.
Station alimentaire	: Surface virtuellement exploitable par un animal sans qu'il bouge les pattes avant.

Résumé

Mon stage consiste à compléter un **simulateur** parcelle/troupeau de manière à le rendre capable de gérer la **mémoire spatiale** et, ainsi, de fournir un premier prototype complet et opérationnel.

Le **simulateur** PARIS modélise l'interaction entre une prairie et un troupeau de ruminants le pâturant. Afin de simplifier ce modèle, il a été divisé en plusieurs sous-modèles : le modèle végétal gère la dynamique de la végétation, répercute les défoliations et fournit au modèle animal les informations dont il a besoin. Ce dernier met à jour l'état physiologique de l'animal et gère ses prises de décision . Enfin vient s'ajouter le sous-modèle social qui permet de gérer les interactions entre les animaux. Ce dernier sera le cœur de mon travail.

Mon stage s'est décomposé en plusieurs étapes. J'ai tout d'abord analysé le **simulateur** en l'état, installé une **documentation** grâce à l'outil **Doxygen** et réalisé quelques modifications qui améliorent la lisibilité et optimisent le programme. J'ai ensuite mis à jour la **visionneuse** de façon à ce qu'elle accepte les sorties actuelles du **simulateur** et qu'elle puisse afficher plusieurs trajectoires. J'ai de plus créé une interface pour cette dernière. Enfin, j'ai intégré la **mémoire spatiale** au **simulateur** telle qu'elle avait été implémentée puis je l'ai améliorée afin qu'elle respecte au mieux les observations in situ. Chacune de ces étapes a d'abord été modélisée avec la norme **UML** puis implémentée sous Linux avec le langage **C++** ou **JAVA** dans le cas de l'interface.

A l'heure actuelle, mon travail n'est pas entièrement fini. En effet, l'oubli et l'exploration ont été conceptualisés mais non implémentés. L'objectif de mon stage sera donc atteint fin septembre.

Mots-clés : simulateur, mémoire spatiale, documentation, Doxygen, visionneuse, UML, C++, JAVA

Abstract

My training period consists in completing a plot/herd **simulator** so as to make **spatial memory** working , and then to create a first operationnal complete prototype.

The PARIS **simulator** models interactions between a meadow and a ruminant herd grazing it. In order to simplify this model, it was divided into three sub-models : the vegetation model manages vegetation dynamics, takes into account defoliations and provides to the animal model informations which it needs. This last sub-model updates animal physiological state during a simulation. Finally, the social model deals with interactions between animals. This part will be the center of my work.

My work was broken up into several steps. Firstly, I analysed the code of the **simulator** as it was, installed a **documentation** with the freeware **Doxygen** and realized some modifications which improve legibility and optimize the program. Then I updated the "**visionneuse**" so that it can use current output files and dispay on screen more than one trajectory. To use it easily, I created a **graphic interface.** Lastly, I integrated the **spatial memory** into the **simulator** such it had been implemented. Then I modified it to make it more realistic. Each step has been modeled with the **UML** modeling language, before to be programmed under Linux whith the **C++** language and the **JAVA** language for the "**visionneuse**" graphic interface.

My work remains unfinished as we still need to add the forgetting and the exploration to the spatial memory. This should be done by the end of september.

Keywords : simulator, spatial memory, documentation, Doxygen, visionneuse, UML, JAVA, C++

Table des matières

TOME I

TOME II

Introduction

L'équipe RAPA de l'Unité de Recherche sur les Herbivores de l'INRA développe actuellement, en collaboration avec l'équipe FGEP de l'Unité d'Agronomie de l'INRA et le LIMOS de l'Université Blaise Pascal, un simulateur parcelle/troupeau nommé PARIS dans le cadre d'un projet de recherche intitulé « l'utilisation par le pâturage et l'évolution de la végétation des surfaces herbagères hétérogènes et peu chargées ». Celui-ci a pour but de modéliser les interactions entre un troupeau de ruminants et la parcelle sur laquelle il se trouve, et ce afin d'observer l'impact du pâturage sur la végétation et ainsi d'éviter l'embroussaillement des parcelles. L'implémentation de ce simulateur a commencé il y a six ans et fait toujours l'objet de nombreux stages et projets.

Le prototype actuel permet déjà de modéliser la dynamique d'un troupeau en prenant en compte l'état physiologique de chaque ruminant ainsi que ses relations sociales avec le reste du groupe, tel que le grégarisme ou le leadership. Toutefois, il n'inclut ni la mémoire, ni les dernières modifications apportées à la modélisation des végétaux, et reste encore en cours de validation.

Mon travail consistera donc dans un premier temps à modifier le simulateur pour que ce dernier prenne en compte les modifications les plus récentes. Je devrai dans un second temps réaliser une analyse complète du modèle social et surtout de la mémoire afin d'intégrer cette dernière à l'existant.

Je présenterai d'abord l'état du simulateur tel qu'il était avant le début de mon stage puis, dans un second temps, je décrirai le travail de conception et de réalisation effectué. Enfin, j'exposerai les résultats obtenus ainsi que les futures améliorations qui pourront être apportées.

Partie 1 : Contexte technique

1.1 Description de l'approche multi-agents

Un système multi-agents consiste à faire coopérer un ensemble d'entités ou agents dotées d'un comportement intelligent, à coordonner leurs buts et leurs plans d'actions pour résoudre un problème [LEMLOUMA et BOUDINA 2001].

On peut donc déduire les caractéristiques principales de ces entités :

- Autonomie dans la prise de décision.
- Connaissance d'elles-mêmes mais aussi des autres.
- Capacité d'agir.

Dans notre cas, le système complexe qu'est le troupeau peut être décomposé en éléments simples que sont chacun des ruminants [DUMONT et HILL 2003]. On peut donc analyser les informations séparément et les combiner afin de comprendre le comportement du groupe.

1.2 Description et analyse de l'existant

Ce simulateur a pour objectif de modéliser le comportement d'un troupeau d'herbivores pâturant sur une parcelle de prairie hétérogène [BAUMONT et coll. 2002]. Pour représenter les interactions entre animaux, la variabilité de leur comportement et l'utilisation spatiale d'une végétation hétérogène, une approche multi-agents spatialisée a été choisie [PEROCHON et al. 2001]. Ce troupeau est constitué d'un nombre quelconque d'animaux qui suivent tous un comportement aussi bien individuel que collectif; la croissance et le vieillissement de la végétation est elle aussi modélisée [CARRERE et al. 2002]. Tout cela implique un haut niveau de complexité et a amené à décomposer ce simulateur en plusieurs sous-modèles qui seront chargés chacun de gérer un point précis, comme le conseille l'adage « Diviser pour mieux régner ».

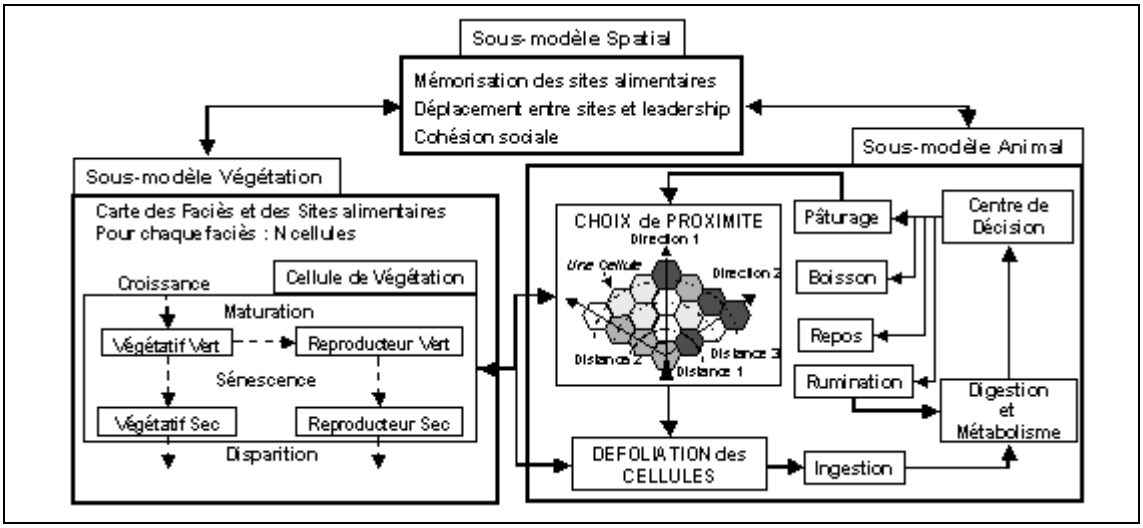

Figure 1 : Représentation simplifiée du simulateur [BAUMONT et coll. 2002]

Trois modèles principaux composent le simulateur. A ceux-ci vient s'ajouter le modèle technique qui contient tous les aspects propres à la simulation et non au domaine étudié. Je vais donc les décrire un par un, en insistant tout particulièrement sur le modèle Social, cœur de mon travail.

1.2.1 Le modèle Végétal

1.2.1.1 Modèle conceptuel

Ce modèle a plusieurs objectifs :

- la mise à jour de la parcelle, via la sénescence et la montaison de la végétation en tenant compte du climat.
- la mise à disposition d'informations destinées au modèle Animal lors du choix par le ruminant de la cellule à défolier.
- la mise à jour, suite à la défoliation d'un animal, de la quantité de biomasse disponible dans la cellule.

Le modèle Végétal, ou ModVege, s'occupe de la gestion de la parcelle au niveau de la végétation. Cette dernière se décompose en cellules hexagonales de 0.1m², ce qui représente la station alimentaire moyenne d'un mouton. Chacune de ces cellules est associée à un faciès, défini par 25 paramètres qui spécifient entre autre le seuil de démarrage de la végétation, sa précocité d'épiaison, sa digestibilité ou la vitesse d'abscission des tissus morts. Elles appartiennent de plus toutes à un site défini comme étant un ensemble de cellules voisines associées au même faciès.

Chaque cellule est divisée en quatre compartiments : gaines et feuilles vertes (végétatif vert ou Vv), gaines et feuilles sèches (végétatif sec ou Vs), tiges et épis verts (reproducteur vert ou Rv) et tiges et épis secs (reproducteur sec ou Rs). Chacun

de ces compartiments est décrit par un ensemble de variables d'état, comme la biomasse, et par des variables quantitatives, comme la digestibilité des parois végétales. A ces compartiments vient s'ajouter un cinquième fictif, la litière, qui correspond dans ce modèle au flux de sortie du système.

Les interactions entre les compartiments sont simulées à partir de six fonctions qui calculent les flux de biomasse [CARRERE et al. 2002] : une fonction de croissance, une fonction de maturation, deux fonctions de sénescence des compartiments végétatifs et reproducteurs et enfin deux fonctions de disparition des tissus morts.

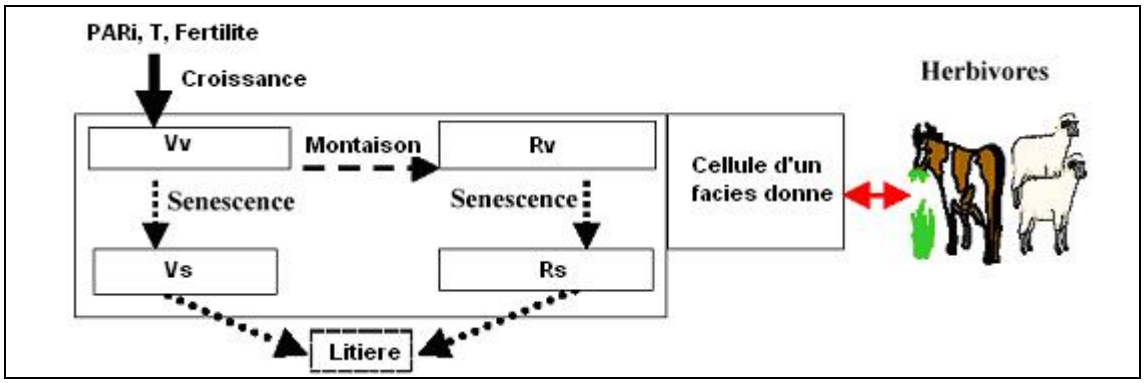

Figure 2 : Structure générale du sous-modèle Végétal

1.2.1.2 Etat d'avancement

Ce modèle est entièrement implémenté et répond aux trois objectifs décrits plus haut. Les fonctions de montaison, de sénescence et de litière ont été implémentées de manière à ce que les flux de biomasse soient plus proches de l'observation *in situ* [MARTIN et PEAN, 2004] et sont actuellement en attente de leurs validations [ASTRE, 2004].

1.2.2 Le modèle Animal

1.2.2.1 Modèle conceptuel

Le Modèle Animal reprend un ancien modèle d'ingestion [SAUVANT et al. 1996]. Il gère les paramètres d'ingestion, l'état physiologique de l'animal ainsi que les actions qu'il va choisir. L'animal peut soit manger, soit se déplacer sur une cellule voisine afin de la défolier ou encore se reposer. Lors des repos ou des déplacements l'animal peut ruminer. Il faut bien dissocier le repos court, période d'indécision, du repos long durant lequel l'animal se couche et dort.

1.2.2.2 Etat d'avancement

Le modèle animal est opérationnel, c'est-à-dire qu'il gère l'état interne de l'animal ainsi que ses déplacements de proximité grâce à un générateur de trajectoires. Ce dernier permet de déterminer la trajectoire suivie par l'animal sur la parcelle pendant une phase de défoliation. En effet, lorsque l'animal a choisi de brouter, il a à faire un choix entre les quinze cellules de son champ de vision. On calcule alors les probabilités que l'animal aille sur chacune des cellules en fonction de la distance qui les sépare, de la qualité de la cellule ainsi que de la direction pour l'atteindre.

En ce qui concerne le choix de l'activité de l'animal, deux algorithmes de décision existent. Le premier représente une vision simpliste du problème puisque l'animal a des choix prioritaires. La figure 3 montre son principe :

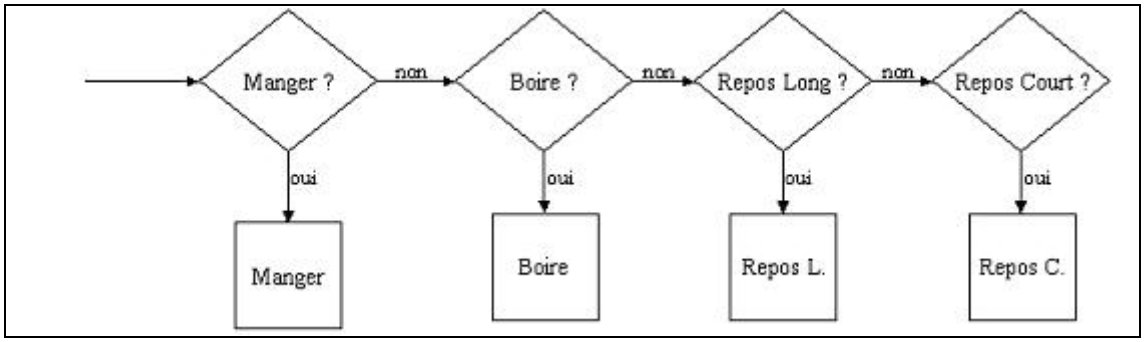

Figure 3 : Premier algorithme de décision

Le problème que soulève cet algorithme est le suivant : un animal peut mourir de soif au bord d'un site de buvée. En effet, tant que ce dernier aura faim, il mangera et donc ne s'abreuvera pas.

