Conception du module spatial et social d'un simulateur parcelle/troupeau

Séverine Portal, Laurent Perochon, Bertrand Dumont, Claude Mazel

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

Complexe des Cézeaux
BP 125
63170 Aubière Cedex

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

Unité de recherche sur les herbivores
Theix
63122 Saint-Genès-Champanelle

Rapport de stage 2ème année

Conception du module spatial et social d'un simulateur parcelle/troupeau

Tome I

Présenté par : Séverine Portal
Lieu de stage : INRA Theix
Responsables du stage :
Laurent Pérochon, Bertrand Dumont
Tuteur ISIMA : Claude Mazel
Tuteur INRA : Laurent Pérochon

Du 7 avril au 26 septembre 2003

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

Complexe des Cézeaux
BP 125
63170 Aubière Cedex

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

Unité de recherche sur les herbivores
Theix
63122 Saint-Genès-Champanelle

Rapport de stage 2ème année

Conception du module spatial et social d'un simulateur parcelle/troupeau

Tome I

Présenté par : Séverine Portal
Lieu de stage : INRA Theix
Responsables du stage :
Laurent Pérochon, Bertrand Dumont
Tuteur ISIMA : Claude Mazel
Tuteur INRA : Laurent Pérochon

Du 7 avril au 26 septembre 2003

Remerciements

Je tiens à remercier M. Laurent Pérochon pour ses conseils et idées en matière d'analyse et aussi pour le temps qu'il m'a consacré tout au long du stage. Je remercie aussi M. Bertrand Dumont qui m'a éclairé sur le comportement social des ruminants à de nombreuses reprises et M. René Baumont qui a réfléchi à des équations pour le module animal.

Je remercie également M. Claude Mazel qui nous a proposé des solutions judicieuses pour résoudre nos problèmes. Merci aussi à M. David Hill pour ses conseils pour la correction des problèmes de l'interface et à M. Pascal Carrère pour son soutien et sa collaboration au cours du stage.

Enfin merci à toute l'équipe RAP de l'INRA qui m'a si bien accueillie.

Table des figures et illustrations

Table des abréviations

INRA : Institut National de la Recherche Agronomique
RAP : Relations Animal / Plantes
RS : compartiment Reproducteur Sec
RV : compartiment Reproducteur Vert
UML : Unified Modeling Language - Langage de modélisation unifié
URH : Unité de Recherche sur les Herbivores
VS : compartiment Végétatif Sec
VV : compartiment Végétatif Vert

Glossaire

Biomasse : Quantité de matière disponible sur une cellule

Défolier : Brouter

Grégarisme : Relation sociale au sein d'un troupeau de ruminants qui pousse les animaux à rester groupés (aussi appelé grégarité)

Leadership : Relation sociale au sein d'un troupeau de ruminants qui pousse les animaux à suivre un animal particulier, appelé leader, lors des longs déplacements sur la parcelle

Sénescence : Vieillissement naturel des tissus et de l'organisme

Résumé

Mon stage consiste à compléter un **simulateur** parcelle/troupeau modélisant une période d'activité d'un ruminant sur une parcelle. L'objectif est d'étendre la simulation à un troupeau de ruminants en gérant l'ensemble des **relations sociales** existant au sein du groupe.

Ce simulateur comprend deux modules biologiques : le module animal qui met à jour les paramètres physiologiques de l'animal au cours de la simulation et le module végétal qui modifie les variables de la végétation pour tenir compte des défoliations de l'animal, de la pousse de l'herbe et de la sénescence. Ces deux modules ont été interconnectés pour donner un **prototype** du simulateur qui comprend aussi des outils facilitant son utilisation. Ce prototype doit être complété par un module spatial et social permettant de tenir compte des relations sociales entre ruminants pour donner un nouveau prototype complet.

Mon travail s'est déroulé en plusieurs étapes. Dans un premier temps, j'ai modifié quelques aspects du simulateur et apporté des corrections qui ont permis de supprimer des erreurs d'exécution. J'ai ensuite ajouté des programmes qui étaient implémentés mais pas encore utilisés puis j'ai modifié le simulateur pour pouvoir intégrer plusieurs animaux. Dans une troisième étape, j'ai pris en compte les relations de **grégarisme** et enfin les relations de **leadership**. Chaque étape a d'abord été modélisée grâce au langage de modélisation **UML** puis programmée sous Linux avec le langage **C++**.

Mon travail est à ce jour encore inachevé puisqu'il reste à ajouter un dernier programme dans le simulateur : la mémoire spatiale. Cette application a été entièrement analysée et codée, il suffira de la rajouter dans le prototype. L'objectif de mon stage devrait donc être atteint à la fin du mois de septembre. Il faudra ensuite vérifier de manière approfondie ce nouveau prototype, même si chaque étape de mon travail a été testée au fur et à mesure, avant de pouvoir commencer sa validation.

Mots-clés : simulateur, relations sociales, prototype, grégarisme, leadership, UML, C++

Abstract

My training period has consisted in completing a plot/herd **simulator,** which so far modeled the grazing activity of a ruminant within a plot. The aim was to extend this simulation so that we can represent the activity of a whole group, which implies to consider the **social relationships** between group mates.

This simulator includes three sub-models. The animal sub-model updates the physiological parameters of an animal during the simulation and the vegetal sub-model updates the vegetation variables to take into account grass growth and senescence, and defoliation by the animals. These two sub-models have already been connected to build a first **prototype,** which includes some additional tools facilitating its use. This prototype needed to be completed by a spatial and social sub-model dealing with the social relationships within the herd, in order to obtain a first simulator of the whole system.

My work has unfolded into several steps. First, I have modified some aspects of the first prototype, and made corrections which have suppressed some execution errors. Then, I have included other programs into this first prototype, which had been developed but were not used yet, and I have modified the simulator so that he can cope with the simultaneous grazing of several animals. At last, I have been developing **social attraction** relationships and finally **leadership**. Each step has been modeled with the **UML** modeling language, before to be programmed under Linux with the **C++** language.

My work remains unfinished as we still need to add the spatial memory program in the simulator, though this application has been entirely analyzed and developed. This should be done by the end of September. Then, this new prototype will have to be verified as a whole, even though each step of my work has already been tested, before to start the validation process.

Keywords : simulator, social relationships, prototype, social attraction, leadership, UML, C++

Table des matières

Tome I

Tome II

Introduction

Dans le cadre de mon stage qui s'est déroulé du 7 avril au 26 septembre 2003 à l'Institut National de la Recherche Agronomique de Clermont-Ferrand - Theix, j'ai eu l'occasion de travailler sur un simulateur parcelle/troupeau.

Ce simulateur a été développé dans le cadre d'un projet de recherches nommé « l'utilisation par le pâturage et l'évolution de la végétation des surfaces herbagères hétérogènes et peu chargées » pour observer l'impact du pâturage sur l'évolution de la végétation et ainsi éviter l'embroussaillement des parcelles car la végétation y est moins riche. Ce simulateur a déjà été commencé depuis plusieurs années et a fait l'objet de nombreux projets et stages de développement.

Le prototype actuel intègre un seul animal, le but de mon stage est d'étendre la simulation à un troupeau de ruminants. Les relations sociales entre les animaux sont de deux types, le grégarisme et le leadership. Ces relations devront être gérées puisqu'elles modifient la répartition géographique des animaux sur la parcelle et leur comportement.

Mon travail consistera d'une part à rajouter des programmes non utilisés dans le simulateur, après avoir étudié leur fonctionnement et ensuite à faire une analyse détaillée du comportement social des ruminants et l'implémenter.

Je présenterai dans un premier temps le fonctionnement du simulateur avant le début de mon stage. Je décrirai ensuite le travail de conception et de réalisation effectué ainsi que les solutions retenues. Enfin, je conclurai en donnant les résultats obtenus et les éventuelles améliorations qui pourraient être apportées au simulateur.

Partie I : Contexte technique

1. Objectifs du stage

L'équipe Relations Animal Plantes de l'Unité de Recherche sur les Herbivores de l'Institut National de la Recherche Agronomique de Clermont-Ferrand - Theix effectue des recherches sur le comportement alimentaire des ruminants. Un de ses axes de recherche est la création d'un simulateur modélisant les interactions entre un troupeau de ruminants et la dynamique d'une parcelle de végétation. Ce simulateur est basé sur un système multi-agents. Les agents sont ici des animaux possédant des caractéristiques propres et interagissant directement (par éloignement ou rapprochement) ou indirectement (par le partage d'une même ressource : la végétation). L'objectif est de pouvoir expliquer l'impact du troupeau sur la végétation en comprenant les mécanismes qui se situent au niveau individuel. Cette approche permet non seulement d'étendre les connaissances scientifiques et de tester des hypothèses avec une souplesse impossible à avoir en condition réelle. A terme, il pourra conduire à la création d'un outil permettant de tester plusieurs possibilités d'utilisation de la parcelle, afin de pouvoir mieux la gérer.

Au début du stage, le dernier prototype du simulateur représente un seul animal sur une parcelle. L'animal se déplace et défolie des cellules qu'il choisit. La végétation a une croissance et une sénescence. Enfin, une visionneuse permet d'observer la trajectoire suivie par l'animal et une interface utilisateur facilite l'utilisation de ce simulateur.

Le but de mon stage est de permettre l'utilisation du simulateur avec un troupeau, en gérant les relations sociales et spatiales existant entre les individus. Ceci constituera la première version complète du simulateur.

2. L'approche multi-agents

Un système multi-agents consiste à faire coopérer un ensemble d'entités ou agents dotées d'un comportement intelligent, coordonner leurs buts et leurs plans d'actions pour résoudre un problème [LEMLOUMA et BOUDINA 2001].
Un agent est une entité, physique ou abstraite, caractérisée par :

- Son autonomie dans la prise de décision ;
- Ses connaissances sur lui même et sur les autres ;
- Sa capacité d'agir.

Si un système est considéré comme un ensemble d'éléments en interaction, un système complexe peut être compris en le décomposant en éléments simples analysables séparément [DUMONT et HILL 2003]. Chez les herbivores, la simulation des individus aide à obtenir des informations élémentaires qui peuvent être utilisées et combinées pour aider à comprendre le comportement du groupe.

L'approche multi-agents est fondée sur un principe général d'individualisation des systèmes en un ensemble d'entités élémentaires en interaction [FERRAND 1998]. Elle consiste donc pour un système complexe, à le décrire, le modéliser, l'analyser puis éventuellement, sur une base informatique, à le simuler, résoudre les problèmes afférents (recherche de configurations satisfaisant des contraintes), instrumenter sa gestion en distinguant en son sein un ensemble d'entités individualisées : les agents. Identifiés par un état et une évolution dynamique, ils peuvent interagir simultanément, entre eux (interactions internes ou sociales), et avec un ensemble de données et processus extérieurs : l'environnement. Les agents sont structurées au sein d'une organisation, et constituent ainsi une information complémentaire produite par le système. Les agents sont aptes à contrôler leurs interactions en fonction de leur état.

Le concept d'agents logiciels indépendants a été introduit par l'intelligence artificielle distribuée comme une extension des acteurs et objets logiciels. La notion d'acteur informatique a été introduite pour exprimer les aspects actifs et autonomes de certains objets. Ensuite les agents informatiques ont été introduits dans une notion plus avancée pour prendre en compte des interactions sociales complexes. En effet, dans un système multi-agents, les agents renferment et exécutent leurs tâches individuelles localement mais ils influencent le comportement global du système. Les systèmes multi-agents sont donc appropriés au pâturage des herbivores parce que ce sont des organismes individuels qui effectuent des tâches en parallèle et qui ont des actions individuelles influençant les comportements de groupe.