C'est pour cela qu'a été développé un automate décisionnel [MASSON 2001]. Il permet de pallier le défaut présenté ci-dessus et de modéliser ainsi un comportement plus crédible puisque l'animal peut effectuer des choix opportunistes, c'est-à-dire que ce n'est plus seulement son état physiologique qui régit son choix mais aussi sa position sur la parcelle. Ainsi, on détermine tout d'abord une probabilité pour chaque action, dans notre exemple faim prend comme valeur 1, soif 0.3 et repos long 0.01. On divise ensuite toutes ces propriétés par leur somme multipliée par 1.1, ce qui laissera une petite part de chance d'avoir un repos court qui représentera une certaine indécision. On divise ensuite chacune d'entre elles par le résultat obtenu. On obtient dans notre cas pratique :

Faim = 0.7
Soif = 0.2
Repos long = 0.01
Repos court = 0.09

Ceci nous donne la figure 4 :

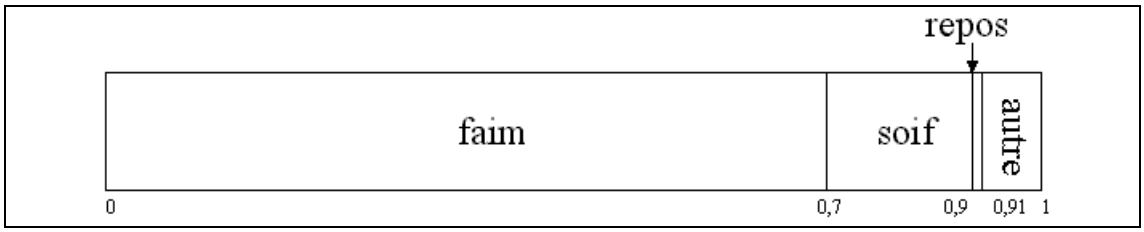

Figure 4 : Automate décisionnel [MASSON 2001]

On considère deux types de déplacements : tête haute, dans le cas de déplacements longs (voir paragraphe 1.2.3.1), et tête basse pour les déplacements de proximité. Comme deux animaux ne peuvent pas se retrouver au même moment sur une même cellule, lorsqu'un ruminant arrive sur une cellule déjà occupée, il devra choisir immédiatement une nouvelle cellule libre dans son champ de vision.

1.2.3 Le modèle social

1.2.3.1 Modèle conceptuel

C'est ce modèle qui gère les relations sociales au sein du troupeau. Ces relations peuvent être de deux types : le grégarisme ou le leadership. Elles influencent toutes deux le comportement de l'animal puisqu'elles agissent sur ses déplacements.

Afin de modéliser correctement ces comportements, chaque animal possède une mémoire. Il doit pour cela, après chaque défoliation, mémoriser la qualité de la cellule en partie ingérée. Cette qualité est calculée à partir d'une des deux fonctions de perception (voir Annexe E) choisie par l'utilisateur au lancement de la simulation. A partir de ces valeurs, le ruminant crée des sites mémoriels rassemblant les cellules de qualité similaire qui sont adjacentes.

De plus, lorsque l'animal retourne sur un site visité, une mise à jour de la qualité du site a lieu et sa forme est recalculée. Si la qualité du site visité est trop différente de la qualité mémorisée, on scinde alors ce dernier en deux nouveaux sites distincts : l'ancien et la partie en cours de défoliation, c'est ce que l'on nomme la désagrégation

Toutefois cette mémoire doit être de capacité limitée, afin de respecter les observations faites sur le terrain [DUMONT et HILL 2001]. Ainsi, si à l'ajout d'un nouveau site, la mémoire est pleine, l'animal oubliera le site de plus mauvaise qualité, c'est-à-dire celui dont le coefficient $\log(nb\_visites+1)\times qualite$ est le plus faible. De plus, une fonction d'oubli vient s'ajouter à tous ces processus, ce qui permet de faire converger les qualités des sites mémoriels vers leur moyenne.

Mais cette mémoire ne peut pas être utilisée par tous les animaux. A un instant donné, un unique animal peut devenir le leader. Chaque ruminant a sa propre probabilité de le devenir, fixée dans un fichier de configuration. Ainsi, lors du choix d'une activité, on vérifie préalablement que la position actuelle de l'animal est adaptée à son activité. Si sa position est satisfaisante, l'animal réalise cette activité, sinon il effectue un tirage aléatoire pour déterminer s'il devient le leader. Deux cas se présentent alors :

- S'il n'est pas leader, il effectue une action par défaut en procédant à un nouveau choix d'activité.
- Il « réussit » son tirage. Deux possibilités s'offrent maintenant à lui. Il peut :

  - Explorer la parcelle avec une probabilité de $1-\frac{SurfaceVisitée}{SurfaceTotale}$. Il se déplace donc tête basse aléatoirement vers une des quinze cellules de son champ de vision, ce qui va lui permettre d'augmenter le nombre de sites connus de sa mémoire.
  - Consulter sa mémoire et choisir le meilleur site mémorisé, c'est-à-dire celui qui a le plus grand coefficient $\frac{qualite \times surface}{eloignement}$. Une fois ceci réalisé, l'animal détermine le point de ce site le plus proche par rapport à sa position actuelle et s'y dirige, en ayant préalablement annulé toute autre activité. A partir de cet instant, les animaux réalisent alors tous un test d'éloignement par rapport au leader qui va permettre au troupeau de le suivre. Afin d'éviter un départ global de la part de tous les individus à un instant donné, on répartit uniformément les tests entre deux dates.

1.2.3.2 Etat d'avancement

Le modèle social n'est pas encore totalement effectif car la mémorisation a été développée indépendamment du simulateur. De plus, la notion de site de buvée et de repos n'est pas implémentée. L'animal ne fait donc pour l'instant appel qu'au leadership alimentaire, ce dernier lui renvoyant comme coordonnée de destination un abscisse et une ordonnée fixe. Les relations sociales au sein du troupeau sont quant à elles effectives.

1.2.3.2.1 Le grégarisme

1.2.3.2.1.1 Tests d'éloignement

Pour être en accord avec la grégarité, une règle a été implémentée afin d'assurer la cohésion sociale du troupeau. Ainsi, les animaux ne doivent ni trop s'éloigner les uns les autres, ni être trop proches. Chaque animal vérifiera donc à intervalles réguliers, deux minutes précisément, s'il n'est pas trop loin du reste du troupeau ni trop prés d'un autre animal.

Le calcul de la distance entre un animal et le troupeau a été implémenté de trois manières différentes :

- Par rapport à l'animal le plus proche.
- Par rapport au barycentre des N animaux les plus proches.
- Par rapport à un animal particulier du troupeau, fixé dans un fichier de configuration, ce qui permet de prendre en compte les affinités entre deux ruminants (par exemple mère et jeune).

On notera par la suite P le point à partir duquel la distance est calculée.

Le test d'éloignement s'effectue en quatre étapes et utilise une fonction de motivation à se déplacer qui dépend de trois paramètres : d1Troupeau, d2Troupeau et d3Troupeau. Ces valeurs, ainsi que la méthode de calcul de la distance, seront fixées dans le fichier troupeau.cfg.

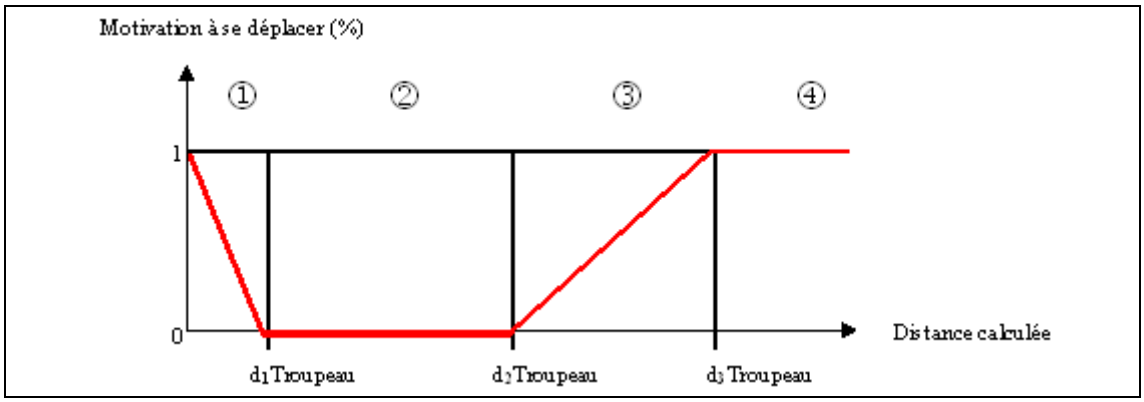

Figure 5 : Motivation à se déplacer en fonction de la distance [PORTAL 2003]

On calcule donc dans un premier temps la distance entre l'animal et son congénère le plus proche. Si elle est inférieure à d1Troupeau, alors l'animal est trop proche, on calcule donc sa motivation à se déplacer et on réalise un tirage aléatoire afin de savoir si l'animal se déplace.

Si l'animal n'est pas trop proche, on réalise un nouveau calcul de distance (dtroupeau) entre l'animal et le troupeau. Si cette distance est comprise entre d1Troupeau et d2Troupeau, on considère alors que l'animal est bien placé et qu'il ne cherche pas à se déplacer. Si en revanche dtroupeau est supérieur à d3Troupeau, alors l'animal cherche à se rapprocher de ses congénères. Sinon, on a dtroupeau compris

entre d2Troupeau et d3Troupeau et dans ce cas, on calcule sa motivation à se déplacer puis on réalise un tirage aléatoire pour savoir si l'animal va se rapprocher.

Tous ces déplacements se font tête haute et l'animal interrompt toute activité précédemment commencée. Il ne mémorisera pas les cellules parcourues au cours de ce trajet et ne choisira une nouvelle activité que lorsqu'il sera arrivé à destination.

1.2.3.2.1.2 L'éloignement

On peut voir sur la figure 6 le champ de vision type d'un animal se situant à la case 0. Lorsque l'animal veut s'éloigner, il va choisir l'une des deux cellules les plus éloignées de lui, c'est-à-dire la cellule 9 ou la cellule 15. Il va donc choisir entre ces deux dernières en fonction de la position de l'animal le plus proche et mettre sa direction à jour.

Dans le cas où l'animal se trouverait sur un des bords de la parcelle, il prend comme direction l'inverse de la direction trouvée et choisit à nouveau une cellule cible.

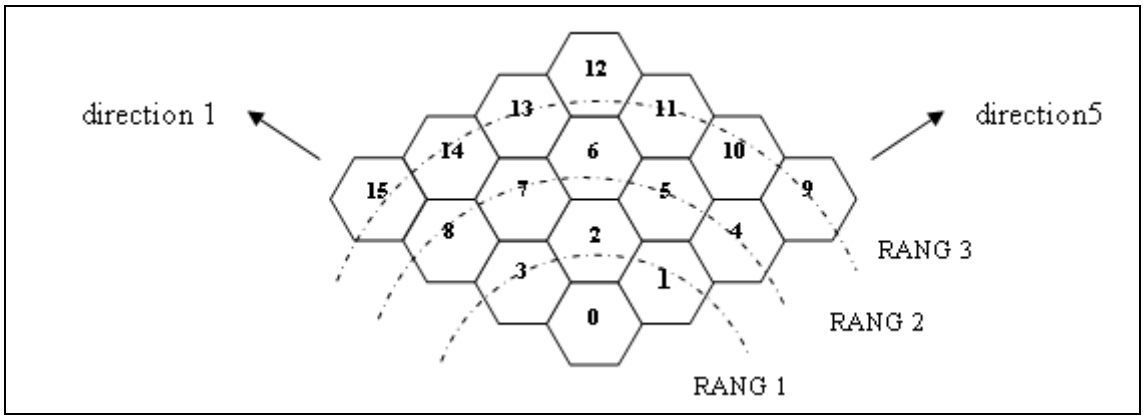

Figure 6 : Choix d'une cellule lors de l'éloignement [PORTAL 2003]

1.2.3.2.1.3 Le rapprochement

On détermine tout d'abord la cellule la plus proche de l'animal se situant dans le cercle de centre P et de rayon d2Troupeau. L'animal va alors se diriger vers cette cellule, et ce après avoir mis à jour sa direction.

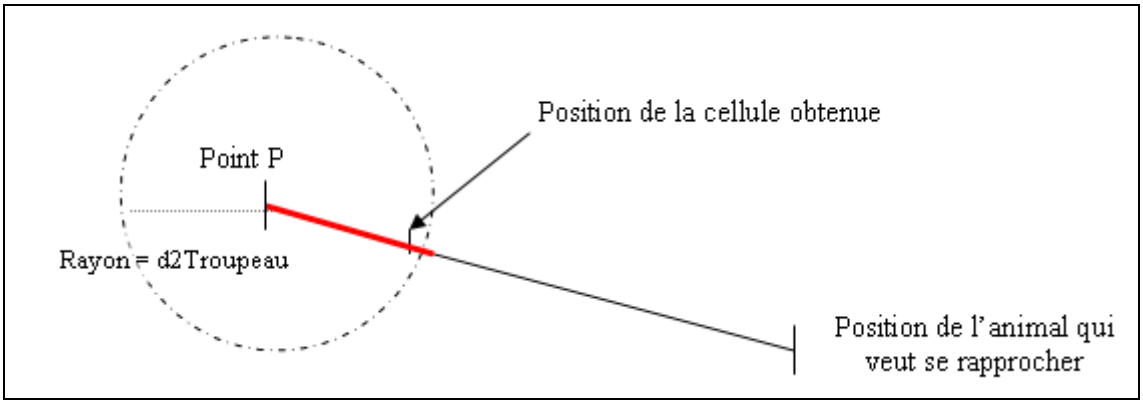

Figure 7 : Rapprochement de l'animal par rapport au troupeau [PORTAL 2003]

1.2.3.2.2 Le leadership

Chaque animal possède une probabilité non nulle, fixée en début de simulation de devenir le leader. A un instant donné il ne peut y avoir qu'un seul leader dans le troupeau.

Dans le prototype actuel, chaque ruminant a également une probabilité d'être satisfait. Dans le cas où il ne l'est pas, il réalise un tirage de leadership. Si ce dernier est « réussi », il se dirige alors vers une cellule dont les coordonnées ont été placées en « dur » dans le code. L'exploration et la mémoire n'interviennent donc pas dans ce processus.

1.2.3.2.3 La mémoire

Deux méthodes ont été à ce jour implémentées mais ne sont pas intégrées au simulateur [MASSON 2001]. Des essais ont montré qu'elles sont équivalentes en terme de mémoire occupée et de temps de réponse durant la phase de mémorisation. Dans les deux types de mémoires, les principes restent les mêmes. Lorsque l'animal défolie une cellule, il mémorise, sur elle ainsi qu'autour, sa qualité, résultat d'une fonction de perception. Un ensemble de cellules juxtaposées d'une même qualité seront agrégées en un site mémoriel. Ce dernier ne sera désagrégé en deux sites distincts que lorsque deux groupes de cellules auront leurs qualités séparées d'un certain seuil.

En ce qui concerne la fonction d'oubli, elle fait converger la qualité de tous les sites de n% vers leur moyenne. Ainsi, si un site a une qualité de 1 et un autre une de 3, que l'utilisateur a fixé un pourcentage d'oubli de 10%, alors le premier site aura pour valeur de qualité après l'oubli 1.1 et le second 2.7.

De plus, lorsqu'un site est mémorisé et que la capacité de la mémoire est dépassée (cette capacité est bien sûr elle aussi paramétrable), on retire alors le plus mauvais site de la mémoire et on le remplace par celui à mémoriser.