L'approche multi-agents a été retenue parce qu'elle s'adapte remarquablement bien aux modèles nécessitant la prise en compte de l'espace et des comportements individuels [DUMONT et col. 2001]. Ainsi, chaque agent simule le comportement d'un animal qui peut avoir des capacités cognitives (capacité de mémorisation, etc.), des préférences alimentaires et un statut social (comportement de leadership, etc.) propres. Chaque agent est un objet actif, autonome et intégrant un comportement social, qui agit en fonction de son état interne et de ce qu'il perçoit de son environnement.

3. Description de l'existant

L'analyse a conduit à une séparation du système biologique en trois modules (végétal, animal et social et spatial) représentant certaines fonctions spécifiques. Un module spécial a également été analysé, il s'agit de tous les aspects techniques liés à la simulation. L'implémentation a ensuite commencé et différents programmes ont été conçus. Certains d'entre eux ont été agrégés en un prototype. C'est ce que je présenterai ici.

3.1. Description des modules

3.1.1. Le module végétal (V)

Le module végétal se charge de la gestion de la parcelle sur laquelle évolue l'animal. La parcelle se compose de cellules hexagonales regroupées en sites et faciès. Un faciès est un ensemble de cellules ayant les mêmes caractéristiques et appartenant à la même communauté végétale. Les cellules voisines d'un même faciès sont regroupées en sites. Un site est donc associé à un seul faciès alors qu'un faciès peut contenir plusieurs sites.

Une cellule est également composée de quatre compartiments : le végétatif vert (VV), le végétatif sec (VS), le reproducteur vert (RV) et le reproducteur sec (RS). Les deux compartiments VV et RV correspondent à la végétation fraîche qui est préférée par l'animal à la végétation fanée composée des compartiments VS et RS.

Les propriétés de la parcelle doivent évoluer au cours de la simulation, c'est le rôle du module végétal. Notamment, la biomasse de chaque compartiment est mise à jour lors d'une défoliation de cellule par l'animal et aussi tous les soirs pour tenir compte de la pousse de l'herbe et de la sénescence de la journée.

3.1.2. Le module animal (A)

Le module animal reprend le modèle d'ingestion de Sauvant et al. [SAUVANT et al. 1996]. Il gère les paramètres d'ingestion, l'état physiologique de l'animal ainsi que les actions qu'il va choisir d'effectuer au cours de la simulation. L'algorithme de décision est très simple, le choix de l'activité dépend du besoin en nourriture de l'animal. Il peut manger, ruminer ou se reposer mais aussi se déplacer sur la cellule voisine pour la défolier. Il s'agit d'un déplacement court, à différencier du déplacement long qui sera décrit plus loin. Il faut également dissocier le repos court, période d'activité ou d'indécision, du repos long où l'animal se couche et dort. La rumination est une activité particulière puisqu'elle peut être effectuée en même temps que le repos ou le déplacement.

3.1.3. Le module spatial et social (S)

Ce module gère les relations sociales et spatiales au sein du troupeau. Il constitue donc une grande partie de mon stage.

Deux types de relations sociales vont influencer la position et le déplacement des animaux : le grégarisme et le leadership.

Dans un troupeau, les animaux ont tendance à rester groupés, à ne pas trop s'éloigner des congénères tout en restant à une distance raisonnable : c'est ce que l'on appelle le grégarisme.

Les animaux ont tendance à suivre un animal particulier du troupeau, appelé leader. Le leader provoque des déplacements de groupe, appelés déplacements longs, pour se rendre sur un autre site de la parcelle afin de manger, boire ou se reposer puisqu'il est capable de mémoriser les sites précédemment visités.

Chaque animal possède une mémoire pour enregistrer les meilleurs sites alimentaires visités ainsi que les sites de buvée et de repos. Pourtant seul l'animal leader peut utiliser les données de sa mémoire. Aussi, lorsque le leader veut effectuer une action et que les cellules voisines ne sont pas adaptées à son activité, il consulte sa mémoire pour y trouver un site plus approprié. On parle de déplacement long lorsque le leader change de site pour y effectuer une activité.

La taille de la mémoire est limitée donc si celle-ci est pleine, l'ajout d'un nouveau site s'accompagne de l'oubli d'un site de mauvaise qualité, celui dont le coefficient log(nb_visites+1)*qualité est le plus faible. Les sites oubliés les premiers sont donc ceux de mauvaise qualité et visités les moins souvent. L'oubli est effectué en modifiant la qualité de tous les sites connus en fonction de la valeur moyenne de la parcelle estimée par l'animal.

Deux types de sites sont stockés dans la mémoire de l'animal : les sites alimentaires et les sites non alimentaires, c'est-à-dire les sites de repos et de buvée. Lorsque le leader veut boire ou se reposer, il choisit le site approprié le plus proche. Par contre, pour manger, plusieurs paramètres rentrent en compte : la position du site par rapport à l'animal, la qualité mais aussi l'aire du site. Le site le plus attractif est celui dont le coefficient qualité*aire/distance est le plus élevé.

Lorsque l'animal retourne sur un site visité, une mise à jour de la qualité du site a lieu et la forme du site est recalculée s'il y a eu une agrégation ou une désagrégation. Une agrégation consiste à fusionner deux sites voisins ayant une qualité nutritionnelle pratiquement équivalente. On ajoute au premier site la surface du second que l'on détruit. Une désagrégation consiste à séparer un site en deux sites distincts. Une désagrégation est effectuée lorsque la nouvelle qualité du site mémoriel est très différente de la qualité moyenne du site auquel il appartient.

3.1.4. Le module technique

Ce module regroupe toutes les classes qui ne sont comprises dans aucun des modules biologiques décrits ci-dessus et qui ont trait à la simulation : la gestion du temps, les entrées/sorties et les aspects graphiques de la simulation.

3.2. Etat d'avancement de la programmation

3.2.1. Module végétal

Ce module, entièrement développé au centre INRA de Clermont-Ferrand – Crouël, est terminé, il possède trois fonctionnalités qui sont la mise à jour de la parcelle (croissance et sénescence), la prise en compte de la défoliation de la parcelle et la mise à disposition de certaines propriétés des cellules à l'animal.

3.2.2. Module animal

Le module animal est opérationnel, il gère l'état interne et les déplacements de proximité de l'animal grâce à un générateur de trajectoires. Ce générateur permet de construire la trajectoire suivie par l'animal sur la parcelle pendant une phase de défoliation. Lorsque l'animal se situe sur une cellule, il a le choix entre les quinze cellules de son champ de vision. Une probabilité est calculée pour chacune de ces cellules, elle dépend à la fois de la qualité, de la distance et de la direction à suivre pour atteindre la cellule.

L'animal peut effectuer différentes activités. Deux algorithmes de décision d'activité existent : un premier très simplifié et un autre, plus évolué : l'automate décisionnel, qui a été codé par Lucie Masson lors de son stage en 2001 [MASSON 2001].

L'automate décisionnel permet de rajouter les actions boire et repos long. Les actions de buvée et de repos long ne peuvent pas être effectuées à n'importe quel endroit de la parcelle, elles nécessitent la présence respectivement d'un point d'eau et d'une aire de couchage. On aurait pu penser que les positions de ces sites particuliers seraient précisées à la création de la carte de la parcelle mais ce n'est pas le cas. Ces sites sont seulement connus des animaux du troupeau car stockés dans leur mémoire et ne peuvent pas être défoliés : ils servent uniquement à la buvée ou au repos.

Cet algorithme permet de corriger la vision trop simplifiée implémentée par François Guerry [GUERRY 2000] et représentée par la Figure 1.

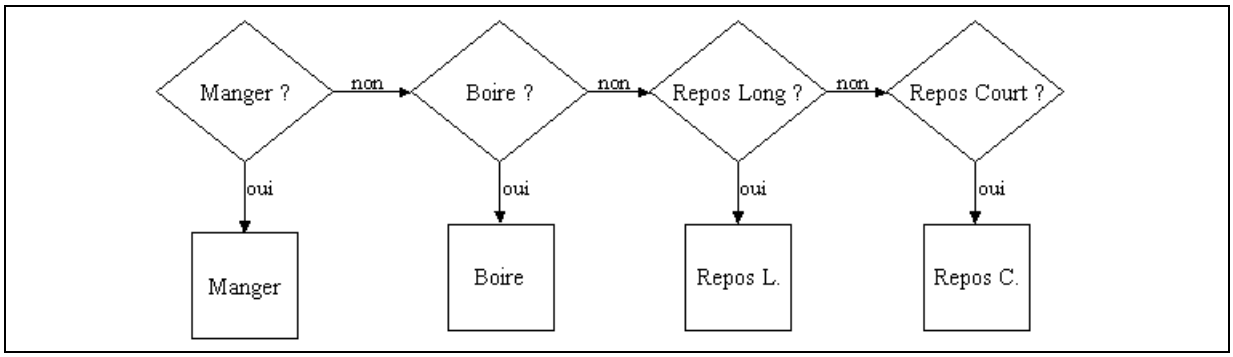


Figure 1 : Première version de l'algorithme de décision

Dans cette implémentation, l'activité manger a priorité. Ainsi, l'animal mangera tant qu'il aura faim et ne pourra effectuer aucune autre action tant qu'il ne sera pas rassasié. L'animal ignore donc la soif et la fatigue tant qu'il a faim, ce qui pose problème puisqu'il peut mourir de soif alors qu'il a un point d'eau à proximité.

L'automate décisionnel rend l'animal opportuniste, c'est-à-dire qu'il va par exemple profiter de son passage près d'un point d'eau pour boire, même s'il a faim et pas particulièrement soif. Dans cet algorithme, le choix de l'activité n'est plus seulement effectué en fonction des besoins biologiques de l'animal mais aussi en fonction de sa position sur la parcelle. Ces deux paramètres, la position et le besoin, permettent de calculer une probabilité pour chaque action puis un tirage aléatoire est effectué.

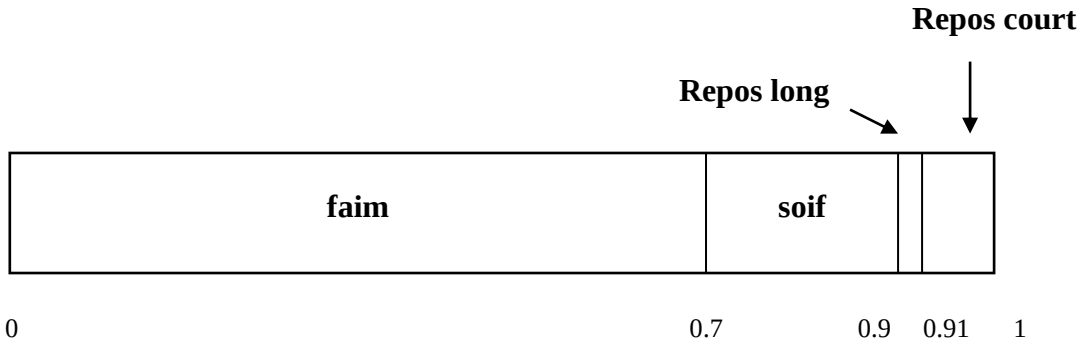
Mais prenons un exemple :

On a obtenu les priorités suivantes :

faim	=	1
soif	=	0.3
repos long	=	0.01

On ramène ces priorités entre 0 et 1 en les divisant chacune par total = (faim + soif + repos long)*1.1 pour laisser une faible chance à l'animal de n'effectuer aucune des trois actions et donc de choisir le repos court, action par défaut. On obtient :

faim	=	0.7
soif	=	0.2
repos long	=	0.01
repos court	=	0.09



On tire ensuite une valeur entre 0 et 1, 0.505 dans notre exemple. D'après le graphique, on peut voir que pour une valeur de 0.505, l'activité manger sera choisie.