1.2.3.2.3.1 Méthode des contours

Cette méthode consiste à mémoriser la trajectoire de l'animal durant une phase de défoliation élargie à n cases, n étant paramétrable, ce qui représente son champ de vision lorsqu'il pâture. Le floue ainsi créé se situe de part et d'autre de l'animal, à égal distance. La figure 8 montre un exemple de mémorisation grâce à cette méthode. La trajectoire du ruminant est représentée par le trait noir et les cellules défoliées par un point. La surface élémentaire mémorisée par l'animal à chaque défoliation est représentée en haut à gauche.

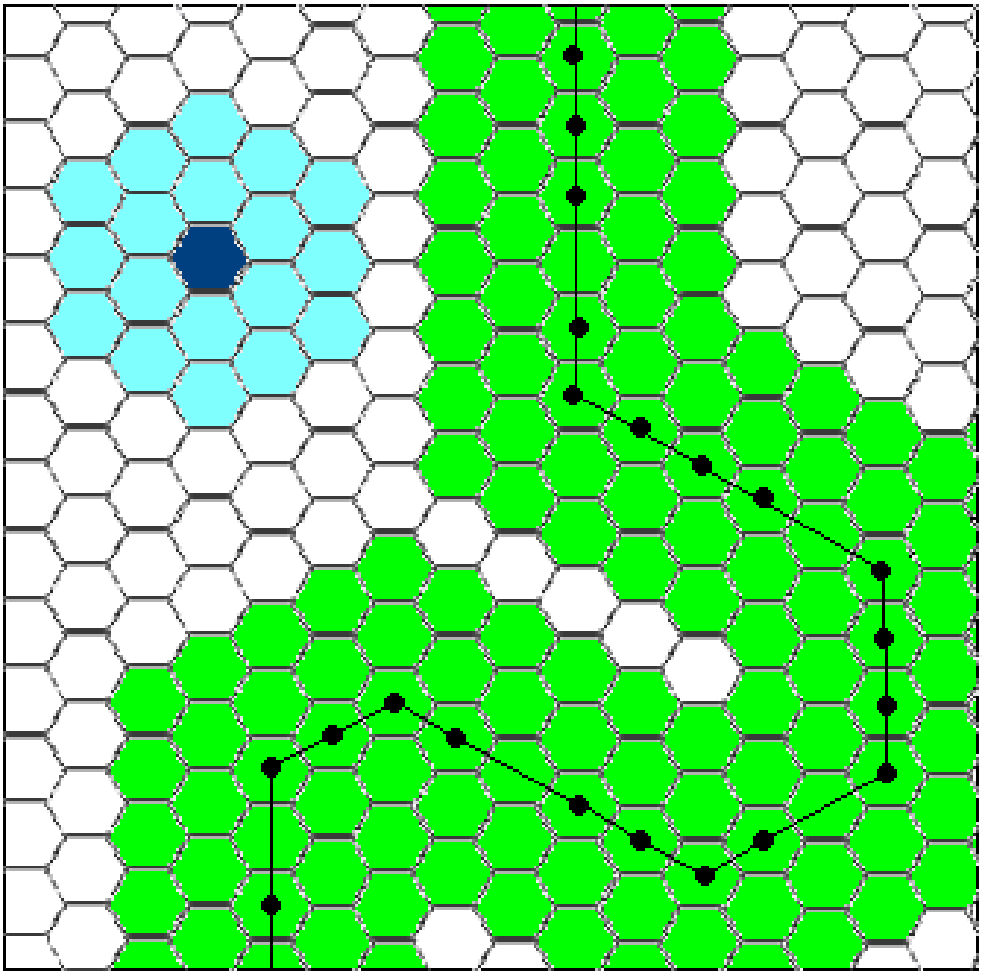

Figure 8 : Exemple de mémorisation par la méthode des contours

La parcelle étant représentée comme une matrice de cellule, la structure de données suivante a été choisie :

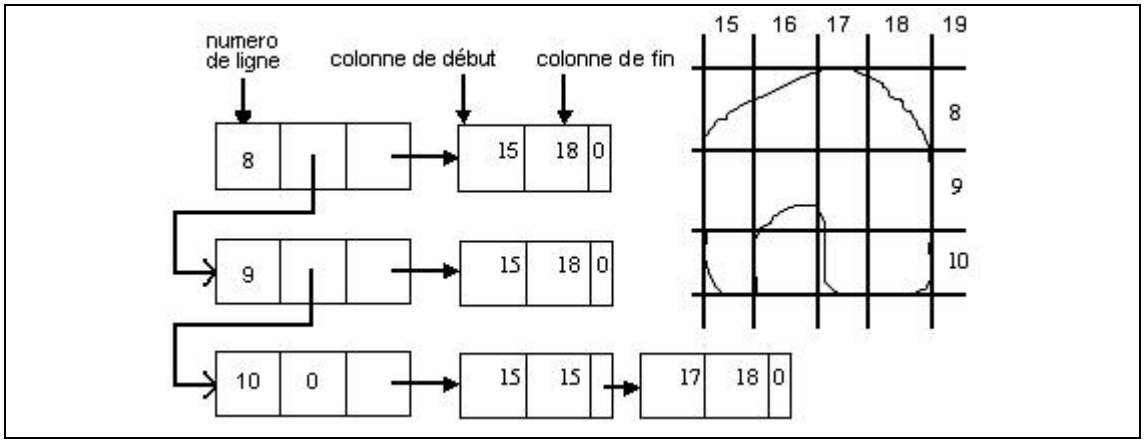

Figure 9 : SDD utilisée dans la méthode des contours

La figure suivante représente le diagramme de classe correspondant à la méthode des contours.

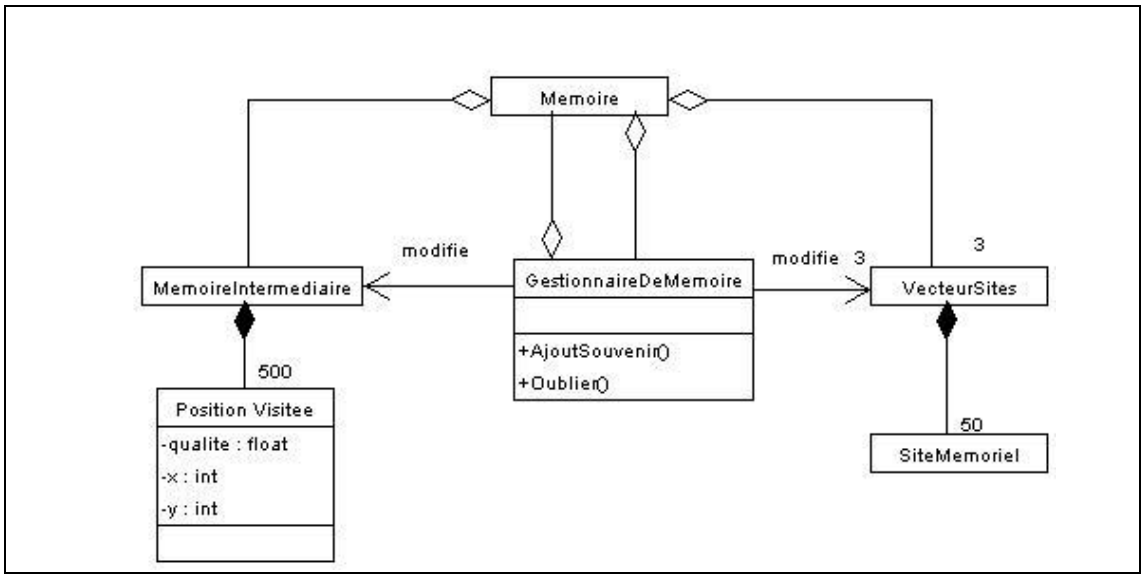

Figure 10 : Diagramme de classe de la mémoire spatiale

On peut donc voir que la mémoire contient trois vecteurs de sites mémoriels correspondant aux trois types de sites mémoriels : alimentaire, buvée et repos. Ces deux derniers ne sont pas utilisés et ont juste été implémentés dans un but prévisionnel.

Le gestionnaire de mémoire contient toutes les méthodes utiles, comme son nom l'indique, à la gestion de la mémoire.

Enfin, la mémoire intermédiaire permet de stocker les informations durant la mémorisation d'un site alimentaire et ce avant la mise à jour de la mémoire. Chaque cellule connue par l'animal est stockée comme une PositionVisitee, cette dernière classe contenant toutes les informations utiles (direction, coordonnées). La « mise en mémoire » se déroule de cette manière : lorsque l'animal arrive sur une cellule à défolier, on teste s'il a changé de site végétal en comparant la qualité de la dernière cellule défoliée à celle de la cellule courante. Si la différence franchit un certain seuil, on crée un nouveau site grâce aux informations stockées dans la mémoire intermédiaire en supprimant, si la mémoire est pleine, le site de plus mauvaise qualité ; puis on vide la mémoire intermédiaire. On ajoute ensuite la cellule courante aux PositionVisitee.

Après un ajout de site, on compare les sites entre eux afin d'effectuer le cas échéant une agrégation ou une désagrégation. Pour cela, on compare tout d'abord si le site mémoriel nouvellement créé recouvre un autre site en mémoire. Si c'est le cas, on compare leurs qualités. Si la différence des deux dépasse un certain seuil, alors on retire de l'ancien site la surface recouverte par le nouveau. Sinon on agrège les deux sites. Le nouveau site a alors comme nouvelle qualité la moyenne des qualités des deux sites, pondérée respectivement par leur surface.

1.2.3.2.3.2 Méthode de la carte pixellisée

Cette méthode utilise quant à elle un autre type de structure de données. En effet, elle utilise une carte pixellisée, contenant des macro-cellules de nxn cellules, pour représenter le contenu de la mémoire. Cette carte est initialisée à la création de la mémoire. Ainsi, chaque macro-cellule regroupe un unique ensemble de cellules qui ne variera plus au cours du temps. Le flou est donc créé par approximation des macro-cellules.

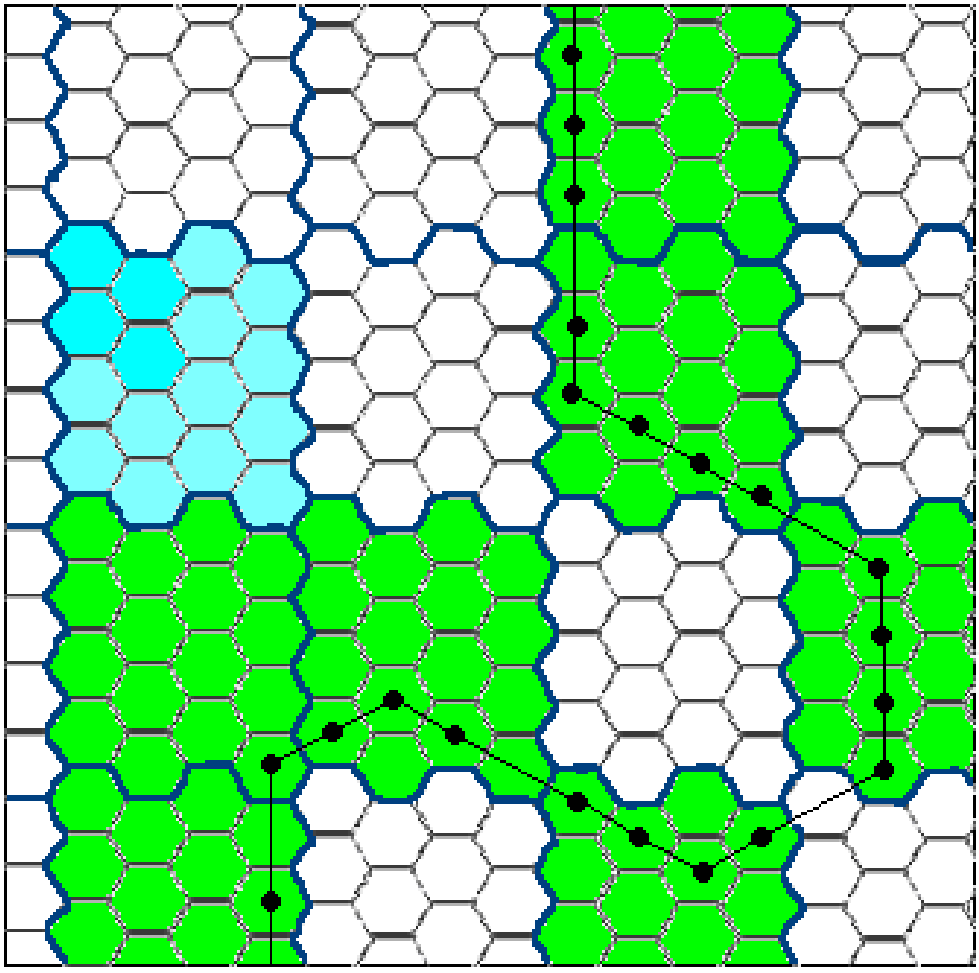

Figure 11 : Exemple de mémorisation par la méthode par pixellisation

Cela entraîne bien sûr une SDD différente de la précédente : les macro-cellules de la carte pixellisée pointent sur les sites mémoriels leur correspondant, et, réciproquement, chaque site pointe, via une liste chaînée, sur les macro-cellules qu'il comporte. Cette solution a été retenue afin de faciliter les recherches en cas d'agrégation.

Le fonctionnement de cette méthode se rapproche tout de même fortement de la méthode des contours. En effet, la mémoire intermédiaire est mise à jour lors de chaque défoliation. La seule différence réside dans le fait que, lorsque l'animal quitte un site végétal, on traite indépendamment chaque macro-cellule selon le même processus que pour l'autre méthode.