3.2.3.Module spatial et social

Le module social est encore peu avancé puisque les relations entre les animaux ne sont pas gérées. Cependant, le processus de mémorisation a été implémenté par Lucie Masson. Deux méthodes sont confrontées : la méthode des contours et la méthode de la carte pixélisée. Ces deux méthodes sont sensiblement équivalentes en terme de mémoire occupée et de temps de réponse, c'est pourquoi aucune des deux n'a pu être pour le moment préférée à l'autre.

3.2.3.1. Méthode des contours

Cette méthode consiste à mémoriser la trajectoire suivie par l'animal durant une phase de défoliation. L'animal ne peut pas mémoriser des positions avec une très grande précision, la surface mémorisée est donc le contour de la trajectoire élargie de trois cellules, c'est-à-dire le champ de vision de l'animal lorsqu'il pâture, mais cette imprécision peut être modifiée.

La structure de données utilisée est présentée dans la Figure 2. Pour chaque ligne, on mémorise la colonne de début et la colonne du fin de la forme du site. Les lignes de la mémoire sont chaînées entre elles et chaque ligne contient au moins une colonne puisque les lignes vides ne sont pas représentées.

Un inconvénient de cette méthode est la place prise en mémoire. Pour réduire cet espace de stockage, on ne mémorise pas toutes les lignes. On mémorise seulement les lignes ayant une différence significative entre elles (plus de six cellules) en conservant toujours la première et la dernière.

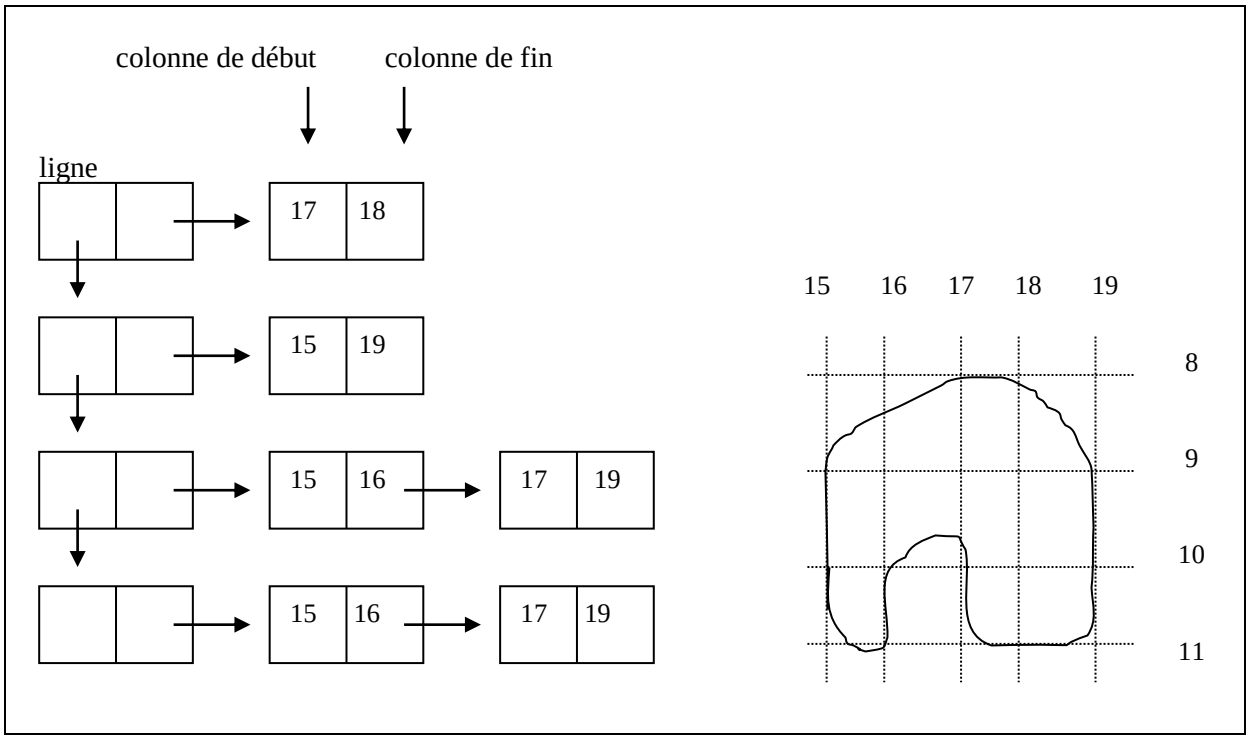


Figure 2 : Structure de données utilisée dans la méthode des contours

Analyse statique :

Le diagramme de classes de la Figure 3 représente la hiérarchie des classes utilisées dans la méthode des contours.