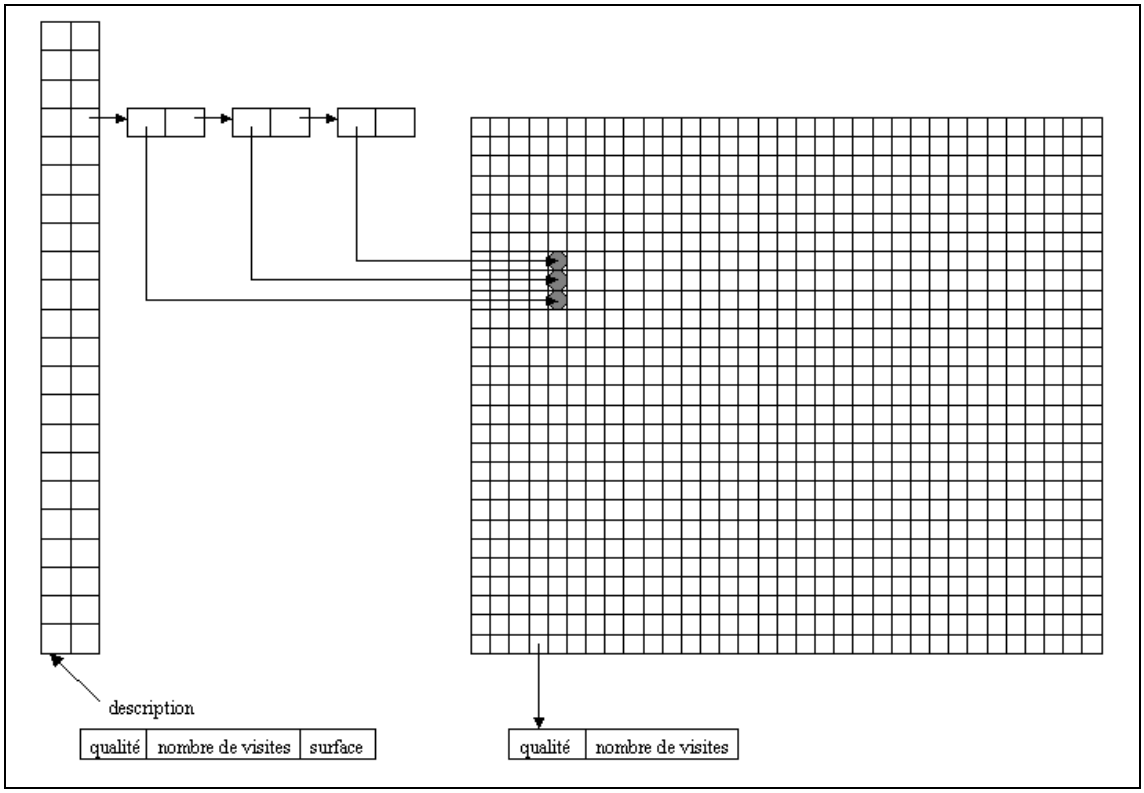

Figure 12 : SDD utilisée dans la méthode par pixellisation

1.2.3.2.3.3 Comparaison visuelle des deux méthodes

Le test suivant a été réalisé avec une petite parcelle contenant un site en forme de U et une trajectoire permettant de sillonner l'ensemble de la carte. On s'aperçoit qu'avec l'une ou l'autres des deux méthodes, le U apparaît bien, bien que légèrement déformé. Ceci est normal puisqu'en aucun cas la mémoire ne doit être parfaite. Ces déformations sont donc le résultat du flou dû, dans le cas de la méthode des contours, à l'imprécision (n dans l'explication du 1.2.3.2.3.1), et dans le cas de la méthode par pixellisation à la structure même des macro-cellules. On peut tout de même constater que la qualité de mémorisation est quasi semblable et que seule diffère la forme, qui, dans le cas de la carte pixellisée, a subi un effet « cube ».

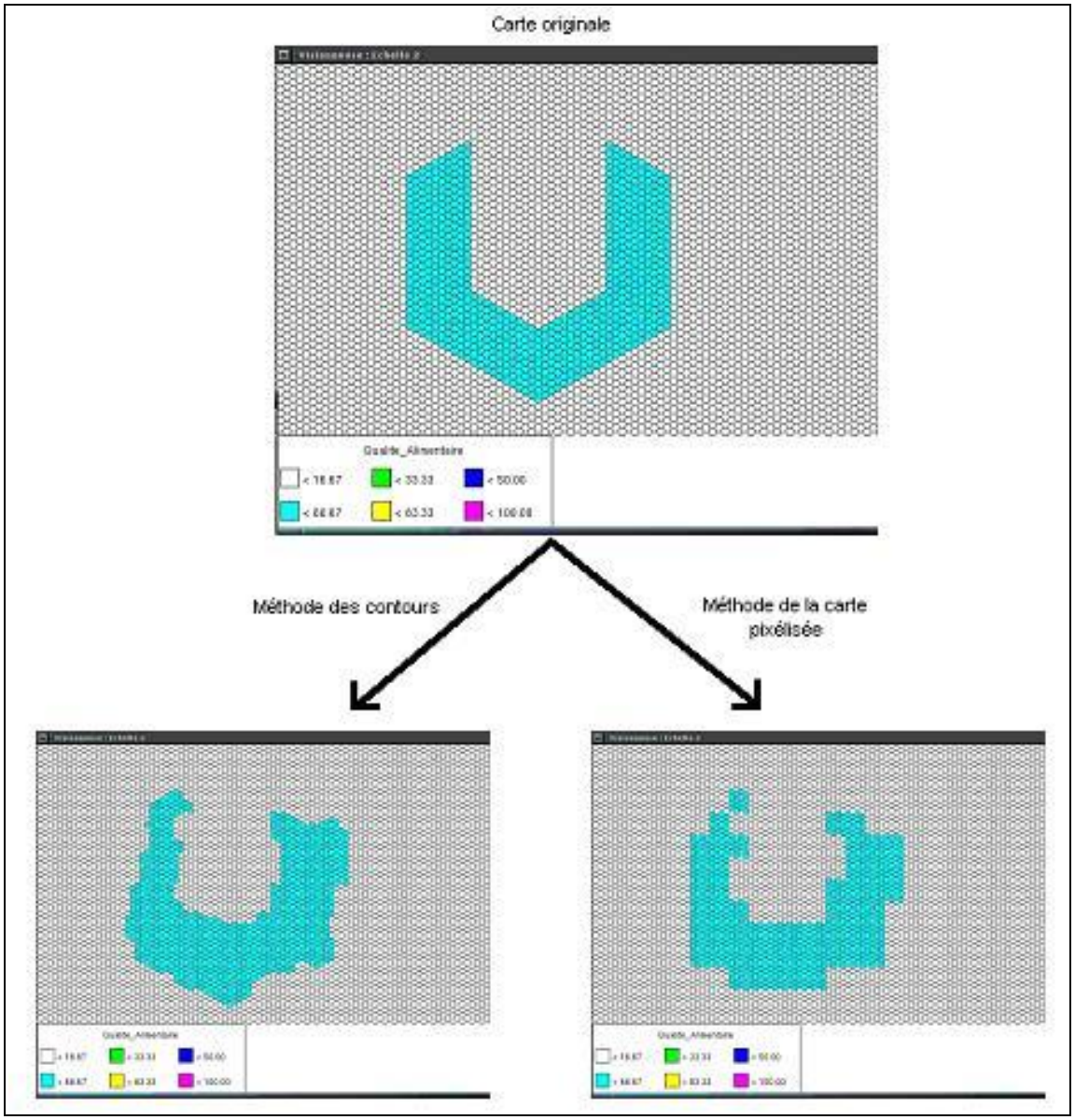

Figure 13 : Comparaison des méthodes de mémorisation

1.2.4 Le modèle technique

1.2.4.1 Description

Ce module regroupe toutes les classes et autres programmes qui ne trouveraient pas leur place dans les différents sous-modèles.

1.2.4.2 Etat d'avancement

1.2.4.2.1 L'interface utilisateur

L'interface utilisateur a été développée récemment afin de faciliter l'utilisation du simulateur [PORTAL et SEGUY 2003]. Elle a été implémentée en JAVA et permet de gérer les entrées et les sorties du simulateur via une succession d'étapes. Elle donne aussi la possibilité à l'utilisateur d'effectuer des réplications et permet évidemment de lancer le simulateur.

1.2.4.2.2 La visionneuse

La visionneuse est un outil graphique permettant de visualiser la parcelle ainsi que les déplacements des animaux et du troupeau [MASSON et VILLEGER, 2001]. Elle comporte trois échelles :

- L'échelle 1, ou vue détaillée, permet d'observer les quinze cellules du champ de vision d'un animal ainsi que leurs différentes valeurs.
- L'échelle 2, ou vue intermédiaire, permet d'observer les déplacements de l'animal sur une partie de la parcelle.
- L'échelle 3, ou vue globale, permet d'observer l'évolution du troupeau au sein de la parcelle.

Toutefois, de par les nombreuses modifications qu'ont subi les fichiers de sorties du simulateur, cet outil ne fonctionne plus avec ceux-ci. Elle permet de visionner les cartes générées par le générateur détaillé ci-après ainsi que les trajectoires créées par le générateur du même nom.

1.2.4.2.3 L'échéancier

L'échéancier permet de discrétiser la gestion des événements, car la fréquence des actions des animaux est à une échelle totalement différente de celle des végétaux [GUERRY 2000]. Ainsi, l'unité de l'échéancier est la seconde mais la découpe du temps se fait en fonction des actions des entités : chaque entité peut donc être considérée comme un objet subissant divers événements qui entraîneront d'autres événements et ainsi de suite…

La figure 14 illustre l'implémentation de l'échéancier :

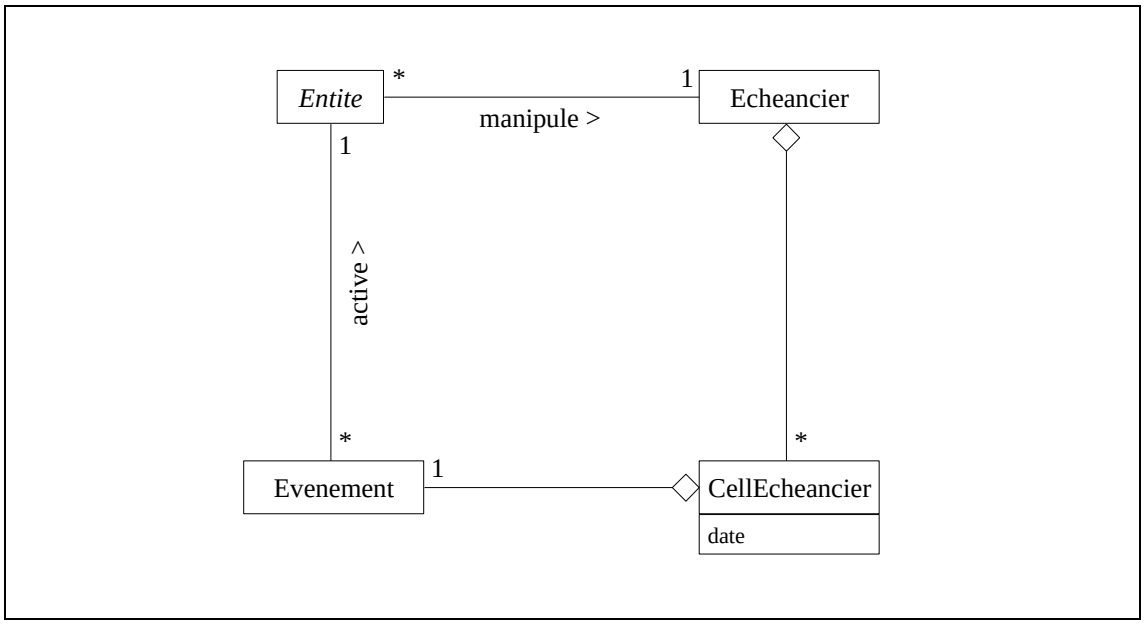

Figure 14 : Diagramme de classe de l'échéancier

On peut donc voir que l'échéancier contient une liste d'événements, chacun étant activé à une date donnée. Ainsi, la classe CellEcheancier contient un événement ainsi qu'une date d'activation. Lorsqu'un événement est exécuté, on active l'entité associée qui va déclencher une action en accord avec le type de l'événement. Enfin, chaque entité peut évidemment manipuler l'échéancier en y insérant ou supprimant des événements qui la concernent. On en déduit donc que tout objet qui devra manipuler des événements devra dériver de Entité. Ainsi cette dernière est la classe mère des classes Ruminant, Troupeau et Simulateur.

Les différents événements implémentés dans le simulateur sont les suivants :

- Choix activité (Choix d'une activité grâce à l'automate décisionnel)
- Choix cellule (Choix d'une cellule dans les 15 du champ de vision de l'animal)
- Fin activité (Fin d'une activité suite à l'arrêt « normal » d'une activité)
- Fin déplacement (Fin d'une séquence de déplacement)
- Force fin activité (Fin d'une activité suite au leadership)
- Maj position (Mise à jour des coordonnées du ruminant)
- Test éloignement leader (Test l'éloignement par rapport au leader)
- Test éloignement troupeau (Test l'éloignement par rapport au reste du troupeau)

1.2.5 Le prototype actuel

Le prototype actuel regroupe pour l'instant la quasi-totalité des fonctions du modèle végétal ainsi que celles du modèle animal. Il contient de plus l'échéancier ainsi que l'automate décisionnel. La mémoire spatiale n'a, quant à elle, pas encore été ajoutée au simulateur. Le grégarisme et le leadership sont gérés. L'interface utilisateur permet de configurer correctement la simulation en adéquation avec les différentes fonctionnalités du simulateur.

Les tests effectués précédemment donnent des résultats cohérents mais cette version est toujours en attente de validation.

Partie 2 : Modifications apportées au simulateur

2.1 Compréhension et modification initiale du simulateur

Le début de mon travail a consisté à me familiariser avec le code du simulateur afin de mieux appréhender les futures modifications nécessaires et d'avoir les bases indispensables à l'adjonction de la mémoire dans le dernier prototype. Lors de cette première lecture, j'ai pu faire plusieurs constats :

- hétérogénéité du code provoquée par la multiplicité de projets et de stages nécessaires à l'élaboration du simulateur.
- portions de code obscures du fait du manque de commentaires.
- présence de fonctions et de fractions de programmes non utilisées.

Le but de mon stage étant de produire à son terme une première version complète, j'ai décidé de mettre en place une documentation exhaustive, pratique et non invasive afin qu'elle ne puisse pas apporter son lot d'erreurs au sein d'un programme déjà complexe. De manière à générer automatiquement ce document, j'ai choisi d'utiliser l'utilitaire Doxygen, dont le formalisme est donné en annexe, associé au logiciel Graphviz qui permet de créer des diagrammes UML automatiquement.

Il a donc fallu dans un premier temps compléter la documentation de toutes les classes, les méthodes, etc. Cela m'a permis de mieux cerner le simulateur dans son ensemble et, ce faisant, d'homogénéiser quelque peu le code. En effet, j'ai sorti toutes les méthodes inline de la définition de la classe, ce qui permet d'en faciliter la lecture, j'ai modifié les noms de certains arguments de manière à suivre le guide style maintenant imposé et, par-là même, d'assurer leur transparence. J'ai aussi revu l'indentation afin que l'aspect du code soit identique indépendamment de l'éditeur utilisé en supprimant par exemple les tabulations.

Le résultat obtenu est une documentation HTML de plus de 1200 pages, à la navigation intuitive. Doxygen peut aussi sortir la documentation sous d'autres formats, tel que le RTF, afin de faciliter son impression.

Toutefois, ce document n'a en aucun cas pour but d'être parfait. Il doit juste servir de support et demande à être en constante évolution. Ainsi, certaines variables n'ont trouvé aucun commentaire explicite et demanderont à être renseignées au fur et à mesure de l'avancement du simulateur.

2.2 Analyse des modèles animal et social

Durant l'élaboration de cette documentation, j'ai réalisé de nombreux diagrammes UML d'activités, afin, dans un premier temps de vérifier si la structure du simulateur est en adéquation avec les modèles conceptuels émis, puis dans un second temps de les utiliser comme base de travail pour des futures modifications comme l'insertion de la mémoire. Ce sont d'ailleurs ces diagrammes qui ont permis de s'apercevoir que l'action de rumination était mal réalisé, ce qui a débouché par une modification de l'automate décisionnel par Benjamin ASTRE [ASTRE 2004]. Ces diagrammes sont au nombre de 14 et sont tous présentée dans l'annexe B. La figure suivante n'a donc lieu que d'exemple et permet d'avoir une vue globale du simulateur.

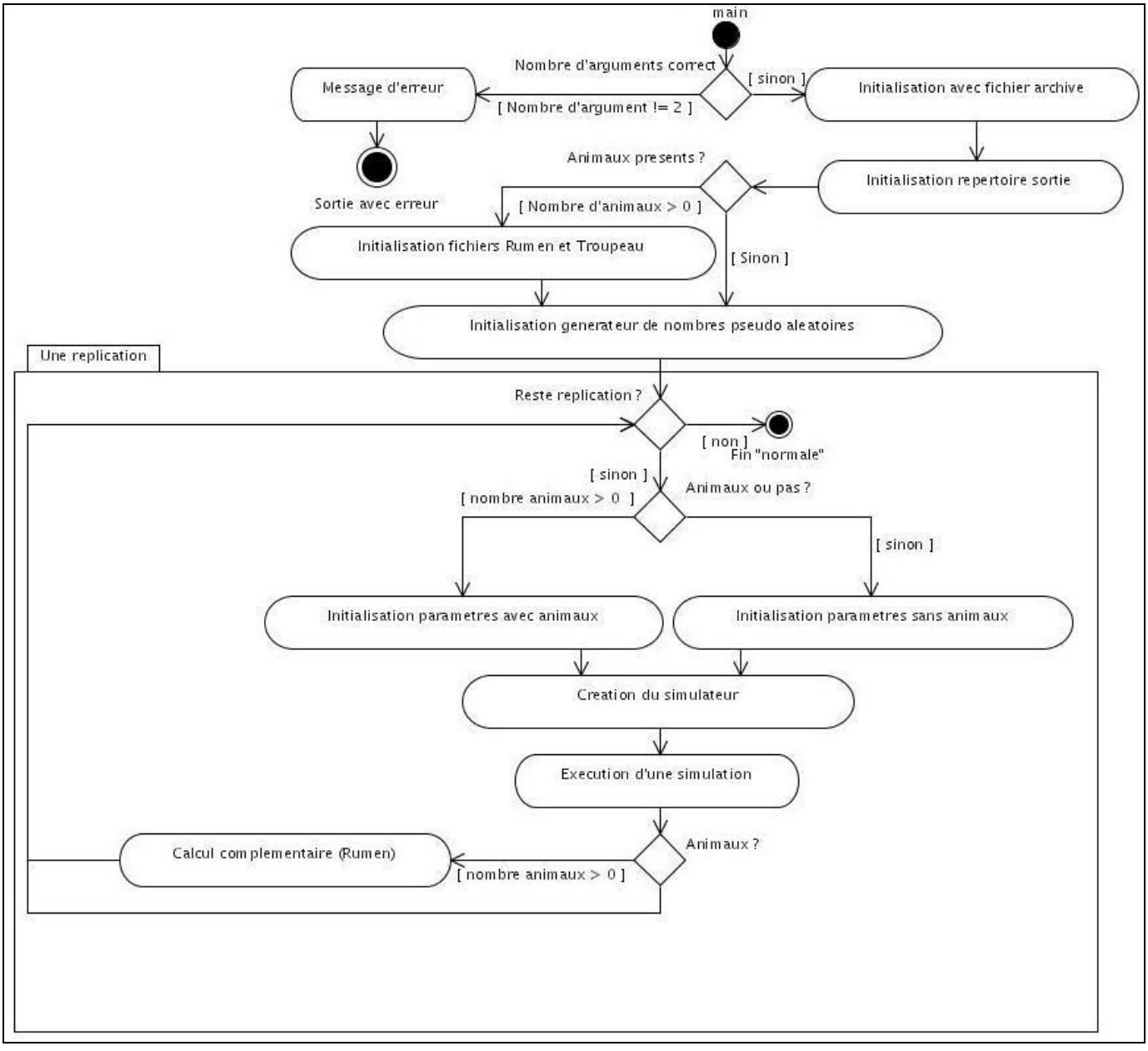

Figure 15 : Diagramme d'activité représentant une vue générale du simulateur

2.3 Modifications apportées au code

De nombreuses modifications ont été apportées lors de l'élaboration de la documentation. Le but de cette partie n'est pas de faire un listing complet de ces dernières, la plupart étant mineures, mais plutôt de présenter dans sa globalité le travail effectué.

Le code a tout d'abord été modifié dans sa forme. Je ne reviendrai pas sur ce point, étant donné qu'il a été exposé plus en détail dans la partie précédente.

Le code a aussi été optimisé. Ces optimisations pouvaient être aussi bien mineures, comme le fait de remplacer les <variables>++ par des ++<variables>, que plus importantes comme lorsqu'à l'intérieur de boucles itératives, des tests étaient effectués sur des variables compteurs. Cela augmentait significativement les temps de réponse puisque ces derniers concernaient les cellules qui, pour rappel, se chiffrent en millions. Ainsi, ces boucles ont été séparées en deux, ce qui permet de traiter en dehors le cas particulier. Par exemple, on pouvait trouver :

```
Pour i = 1 à n faire
  Si i = k alors
    Opération 1
  Sinon
    Opération 2
  Fin si
Fin pour

J'ai remplacé cela par (avec k ≤ n) :

Pour i=1 à k-1 faire
  Opération 2
Fin pour
++i
Opération 1
Pour i=k+1 à n faire
  Opération 2
Fin pour
```

Figure 13 : Exemple de modification d'une boucle

De nombreuses autres améliorations ont pris place, comme le fait de passer inline les méthodes de tailles minimes (accesseurs, destructeurs, la majorité des constructeurs ...).

Toutefois certains facteurs ont limité cette optimisation. Par exemple, de nombreuses inclusions sont dites embrassées. Une classe A inclut une classe B qui inclut cette dernière. Il en résulte qu'une des deux classes ne peut avoir dans son fichier d'en-tête des méthodes concernant l'autre. Cela a empêché la mise en inline de certaines méthodes.

Enfin, j'ai apporté quelques corrections. Ainsi, le barycentre entre N animaux n'était en réalité calculé qu'entre les N-1 premiers, ce qui, à l'échelle d'un troupeau, n'était pas perceptible, mais qui, lorsque le groupe de ruminants n'était composé que de deux individus, pouvait poser des problèmes.

Les makefile ont tous aussi été modifiés. En effet, un système de dépendance automatique avait été mis en place. Or, dans le simulateur actuel, ce dernier était devenu obsolète et ne servait donc plus. Les makefile ont donc été purgés afin de devenir plus compréhensibles. Ainsi la commande « make dep » ne sert plus à créer les dépendances mais juste à nettoyer les dossiers des *.o et autres ~*.

Une fois ces modifications apportées, j'ai commencé à analyser l'insertion de la mémoire dans le prototype existant. Tout d'abord j'ai commencé à me préoccuper des futurs tests que je devrais effectuer pour vérifier les résultats. Ceux-ci seront surtout basés sur une analyse visuelle de la mémoire des animaux, de leurs préférences, et de leurs trajectoires. Ces besoins ainsi que des modifications apportées dernièrement au simulateur ont conduit à modifier la visionneuse.

Partie 3 : Modifications de la visionneuse et création de l'interface associée

3.1 Adaptation de la visionneuse aux sorties du simulateur

3.1.1 Besoins et analyse

Toutes les sorties du simulateur sont maintenant au format csv, avec comme séparateur le « ; ». Or la visionneuse est conçue pour lire des fichiers prenant comme séparateur des espaces. Ce type de fichier était bien généré par le générateur de carte mais ce dernier étant maintenant obsolète, il faut adapter la visionneuse à ce type de sortie.

De plus, la visionneuse lit les valeurs ligne par ligne. Ainsi, pour une parcelle de 500 par 500 cellules, le fichier qu'elle acceptait était du type 500 lignes et 500 colonnes, chacune des cases « correspondant » à la valeur de la cellule (biomasse en Vv, hauteur en Vv, etc.). C'est cette valeur que va afficher la visionneuse. Ce type de fichier est là encore devenu caduc, puisque généré uniquement par le générateur de carte. Les fichiers de sorties du simulateur exploitables sont ceux de mises à jour de la végétation, qui représentent, à la fin de chaque journée, l'état de chaque cellule. Ces derniers sont d'un type différent. En effet, chacune des lignes représente une cellule, avec l'abscisse en première colonne, puis l'ordonnée, le numéro du site et ainsi de suite pour chacune des autres variables végétales choisie par l'utilisateur dans l'interface. Il faut donc là encore adapter la visionneuse, tout d'abord en modifiant la façon de parcourir le fichier, puisqu'une seule valeur doit maintenant être sélectionnée pour chaque ligne, puis en permettant à l'utilisateur de choisir le type de valeur à afficher.

La visionneuse étant un outil de vérification et d'analyse des résultats, elle doit permettre de vérifier la cohésion sociale. Or, dans l'état actuel des choses, une seule trajectoire d'une seule journée peut être affichée à la fois. Cela implique que l'utilisateur ne peut pas se faire une idée de la trajectoire de l'ensemble d'un troupeau, ni afficher la trajectoire sur plusieurs jours d'un ruminant. Il va donc falloir permettre à l'utilisateur de réaliser ceci. De plus les fichiers de trajectoires sont là encore inexploitables. En effet, les séparateurs sont devenus des « ; » et une nouvelle colonne est apparue : la collision. Cette dernière permet d'indiquer que l'animal ne s'est pas réellement rendu sur une cellule car elle était occupée, mais a fait un nouveau choix dans les cellules adjacentes. Il est décidé d'ignorer simplement cette colonne afin de ne pas surcharger inutilement ni la visionneuse, ni le simulateur.

Enfin, et ce dans un souci d'uniformisation et de cohérence entre tous les fichiers d'entrées, le fichier de légende ainsi que sa lecture ont aussi été modifiés pour que ce dernier soit du type cfg, type utilisé dans le reste du simulateur pour les fichiers de configuration.

3.1.2 Implémentation

La visionneuse étant elle aussi le résultat de multiples stages, les mêmes problèmes que ceux observés dans le simulateur étaient présents (absence de commentaire, hétérogénéité du codage,…). J'ai donc mis en place une documentation en Doxygen allégée, car beaucoup moins détaillée. Ceci m'a permis de me familiariser avec le code et de trouver ainsi les portions de code à modifier, tout en permettant de rendre les futures retouches de manière plus aisée. Les séparateurs ont donc été remplacés dans le même temps.

Dans un second temps, la lecture du fichier de mise à jour a elle aussi été modifiée. Le programme « lit » chaque ligne en sélectionnant la i^{eme} valeur, i correspondant au numéro de la colonne contenant les valeurs que souhaite visualiser l'utilisateur. Il affiche alors cette dernière. Cette modification n'est en fait pas très lourde puisque les changements se situent dans la boucle principale de lecture.

Enfin, en ce qui concerne l'affichage des trajectoires journalières, il suffit de placer, au lieu d'un unique nom de fichier de type « .traj », autant de noms de fichiers que l'on souhaite. L'ajout d'une boucle globale permet de réaliser autant d'itérations qu'il y a de fichiers, ce qui permet de tracer les trajectoires souhaitées, en les coloriant selon un dégradé du bleu vers le rouge. La figure 16 représente le résultat après ces modifications :

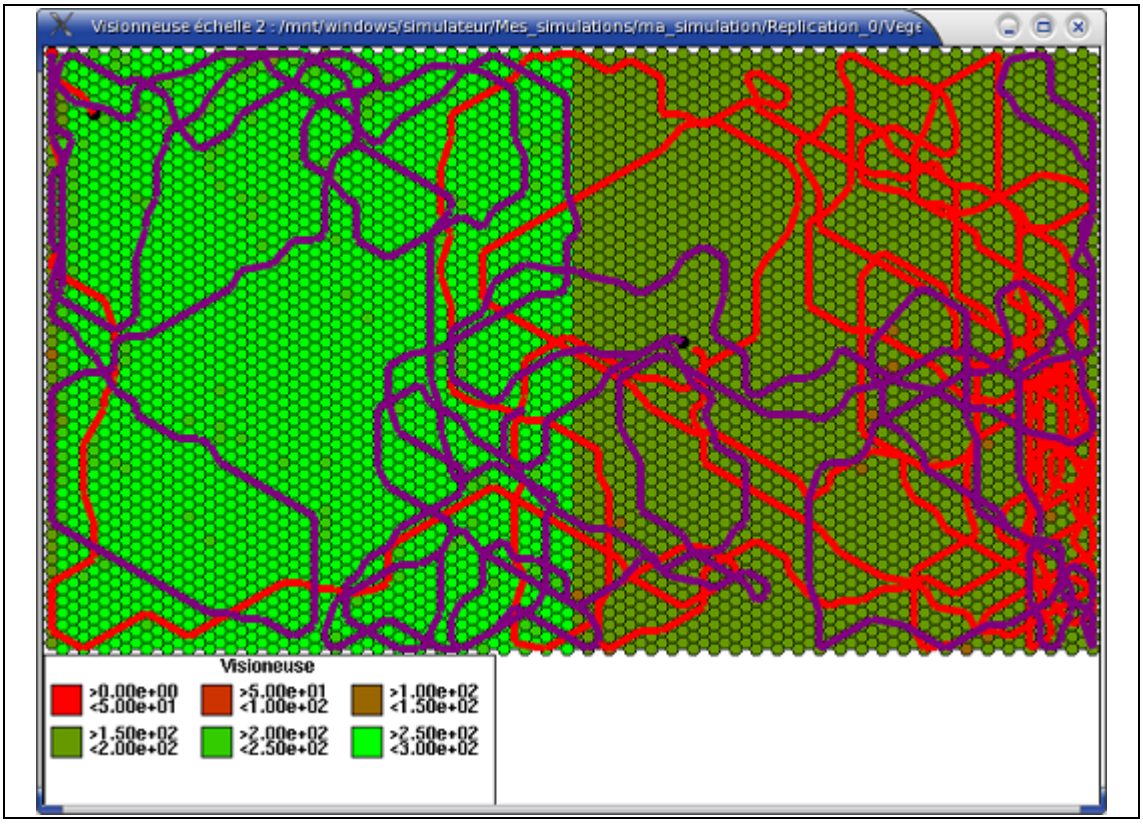

Figure 16 : Un exemple de vue intermédiaire avec deux trajectoires

3.2 Création de l'interface associée

3.2.1 Besoins et analyse

Toutes ces modifications réalisées permettent à la visionneuse d'être effective mais impliquent par-là même une plus grande complexité de la commande pour lancer cette dernière. De plus, pour la lancer à plusieurs reprises, il faut à chaque fois saisir l'ensemble des noms fichiers, vérifier que ces derniers sont corrects, etc.