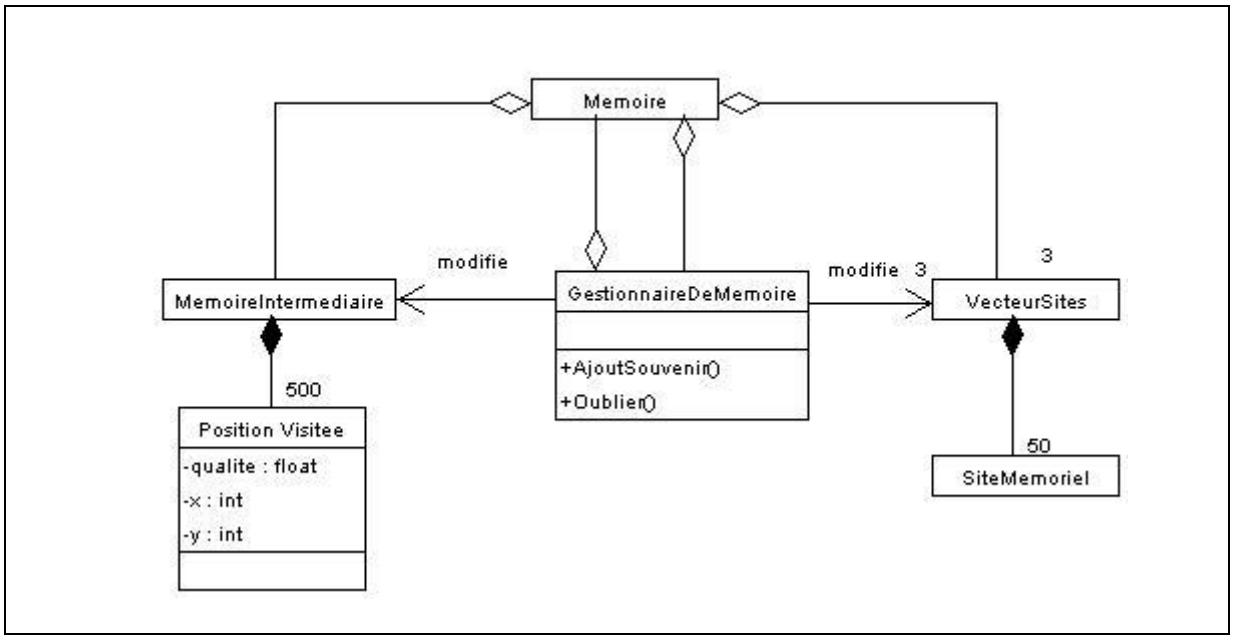


Figure 3 : Diagramme de classes général de la mémoire spatiale

La mémoire contient trois vecteurs de sites mémoriels : un pour les sites alimentaires, un pour les sites de repos et le dernier pour les sites de buvée. Les sites non alimentaires sont mis en mémoire au début de la simulation et n'évoluent pas au cours du temps. Le gestionnaire de mémoire contient toutes les méthodes utilisées par la mémoire, il manipule la mémoire et la mémoire intermédiaire.

La mémoire intermédiaire sert à stocker les informations pendant la mémorisation d'un site alimentaire avant la mise à jour de la mémoire. Elle contient un objet de type *PositionVisitee* pour chaque cellule visitée par l'animal. On connaît ainsi la position et la direction de chaque cellule qui permettent de définir le contour du site ainsi que la qualité qui permet de calculer la qualité moyenne du site végétal.

Analyse dynamique :

Dans la méthode des contours, une mise à jour de la mémoire intermédiaire est effectuée à chaque défoliation. L'animal arrive sur ne nouvelle cellule, il y a donc création d'un nouvel objet *PositionVisitee*. On teste dans un premier temps si l'animal a changé de site végétal, ce qui est le cas lorsque le rapport entre la qualité moyenne du site en cours de création et la qualité de la cellule est trop différent de un. Un nouveau site mémoriel est alors créé à partir des données de la mémoire intermédiaire.

Si la mémoire est pleine, l'ajout d'un site mémoriel entraîne la suppression du plus mauvais souvenir. Le nouveau site est ensuite créé puis ajouté à la mémoire dans le vecteur de sites alimentaires. Le gestionnaire de mémoire recherche les agrégations et désagrégations possibles : elles concernent tous les sites alimentaires avec lesquels ce nouveau site a des recouvrements. Si la qualité des deux sites est semblable il y a agrégation et sinon désagrégation. Enfin, il faut penser à vider la mémoire intermédiaire pour la défoliation suivante.

3.2.3.2. Méthode de la carte pixélisée

Cette méthode utilise une carte pour représenter le contenu de la mémoire. Cette carte, que l'on appelle carte pixélisée, est moins précise que la carte de la parcelle et contient des macro-cellules, c'est-à-dire des ensembles de 8x8 cellules. Comme dans la méthode des contours, la mémoire contient trois vecteurs de sites alimentaires, de repos et de buvée. Chaque site alimentaire est décrit par une qualité, un nombre de visites et une surface mais contient aussi un pointeur sur une liste de pointeurs sur la carte.

Analyse statique :

Le fonctionnement général de cette méthode, tel qu'il est décrit dans la Figure 3, est le même que pour la méthode des contours. On retrouve la classe centrale *Memoire* qui contient trois vecteurs de sites : alimentaires, de repos et de buvée. De même, le gestionnaire de mémoire contient les fonctions servant à manipuler la mémoire.

La mémoire intermédiaire évite de mettre à jour la mémoire à chaque défoliation. Chaque cellule défoliée est enregistrée dans la mémoire intermédiaire, ses informations permettent de créer un nouveau site végétal dans la mémoire. Ainsi, les agrégations ou désagrégations inutiles qui auraient lieu si la mémoire était mise à jour à chaque défoliation sont évitées.

La carte pixélisée pointe sur les sites mémoriels, ce qui permet de retrouver rapidement les sites voisins sans avoir à parcourir tous les sites. Chaque macro-cellule de la carte pointe donc vers le site auquel elle appartient. Chaque site contient de plus une liste chaînée de pointeurs vers les macro-cellules de la carte, ce système de double pointeur rendant les recherches plus rapides en cas d'agrégation.

Analyse dynamique :

Dans cette méthode, la mise à jour de la mémoire intermédiaire à chaque défoliation est effectuée de la même façon que pour la méthode des contours. La seule différence concerne la modification de la mémoire lorsque l'animal quitte un site végétal. Il faut traiter chaque macro-cellule stockée dans la mémoire intermédiaire individuellement. On recherche tout d'abord si la macro-cellule défoliée par l'animal est connue, c'est-à-dire si elle appartient à un site déjà enregistré dans la mémoire. Si elle est connue, on calcule la différence entre la qualité actuelle de cette macro-cellule après défoliation et la qualité du site stocké dans la mémoire en valeur absolue. Si cette différence est jugée acceptable, on mettra à jour la qualité et le nombre de visites du site, sinon le site est désagrégé en retirant la macro-cellule traitée. Par contre, si la macro-cellule n'est pas connue ou qu'elle a été retirée du site, on cherche une agrégation possible avec un autre site de la mémoire et sinon on crée un nouveau site avec les informations de la mémoire intermédiaire.

3.2.4.Module technique

Ce module prend en compte tout ce qui a trait à la simulation : la gestion temporelle, graphique et les entrées/sorties.

3.2.4.1. L'échéancier

L'échéancier codé par François Guerry n'est pas encore inclus dans le simulateur. Dans la vision avec l'échéancier, une simulation par évènements discrets a été choisie, chaque événement correspondant à une action d'un animal. L'unité de temps de base de cet échéancier est la seconde.

L'approche par évènements discrets correspond à un mode de gestion de temps du simulateur. Elle revient à découper le temps selon les actions des entités : chaque entité va donc être vue comme un objet subissant divers évènements. Chaque événement va provoquer un comportement particulier de l'entité. Ainsi, lorsqu'on traite une activité, on va vouloir enchaîner sur d'autres. Pour cela, chaque activité va être déclenchée par un événement, qui lui-même prévoira de nouveaux évènements pour les activités futures.

Mais pourquoi choisir cette approche au lieu d'une horloge simple qui active les entités au bon moment ? Notre modèle biologique comporte deux types d'entités très différents : les ruminants et les cellules. Le temps entre deux activations peut être très varié : quelques secondes pour les ruminants, une journée pour les cellules… Gérer tout cela avec une horloge à pas fixe serait alors illusoire puisque le pas de temps serait égal au temps minimal possible entre deux activations, c'est-à-dire de l'ordre de la seconde. Dans ces conditions, chaque entité serait vérifiée à chaque fois, mais activée que très rarement, ce qui est loin d'être optimal. Grâce à l'approche par évènements, le traitement se produit seulement si nécessaire, et les périodes où rien ne se passe sont automatiquement sautées.

Les animaux sont des objets dynamiques capables de décider quand ils vont changer d'état en passant du déplacement à la défoliation, les cellules de végétation sont statiques et peuvent seulement effectuer des actions telles que la croissance quand elles sont déclenchées par un objet dynamique [BEECHAM et FARNSWORTH 1998].

Un objet dynamique possède trois états :

- actif, en train de prendre une décision ;
- suspendu, en attente d'une période de temps finie à la fin de laquelle une décision doit être prise ;
- mort, qui ne participe plus à la simulation.

Comme le montre la Figure 4, seuls le déplacement et la défoliation se déroulent sur une période finie, le choix de cellule est instantané. Alors que l'animal mange ou se déplace, il est en état suspendu en attendant d'être activé à une date fixée dans le futur. Quand ce temps est écoulé, la partie du code qui détermine la décision de l'animal est activée. Aussi, une classe envoie périodiquement (en fin de journée) un signal à chaque cellule pour lui dire de croître et de calculer sa sénescence.

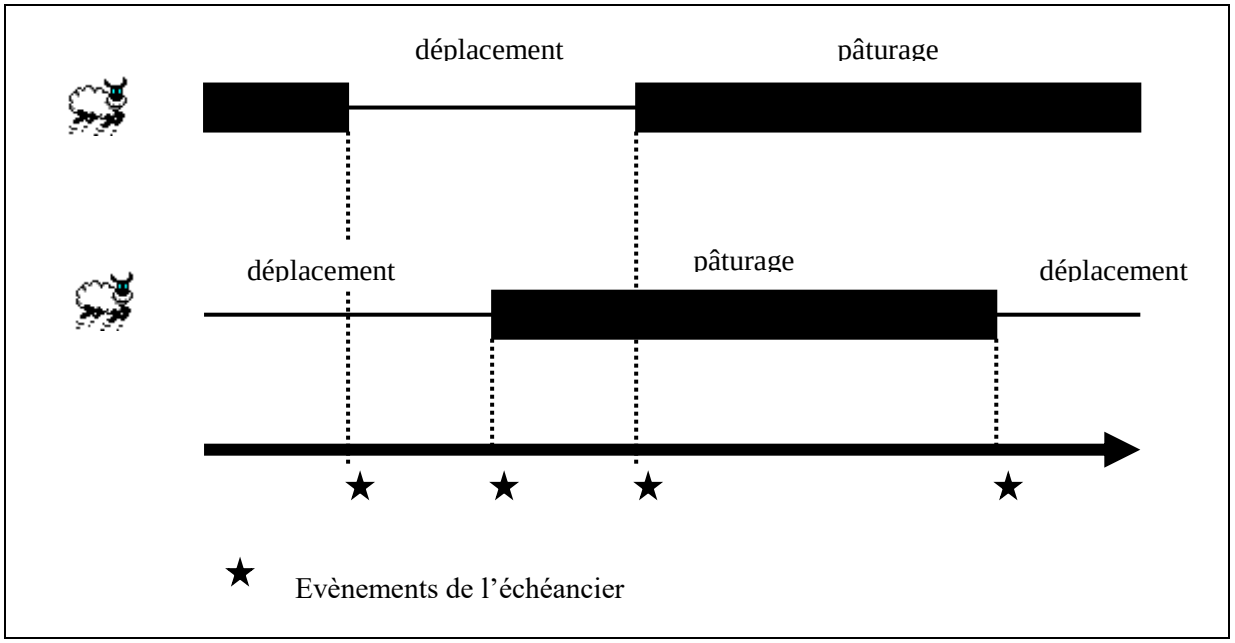


Figure 4 : Schéma expliquant le fonctionnement de l'échéancier

Source : [BEECHAM et FARNSWORTH 1998]

Une simulation par évènements discrets est gérée par un échéancier qui permet de traiter les évènements de manière chronologique, il consulte le premier événement et active l'entité associée qui va alors déclencher un traitement. La simulation se termine lorsque la durée de simulation est écoulée.

La Figure 5 représente le diagramme de classe de l'implémentation de l'échéancier.