Afin de rendre la visionneuse plus intuitive pour le futur utilisateur, j'ai décidé de créer une interface, basée sur le même modèle que celle existant pour le simulateur, possédant bien sûr moins d'étapes mais permettant de la configurer entièrement. La figure suivante présente ce qu'elle doit permettre :

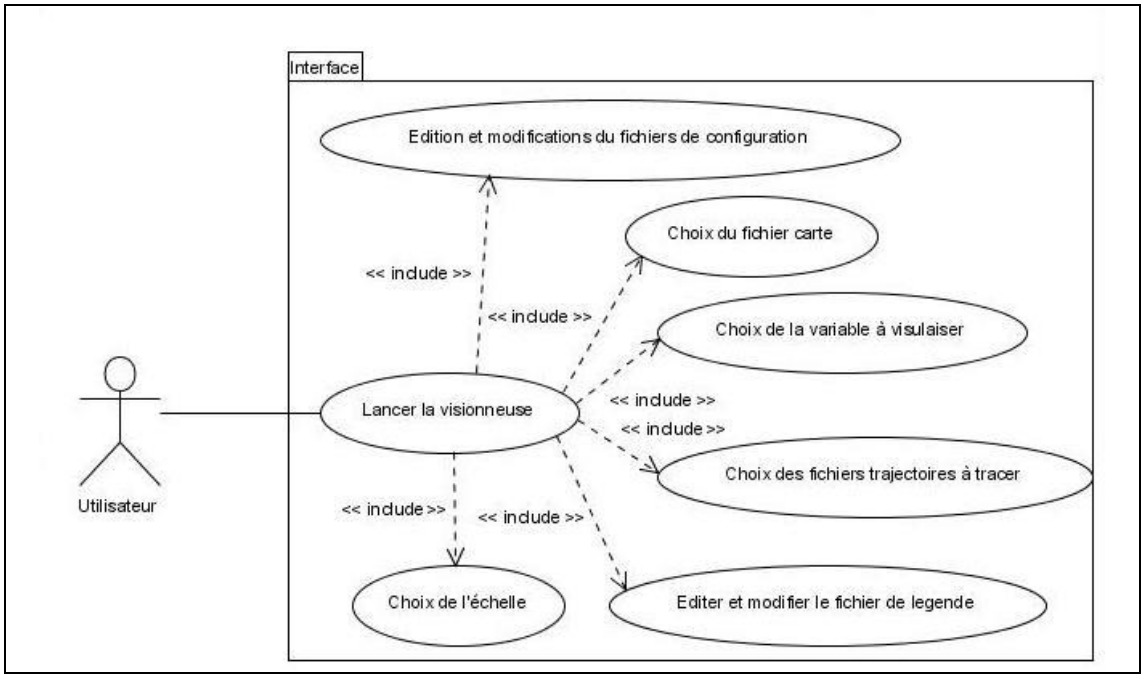

Figure 17 : Cas d'utilisation de l'interface de la visionneuse

3.2.2 Implémentation

L'interface a été réalisée en utilisant le patron MVC [PORTAL et SEGUY 2003]. Ce choix se justifie par le fait que ce patron a été de nombreuses fois testé et approuvé. De plus, le code généré est intelligible et permet de dissocier la partie purement graphique des traitements. L'interface est divisée en trois parties :

- la partie de l'interface utilisateur qui affiche des informations à propos du modèle, appelée « vue ».
- la partie de l'interface utilisateur qui permet d'agir sur le modèle, nommée « contrôleur ».

- le « modèle », ou « métier », qui constitue le cœur de l'application et contient les données et les traitements. Lorsque des changements interviennent au niveau du modèle, la vue est mise à jour par celui-ci.

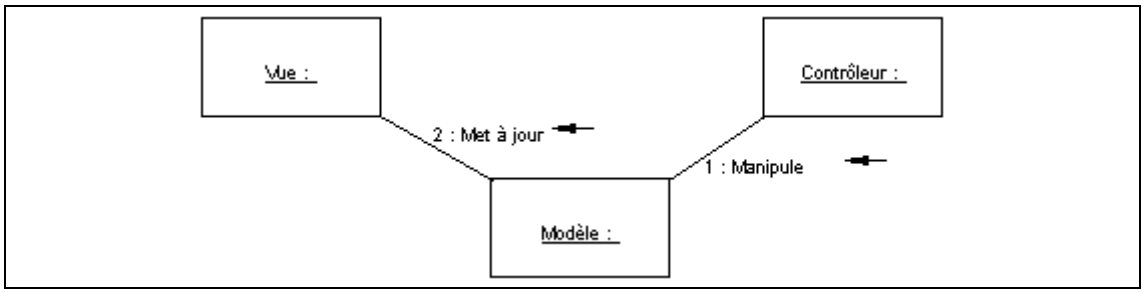

Figure 18 : Diagramme de collaboration du patron MVC

Toutefois, dans leur implémentation de ce patron, Séverine PORTAL et Romain SEGUY ont décidé de lier le contrôleur et la vue dans un but de simplification du code généré.

Ainsi, les classes de l'interface correspondant aux vues et aux contrôles se nomment toutes Vue suffixée d'un nom décrivant l'étape à valider par le biais de cette vue ; ces classes dérivent toutes de la classe abstraite VueEtape.

Les classes de l'interface correspondant aux modèles se nomment toutes Etape, suivi du nom décrivant l'étape et héritent toutes de la classe abstraite Etape.

A chaque vue correspond une unique étape : par exemple, à l'étape EtapeFichierTrajectoire correspond la vue VueFichierTrajectoire.

L'interface a été décomposée selon les paquetages suivants :

- le paquetage fr.inra.clermont.visionneuse regroupe les classes principales de l'interface et toutes les classes qui ne sont pas susceptibles de figurer dans les autres paquetages décrits ci-dessous et qui ne justifient pas la création d'un paquetage particulier.
- le paquetage fr.inra.clermont.visionneuse.configuration regroupe toutes les classes nécessaires au paramétrage de l'interface.
- le paquetage fr.inra.clermont.visionneuse.ihm regroupe toutes les classes graphiques de l'interface : vues, gestionnaire de vues, etc. Ce paquetage inclut aussi les classes permettant aux différents composants graphiques de l'interface de communiquer entre eux.
- le paquetage fr.inra.clermont.visionneuse.operation regroupe toutes les classes correspondant aux différentes étapes.

Au final, voici l'interface obtenue :

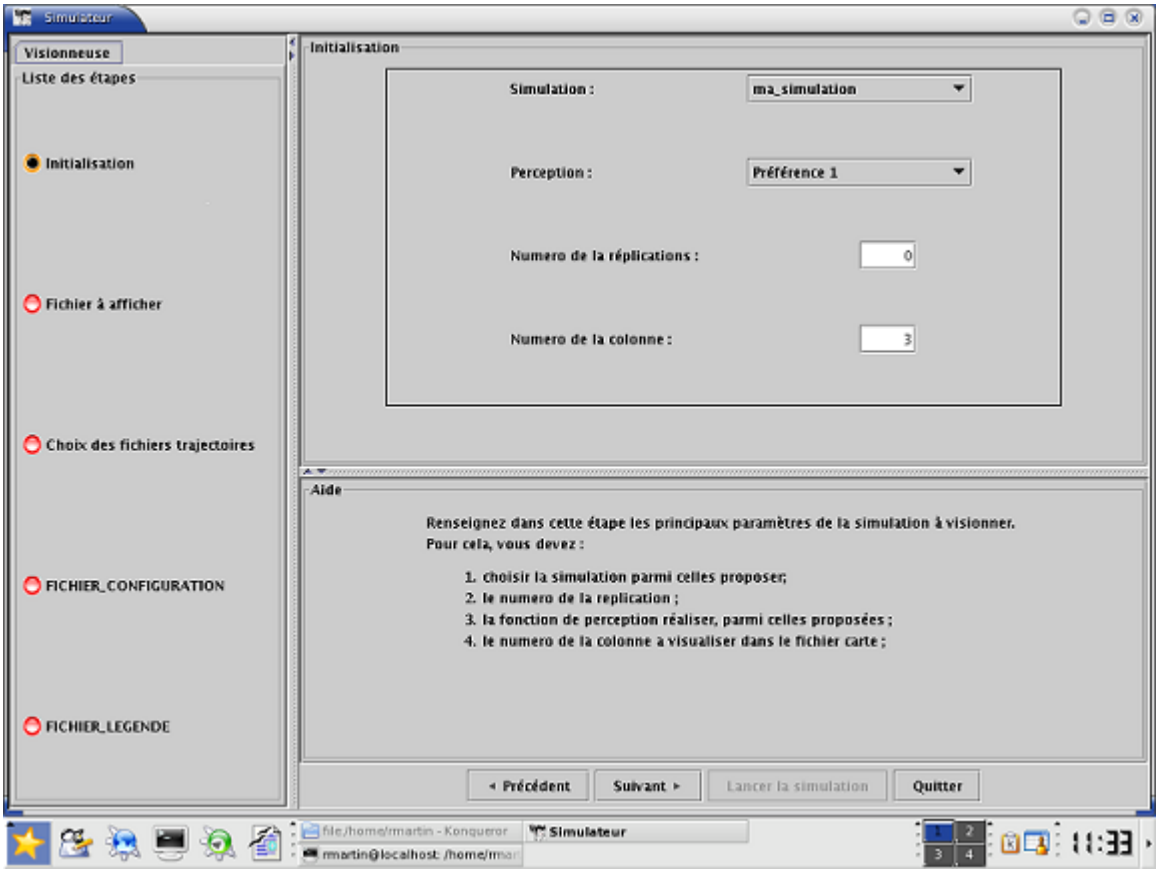

Figure 19 : Première vue de l'interface de la visionneuse

Partie 4 : Mise en place de la mémoire

4.1 Etape préliminaire

Il a tout d'abord fallu réaliser la compilation du programme sur la mémoire développé par Lucie Masson [MASSON 2001]. En effet, le programme avait déjà été développé il y a de cela plusieurs années et le compilateur gcc avait été modifié depuis. Ainsi, la fonction racine carrée n'acceptait plus un entier comme paramètre, il fallait préciser dans certains cas l'utilisation de l' « espace de nom » standard, etc. Ces quelques corrections ont été rapides à apporter.

Une fois l'exécutable créé, les essais sur différents exemples n'ont pas été concluants. En effet, les résultats obtenus n'avaient aucun rapport avec le fichier fourni en entrée. Cela provenait en réalité d'un décalage lors de la lecture. Il n'y avait donc aucune cohérence dans ce qui était affiché sans toutefois faire dysfonctionner le programme. Ce bogue une fois trouvé puis éliminé, les sorties furent enfin cohérentes et en adéquation avec celles présentées lors du précédent stage sur la mémoire [MASSON 2001].

4.2 Insertion dans le simulateur

4.2.1 Analyse

Il est souhaitable d'implémenter les deux types de mémoires car, malgré une logique et une structure totalement différentes, les différents tests réalisés jusqu'à présent ont montré que ces deux méthodes étaient équivalentes en terme de performances. Ainsi, il sera intéressant de comparer leur impact sur le comportement de l'animal.

Etant donné que les mémoires par contour et par pixellisation ont été développées hors contexte et séparément, j'ai décidé, afin de respecter leur logique d'implémentation, d'utiliser le patron de conception « strategy » [GAMMA et al. 1995].

La figure suivante représente ce patron de conception :

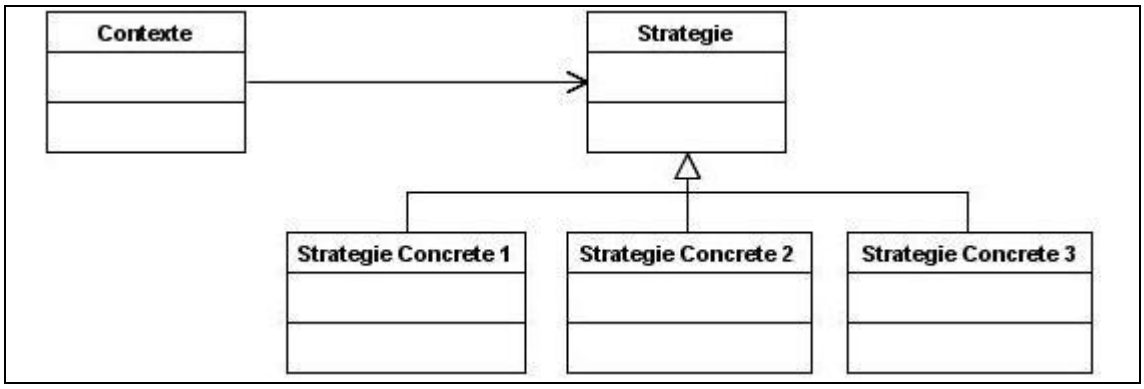

Figure 20 : Patron de conception « strategy »

La classe Contexte est, comme son nom l'indique, le contexte dans lequel la stratégie va s'insérer. Elle doit contenir une référence vers un objet Strategie. La classe Strategie déclare une interface commune à tous les algorithmes à implémenter. Enfin chacune des Strategie Concrete implémente séparément chacun de ces algorithmes.

Ce patron de conception permet tout d'abord de factoriser les fonctions communes à tous les algorithmes. Il permet de plus de bien dissocier le Contexte des différentes Strategie Concrete, ces dernières étant donc totalement indépendantes les unes des autres. Dans notre cas, le ruminant n'a qu'une seule mémoire, il ne doit donc pas « avoir connaissance » que deux méthodes de mémorisation existent. Ainsi, le fait de modifier la méthode par pixellisation ne doit en aucun cas impliquer une modification dans la méthode par contour ou dans la classe ruminant. Il sera aussi possible d'ajouter facilement une tierce méthode dans le futur du simulateur. Ce patron de conception permet aussi d'éliminer la majorité des tests. Ainsi, une unique condition déterminera pour la simulation à venir la méthode à utiliser, et ce lors de la construction de la Strategie Concrete choisie.

Toutefois, il n'a pas que des avantages. En effet, ce patron augmente le nombre de classes présentes dans le simulateur.

Malgré cela, les désavantages qu'apporte ce patron sont négligeables par rapport aux bénéfices apportés. Il est donc bien approprié à l'implémentation des deux méthodes de mémorisation.

En ce qui concerne les attributs de types statiques, ils devront tous être remplacés. En effet, chaque animal a une mémoire distincte. Aucun attribut ne peut donc être mis en commun. Un nouveau fichier de configuration devra être créé afin de paramétrer la mémoire, cette dernière nécessitant un grand nombre de constantes.

4.2.2 Implémentation

J'ai tout d'abord appliqué le patron de conception « strategy » à la mémoire telle quelle avait été implémentée précédemment [MASSON 2001]. Ainsi, la classe Contexte est bien évidemment la classe Ruminant, la classe Strategie la classe Memoire et les classes Strategie Concrete les deux classes principales de chacune des deux méthodes.

Cela soulève déjà un premier problème. Les deux classes principales des deux méthodes s'appelaient Memoire ; et nombre d'autres classes portaient le même nom, se rapportant à leur fonction, dans l'implémentation des deux méthodes de mémorisation (par exemple Memoire, GestionnaireMemoire, MemoireIntermediaire, etc.). Mais, bien sûr, ces classes n'étaient en aucun cas implémentées de la même manière, de par une logique et une SDD différentes. J'ai donc décidé de dissocier celles-ci en préfixant leur nom par « Pix » dans le cas de la méthode par pixellisation.

J'ai ensuite créé la classe mère abstraite Memoire. Cette dernière a été placée à la base de l'arborescence, ainsi que ElementChaine, élément de base d'une liste chaînée. Pour pallier la nécessité de retirer les attributs et méthodes statiques et pour garder la même logique d'implémentation, une classe StatMemoire a été créée. Cette classe sera bien sûr instanciée une unique fois à la création de la mémoire et détruite en même temps que cette dernière. Le code du reste du prototype a ensuite été modifié en conséquence.

Pour donner à l'utilisateur la possibilité de configurer la mémoire comme il l'entend avant le lancement d'une simulation, une nouvelle classe dérivant de FichierConfig a vu le jour : FichierConfigMemoire. Cette dernière contient tous les attributs utiles à la mémoire, y compris son type, ainsi que tous les autres attributs anciennement inclus directement dans le code source, en « dur ».

La figure 21 montre le résultat obtenu :

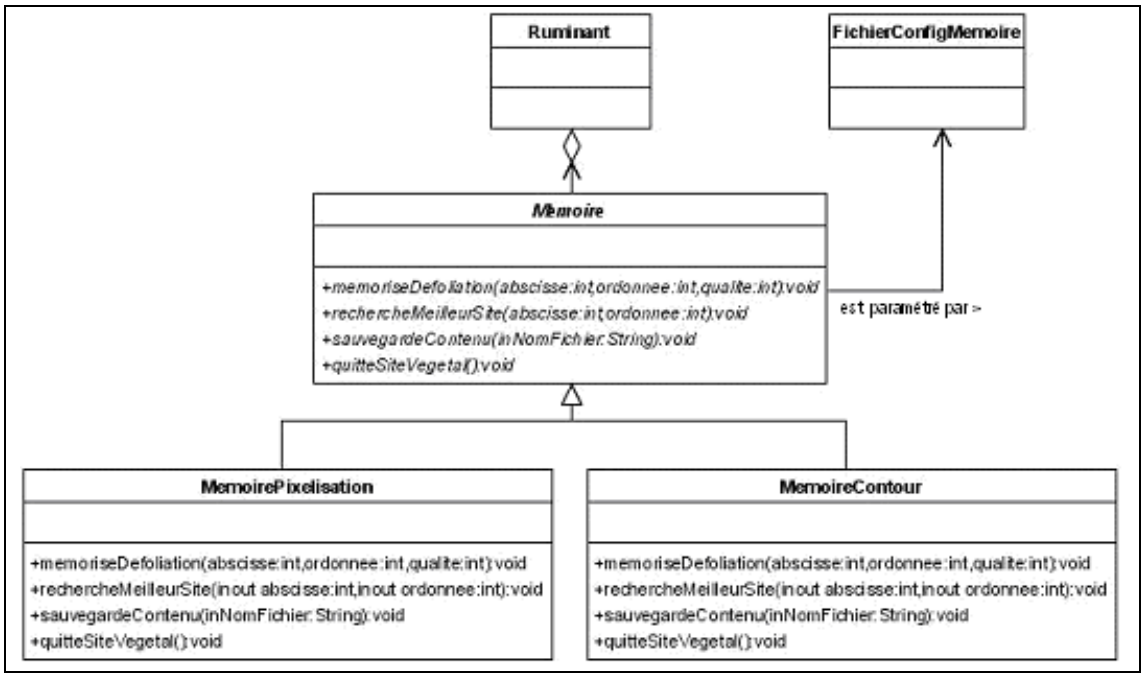

Figure 21 : Diagramme de classe de la mémoire

Ainsi, à la fin de chaque défoliation, le Ruminant fait appel à la méthode virtuelle memoriseDefoliation de sa Memoire qui, par héritage, fait appel à la méthode éponyme d'un de ses deux fils. Il en va de même pour rechercheMeilleurSite qui renvoie les coordonnées de la meilleure cellule selon le coefficient $\frac{qualite \times surface}{eloignement}$, pour sauvegardeContenu qui fait le copier/coller de la mémoire dans le fichier dont le nom est passé en paramètres et pour quitteSiteVegetal qui permet de conclure une séquence de défoliation.

4.3 Insertion de la mémoire dans l'interface du simulateur.

Là encore, l'insertion dans l'interface de la configuration de la mémoire s'est faite aisément. En effet, il a suffi de créer un fichier memoire.cfg regroupant l'ensemble des données utiles (c'est ce fichier qui sera lu par la classe FichierConfigMemoire), de le placer dans le fichier standard du dossier simulateur et de préciser qu'il est à éditer dans le fichier configdyn.csv. Tout est ensuite géré et généré automatiquement.

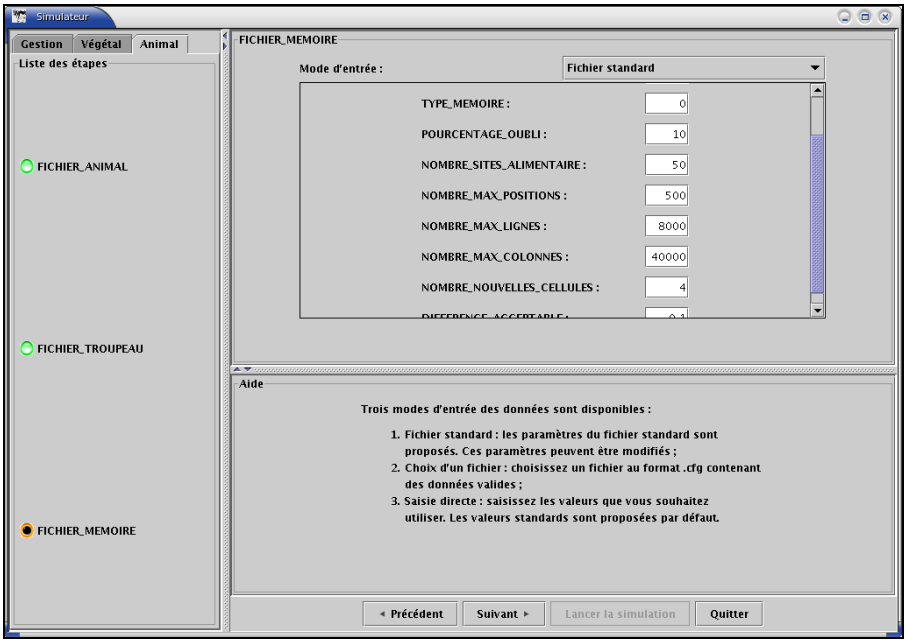

Figure 22 : Etape permettant la configuration de la mémoire

4.4 Bilan provisoire

Les premières observations concernant la création des sites ont montré des anomalies. En effet, la distinction entre deux sites dans la carte mémorielle n'était pas perceptible et l'animal avait tendance à uniformiser toutes les valeurs. Cela était dû à une valeur codée en dur dans le code. Lors de l'agrégation de deux sites, on testait si leur différence de valeur était inférieure à 10. Or, initialement, les tests avaient été effectués avec des valeurs arbitraires comprises entre 0 et 100, puisque la mémoire était développée hors contexte du simulateur. Cela ne causait donc pas de dysfonctionnement dans le programme initial puisque lorsque deux sites étaient distincts, leur différence était le plus souvent supérieure à 10. Mais dans le cas du simulateur, la qualité perçue se situe le plus souvent entre 0 et 4 (bien que théoriquement cette qualité puisse tendre vers l'infini). La mémoire agrégeait alors tous les sites mémoriels ensemble, en pondérant leurs valeurs, ce qui donnait le résultat décrit précédemment. La différence entre deux sites a donc été placée en paramètres.

J'ai procédé alors à quelques tests. Le temps d'exécution des deux modèles a été assez semblable. Ainsi, pour une carte de 1000*1000 comportant 12 sites, avec un animal et pour une durée de cinq jours, la durée d'exécution est de trois minutes 40 secondes approximativement. Les autres résultats corroborent cela.

J'ai ensuite vérifié la cohérence des cartes mémorielles créées par l'animal avec les parcelles fournies en entrée du simulateur. Pour cela, j'ai créé une moulinette qui permet d'extraire la qualité perçue d'une cellule grâce à un fichier de mise à jour. En

effet, aucune sortie du simulateur ne donne directement les valeurs de qualité et l'animal, à un instant donné, ne voit que les 15 cellules de son champ de vision. Il fallait donc créer un outil permettant de se faire une idée de la qualité d'une parcelle, afin de tester le bon fonctionnement de la mémoire. Ce dernier rajoute donc une colonne en fin du fichier de mise à jour de la végétation représentant la qualité théorique de chaque cellule. Ce formalisme permet à ce fichier de continuer d'être exploitable par la visionneuse.

Dans le cas de la perception OFT, l'adéquation entre ce qui est perçu et la réalité était évidente. Les différents sites étaient bien distincts, ne s'agrégeant pas les uns avec les autres. Ils étaient donc identifiables et leur qualité était proche de la qualité théorique. En revanche, dans le cas de la perception Preference, le comportement de la fonction étant extrême (deux faciès à 0.9…, un à 10^{-12} et un à10^{-12}), l'animal ne différenciait que deux sites mémoriels distincts. Ce problème a été soumis à Bertrand Dumont, qui a exprimé le besoin de trouver une relation entre les deux modèles de perceptions.

4.5 Etude des fonctions de perceptions et solution apportée

Les deux équations des fonctions de perception étant très différentes (cf. Annexe E), une bijection aurait été très difficile à trouver, si tant est qu'elle existe. Nous avons donc décidé de faire une étude graphique afin de déterminer si l'une est fonction de l'autre. Les graphiques ainsi créés sont présentés dans l'annexe F. La figure suivante en présente un exemple :

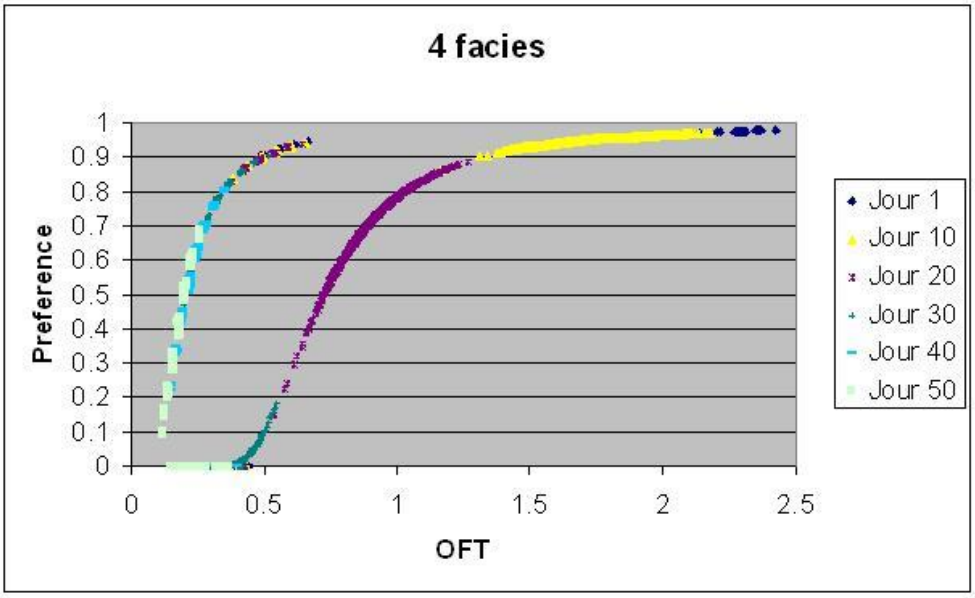

Figure 23 : Graphique permettant d'espérer une bijection entre les deux perceptions

Malgré la recherche de nombreuses relations (simple, rapport, différence), aucune bijection n'a pu être mise en évidence. Les utilisateurs devront donc saisir deux séries de valeurs, l'une pour l'OFT, l'autre pour la Preference. La figure 24 montre la nature chaotique de la relation entre ces deux fonctions :

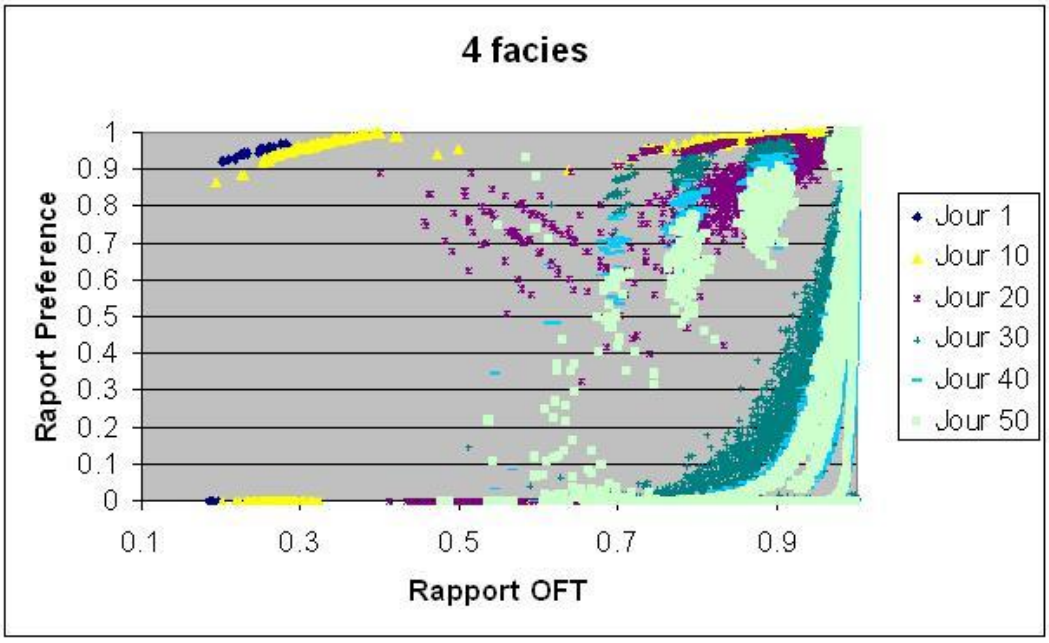

Figure 24 : Graphique infirmant toute hypothèse de bijection entre les perceptions

4.6 Amélioration de la mémoire

4.6.1 Comparaison par rapport à la moyenne

Dans le prototype de la mémoire tel qu'il avait été implémenté, l'animal compare la qualité de la cellule courante à celle défoliée précédemment. Ceci pouvait entraîner le biais suivant : l'animal peut commencer la création d'un site sur une certaine qualité et arriver, après un dégradé successif, à une qualité très éloignée de la première. Pourtant, à aucun moment, la différence de deux qualités successives n'aura franchi la limite qui les différencient comme appartenant à deux sites distincts. Pour pallier cela, il a donc été décidé de comparer non plus la qualité courante à la qualité précédente, mais à celle de la moyenne de l'ensemble des cellules du site en construction. Cela n'assure bien sûr pas que le biais précédent n'aura jamais lieu, mais il peut être maintenant considéré comme négligeable. Ceci a été implémenté directement dans le modèle par contour mais n'a pas pu être fait pour le modèle par pixellisation, puisque ce dernier ne forme les sites qu'à la fin de chaque phase.

4.6.2 Instauration des différences relatives

Une demande de Bertrand Dumont a été d'instaurer des différences relatives, et plus particulièrement les rapports, et non des seuils lors des comparaisons entre sites ou lors des défoliations. En effet, les seules études réalisées sur la mémoire donnent des résultats en termes de rapports et non de différences. Toutefois, cette relation étant non symétrique, ce type de calcul introduit une notion d'ordre. Imaginons deux sites mémoriels, de qualités respectives 1 et 1.11. Si le seuil est de 10%, alors dans un cas, les deux sites s'agrègent ($\frac{1}{1.11} \cong 0.9009 \in [0.9;1.1]$), alors que dans l'autre, les deux sites restent scindés ($\frac{1.11}{1} = 1.11 \notin [0.9;1.1]$). Nous avons donc fixé que le dénominateur serait toujours le nombre le plus grand.

Ensuite, une borne imposant une différence minimum entre deux qualités pour les considérer comme appartenant à deux sites différents a elle aussi été posée afin d'éviter le biais des qualités infimes. En effet, dans le cas où l'on comparerait un site de qualité 10^{-12} à un site de qualité10^{-35}, ils resteraient séparés alors que l'animal ne peut percevoir une telle différence. L'apparition de la borne permet de traiter ce cas.

Enfin, l'idée que les seuils ne soient pas fixes mais suivent une logique floue a été soulevée. Ainsi, si le biologiste entrait comme valeur 5% et 15%, cela signifierait qu'au-dessus de 0.95 lors du calcul du rapport des deux qualités, la probabilité d'agrégation serait de 100%, qu'en dessous de 0.85, elle serait nulle et qu'entre les deux, elle varierait linéairement. Toutefois, cette idée a été rejetée car elle instaurait un flou sur un processus qui se veut déjà flou. Ainsi, cela n'aurait fait qu'augmenter le nombre de paramètres sans pour autant rajouter une donnée significative au simulateur.

Partie 5 : Résultats et discussion

5.1 Outils utilisés

Lors de mon stage, l'INRA a mis à ma disposition un PC muni d'un processeur AMD Athlon XP 2600+, cadencé à 1.9 GHz, ainsi que 512 Mo de mémoire vive.

Cet ordinateur était muni uniquement du système Windows XP ; ma première tâche fut donc d'installer un système d'exploitation Linux, le simulateur ayant été développé sur ce dernier. Le choix s'est porté sur la distribution Mandrake 9.2 dans un premier temps, puis sur la 10.0 quand celle-ci est sortie afin d'assurer une plus grande pérennité. Le simulateur est codé en langage C++ et le compilateur utilisé est g++ de GNU. En ce qui concerne le langage JAVA, utilisé pour implémenter les interfaces, la machine virtuelle utilisée est elle aussi la plus récente possible, c'est-à-dire la version 1.4.2.04.

Afin de développer l'interface de la visionneuse, j'ai utilisé le logiciel gratuit Eclipse version 2.1.1, qui permet une gestion dynamique des erreurs.

En ce qui concerne la mise en place de la documentation, mon choix s'est porté sur un des générateurs le plus célèbre : Doxygen (version 1.3.6), et ce pour les raisons citées plus haut. Je l'ai associé avec le logiciel Graphviz, version 1.12, qui permet la génération automatique de diagramme UML.

Enfin, pour la phase d'analyse, c'est le langage de modélisation UML qui a été retenu car c'est la norme mondiale dans le domaine objet et il permet, même pour des non informaticiens, une compréhension aisée et intuitive. Afin de créer ces diagrammes, j'ai utilisé un des leaders dans ce domaine : le logiciel Poséidon for UML Community Edition 2.3.1.

5.2 Résultats, fonctionnement des programmes

Actuellement, le simulateur inclut la mémoire spatiale, celle-ci n'incluant pas les fonctions d'oubli et d'exploration. L'animal peut donc mémoriser une qualité, créer des sites mémoriels en agrégeant ou désagrégeant ces dernières et finalement faire appel à sa mémoire afin de se diriger vers un site particulier.

Cette mémoire est « injectée » chaque jour dans un fichier de sortie, et ce afin de vérifier son intégrité. Ce fichier est pris ensuite en entrée par la visionneuse via l'interface et permet d'afficher les cartes mémorielles suivantes :

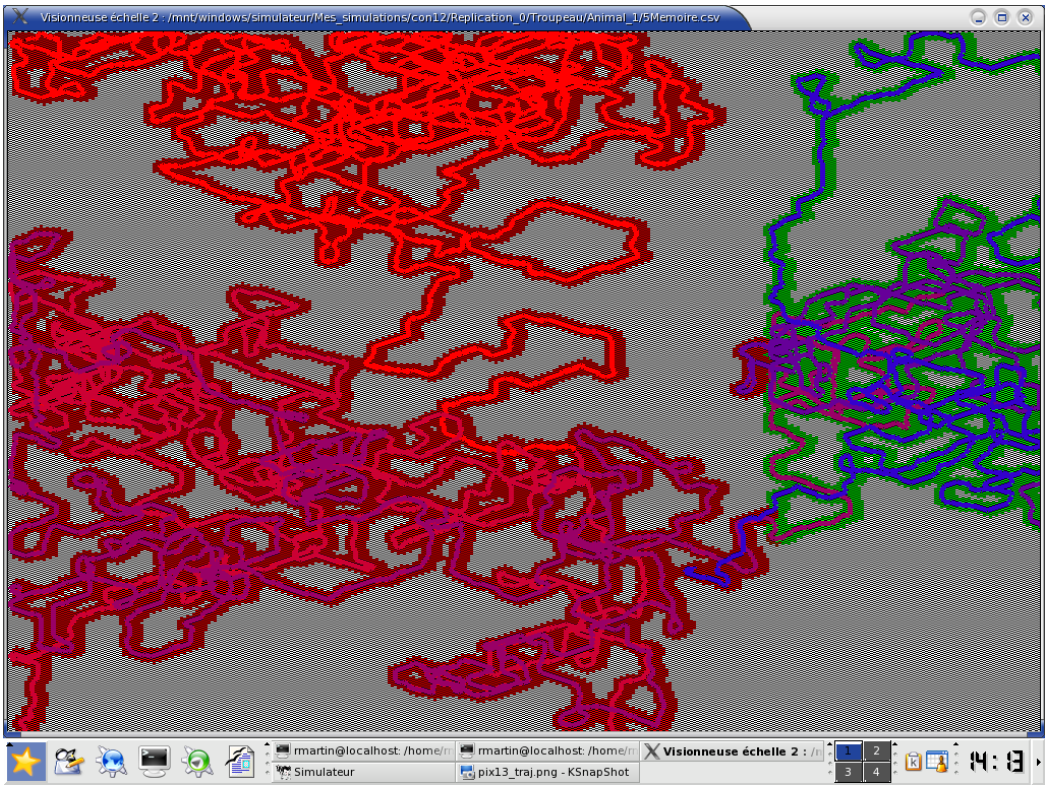

Figure 25 : Capture d'écran représentant la carte mémorielle par contour

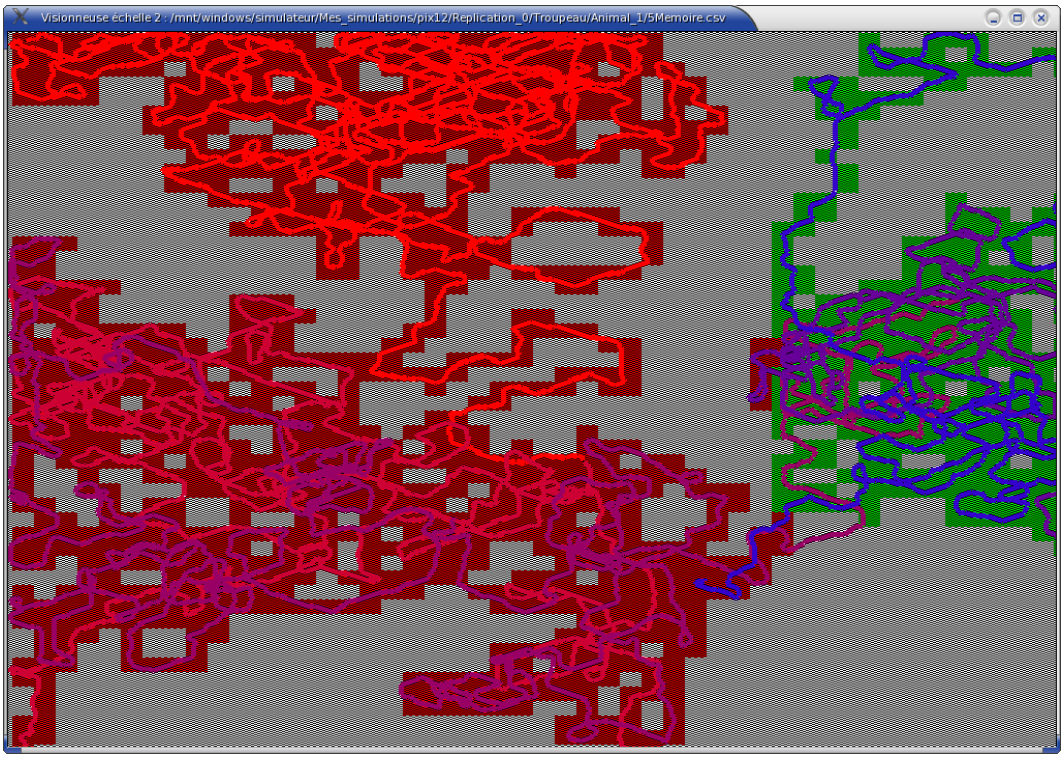

Figure 26 : Capture d'écran représentant la carte mémorielle pixellisée

On peut enfin noter que la documentation du simulateur est effective bien que pouvant et devant être complétée. Elle fournit un outil simple et pratique qui permettra dans le futur un travail plus efficace sur le simulateur.

5.3 Difficultés rencontrées

La première difficulté fut rencontrée avant même l'analyse du simulateur. En effet, j'ai voulu tout d'abord lancer quelques simulations afin de me familiariser avec les sorties et l'interface d'alors, et, le cas échéant, d'essayer de mettre en défaut le simulateur. Mais l'interface, lors de son exécution, levait un grand nombre d'exécutions puis se stoppait. Cela provenait en réalité de la version de la jvm qui ne fonctionnait pas correctement avec le noyau de Linux Mandrake 10.0. Une fois ceci déterminé, j'ai installé une version plus récente qui cette fois autorisait la bonne marche de l'interface. Toutefois, le fait de choisir une version plus récente posa d'autres problèmes. Certains logiciels, tel JBuilder, n'étaient pas adaptés à cette dernière. Je dus donc m'adapter à de nouveaux outils, tel qu'Eclipse.

Une fois le logiciel opérationnel, l'étude à proprement parler du simulateur put commencer et là encore, ceci souleva quelques problèmes. En effet, de par le manque de commentaires et l'abstraction de certaines de ses fonctions, il était trés dur de comprendre leurs effets. La mise en place de la documentation a donc pris du temps, car il fallait tout d'abord rendre le code plus propre, le commenter mais aussi chercher la signification des variables biologiques. Ceci me permit donc de me familiariser avec leurs noms.

Enfin, en ce qui concerne l'implémentation, aucune réelle difficulté ne s'est posée. En effet, les modifications et autres insertions se sont réalisées petit à petit, en collaboration avec les biologistes. Ainsi, le plus ardu fut de leur expliquer d'un point de vue « non informatique » un projet qui lui l'était. C'est en ceci que les diagrammes UML ont été utiles.

5.4 Futur du simulateur

Le simulateur PARIS est un projet qui est maintenant mené depuis plusieurs années. Avec ce stage, il atteint enfin les objectifs posés lors de sa création. Le prochain stage devrait donc permettre de le valider en l'état actuel, et ainsi, de le calibrer. Etant prévu initialement pour des ovins, il devrait, via une modification des paramètres en entrée, accepter tous autres ruminants, comme par exemple des vaches.

Certains outils devront tout de même être améliorés, comme la visionneuse. En effet, des problèmes se posent pour la vision de grandes parcelles et l'échelle trois n'est pas encore au point.

De plus, une étude comparative des deux méthodes de mémorisation pourra être réalisée, afin d'étudier leurs influences sur le comportement animal.

Conclusion

Les objectifs du stage ont, pour la plupart, tous été atteints. Il ne reste, à l'heure actuelle, qu'à implémenter l'oubli et l'exploration, et ceci sera certainement fait à l'issue du stage. En effet, ces deux dernières fonctions ont été modélisées et il ne reste plus qu'à les ajouter proprement au code. Ce prototype sera alors complet et en attente d'une validation.

Ce stage m'a beaucoup apporté, autant sur le plan technique que relationnel. En effet, sur le plan de la méthodologie du travail, j'ai pris conscience de l'importance des diagrammes UML et de la documentation, et ce pour plusieurs raisons. Tout d'abord, cette base ainsi créée permet d'étayer plus facilement ses propos avec des personnes qui ne sont pas dans le projet continuellement et ainsi de dialoguer plus facilement. Elle fournit aussi une aide conséquente lors des phases d'implémentations et de déboguages. En ce qui concerne la programmation, je pense avoir renforcé mes bases en C++ et JAVA.

Sur le plan relationnel, j'ai été en collaboration avec des personnes dont les domaines d'activités étaient différents des miens. J'ai donc dû m'adapter à la terminologie biologique et intégrer à toutes les étapes de conception le point de vue utilisateur. J'utilisais enfin l'outil informatique comme un moyen, non comme une fin.

Je peux donc dire que ce stage m'a apporté une bonne vision de ce que peut être le travail dans un laboratoire de recherche, de la complémentarité de divers domaines au sein d'une équipe et des difficultés de communication que cela entraîne.

Bibliographie

[ASTRE 2004] ASTRE B., 2004. Analyse multi variée des résultats d'un simulateur multi-agents en vue de sa validation, rapport de stage INRA 2004.

[BAUMONT et coll. 2002] BAUMONT R., DUMONT B., CARRERE P., PEROCHON L., MAZEL C., FORCE C., PRACHE S., LOAULT F., SOUSSANA J.F., HILL D.R.C., PETIT M., 2002. Développement d'un modèle multi-agents spatialisé d'un troupeau de ruminants pâturant une prairie hétérogène. Rencontres Recherches Ruminants, 9, 69-72.

[BAILEY et al. 1996] BAILEY D.W., GROSS J.E., LACA E.A., RITTENHOUSE L.R., COUGHENOUR M.B., SWIFT D.M., SIMS P.L., 2001. Mechanisms that result in large herbivore grazing distribution patterns, Journal of range management 49(5), 394-397.

[CARRERE et al. 2002] CARRERE P., FORCE C., SOUSSANA J.F., LOUAULT F., DUMONT B., BAUMONT R, 2002. Design of a spatial model of a perennial grassland grazed by a herd of ruminants : the vegetation sub-model. EGF 2002. Grassland Science in Europe : 7, 282-283.

[DUMONT et HILL 2001] DUMONT B., HILL D.R.C., 2001. Multi-Agent Simulation of Group Foraging in Sheep: effects of Spatial Memory, Conspecific Attraction and Plot Size, Ecological Modelling 141, 201-215.

[DUMONT et HILL 2003] DUMONT B., HILL D.R.C., 2003. Spatially explicit models of group foraging by herbivores : What can Agent Based Models offer ?, Accepté pour publication dans Animal Research 2003.

[GAMMA et al. 1995] GAMMA E., HELM R., JOHNSON R., VLISSIDES J., 1995. Design Patterns, Element of Reusable Object-Oriented Software, Software Development, pages 315-323.

[GUERRY 2000] GUERRY F., 2000. Conception d'u simulateur multi-agents parcelle/troupeau, rapport de stage INRA.

[LEMLOUMA et BOUDINA 2001] LEMLOUMA T., BOUDINA A., 2001. L'intelligence distribuée et les systèmes multi-agents, publication INRIA Rhône-Alpes.

[MARTIN 2002] MARTIN G., 2002. Analyse et conception du couplage entre les modules animal et végétal du simulateur troupeau/parcelle, rapport de stage INRA.

[MARTIN et PEAN 2004] MARTIN R., PEAN B., 2004. Simulateur Prairie/Troupeau : Développement d'un module de défoliation contrôlée, rapport de projet.

[MASSON 2001] MASSON L., 2001. Modélisation de la mémoire spatiale des animaux dans un simulateur parcelle/troupeau, rapport de stage INRA.

[MASSON et VILLEGER 2001] MASSON L., VILLEGER A., 2001. Conception d'un module de visualisation pour un simulateur parcelle/troupeau, rapport de projet.

[PEROCHON et al. 2001] PEROCHON L., CARRERE P., BAUMONT R., DUMONT B., MAZEL C., FORCE C., HILL D., D'HOUR P., LOUAULT F., PRACHE S., SOUSSANA J.F., PETIT M., 2001. Design of a spatial multi-agent model of a perennial grassland ecosystem grazed by a herd of ruminants**.** Simulation in industry'2001, 13th European Simulation Symposium 2001, N. Giambiasi and C. Frydman (Eds.). October 18-20, 2001, Marseille, France. 509-513.

[PORTAL 2003] PORTAL S., 2003. Conception du module spatial et social d'un simulateur parcelle/troupeau, rapport de stage INRA.

[PORTAL et SEGUY 2003] PORTAL S., SEGUY R., 2003. Interface graphique pour un simulateur parcelle/troupeau, rapport de projet.

[SAUVANT et al. 1996] SAUVANT D., BAUMONT R., FAVERDIN P., 1996. Development of a Mechanistic Model of Intake and Chewing Activities of Sheep, Journal of Animal Science, 74. 2785-2802.