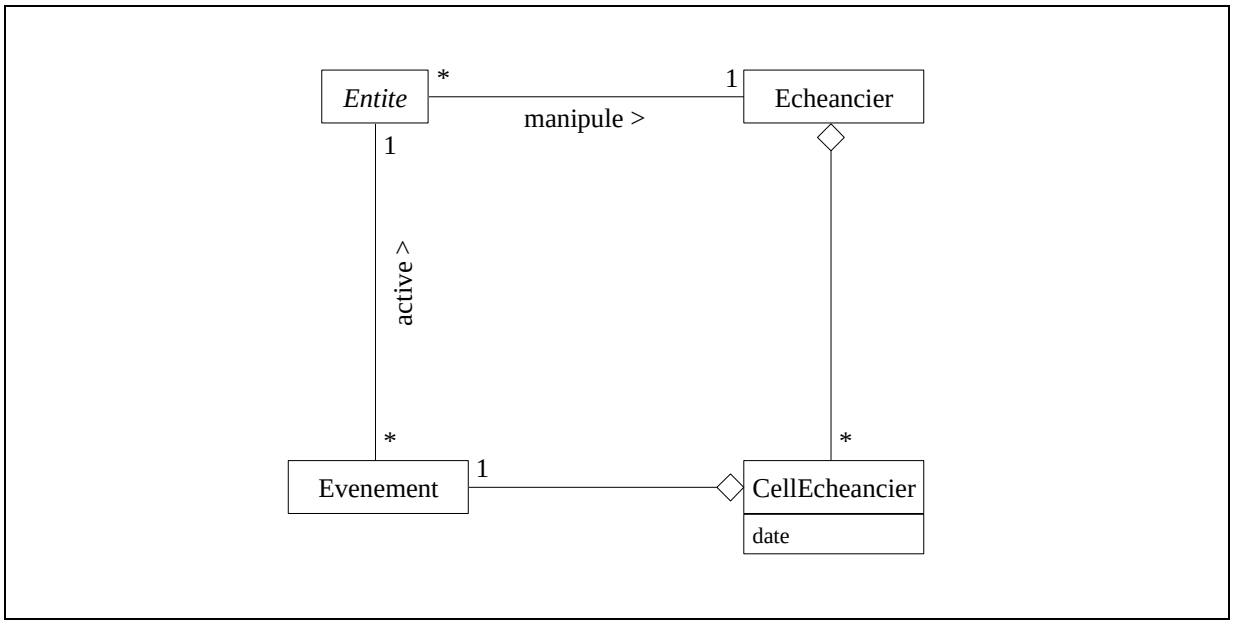


Figure 5 : Diagramme de classes de gestion des évènements

On peut voir que l'échéancier contient une liste d'évènements, chacun étant activé à une date donnée. La classe *CellEcheancier* contient donc un événement et une date d'activation. Lorsqu'un événement est exécuté, il active l'entité associée qui va déclencher une action dépendant du type d'événement. Toute entité peut manipuler l'échéancier et notamment insérer et détruire des évènements qui la concernent. La classe *Entite* est une classe abstraite : chaque objet qui aura à modifier, consulter l'échéancier ou qui manipulera des évènements devra dériver de la classe *Entite*.

3.2.4.2. Le générateur de cartes

Le générateur de cartes permet de créer une carte de végétation perçue par l'animal. Un premier programme, appelé *créateur,* construit une parcelle à partir d'un fichier de définition de sites en forme de rectangle. Le *créateur* affecte à chaque site un faciès. Le second programme est appelé *percepteur*, il applique une fonction de perception de l'animal à la carte construite par le *créateur*. On obtient en sortie une carte où chaque cellule est représentée par une unique valeur correspondant à la représentation que se fait l'animal de la qualité de la cellule. La carte obtenue après utilisation du percepteur est directement utilisée par la visionneuse.

3.2.4.3. La visionneuse

La visionneuse est un outil qui sert à visionner la parcelle et les déplacements de l'animal avec trois échelles différentes.
L'échelle 1 est la vue détaillée, elle donne la vision du ruminant. Elle permet d'observer les quinze cellules du champ de vision de l'animal ainsi que leurs valeurs.

L'échelle 2, ou vue intermédiaire, permet d'observer les déplacements de l'animal sur une partie de la parcelle.
L'échelle 3 consiste quant à elle à représenter l'évolution du troupeau sur des parcelles de grande taille.

3.2.4.4. L'interface utilisateur

L'interface utilisateur est le dernier outil développé pour faciliter l'utilisation du simulateur. Elle permet de gérer, de manière dynamique, les paramètres en entrée et les données en sortie du simulateur. Une succession d'étapes permet de saisir tous les paramètres de la simulation, ainsi l'utilisateur n'a plus à modifier tous les fichiers en entrée en les éditant, l'interface se charge de tout. L'interface gère aussi les réplications et elle se charge d'appeler le simulateur, par un mécanisme de librairie, pour effectuer la simulation.

3.2.5.Le prototype actuel

Après un développement de chaque programme de façon indépendante, les applications ont été interconnectées en plusieurs étapes par Guillaume Martin pour donner plusieurs prototypes du simulateur [MARTIN 2002]. Le dernier prototype regroupe deux modules biologiques, le module animal et le module végétal ainsi que les utilitaires suivants : le générateur de cartes, la visionneuse et l'interface utilisateur. Ce prototype n'est pas complet : premièrement, la partie décisionnelle du module animal est la version simplifiée et deuxièmement on ne peut simuler qu'un seul animal, la gestion d'un troupeau n'est pas encore supportée. La simulation se déroule de façon séquentielle, par appels successifs de fonctions, l'échéancier n'est pas utilisé.

La Figure 6 montre le fonctionnement général de ce dernier prototype du simulateur. L'interface utilisateur assure la saisie des valeurs des paramètres généraux de la simulation et de la carte de la parcelle sur laquelle évolue l'animal. L'interface appelle ensuite le *createur* qui crée une carte directement exploitable par le simulateur. Cette carte peut aussi être utilisée par le *percepteur* pour construire une nouvelle carte où chaque cellule est représentée par une valeur. La visionneuse prend en entrée cette carte pour montrer la parcelle et les déplacements de l'animal. Enfin, le simulateur permet d'obtenir de nombreuses sorties, concernant aussi bien les actions de l'animal que l'évolution de la végétation. Les diagrammes de classes du modèle sont listés dans l'annexe C.

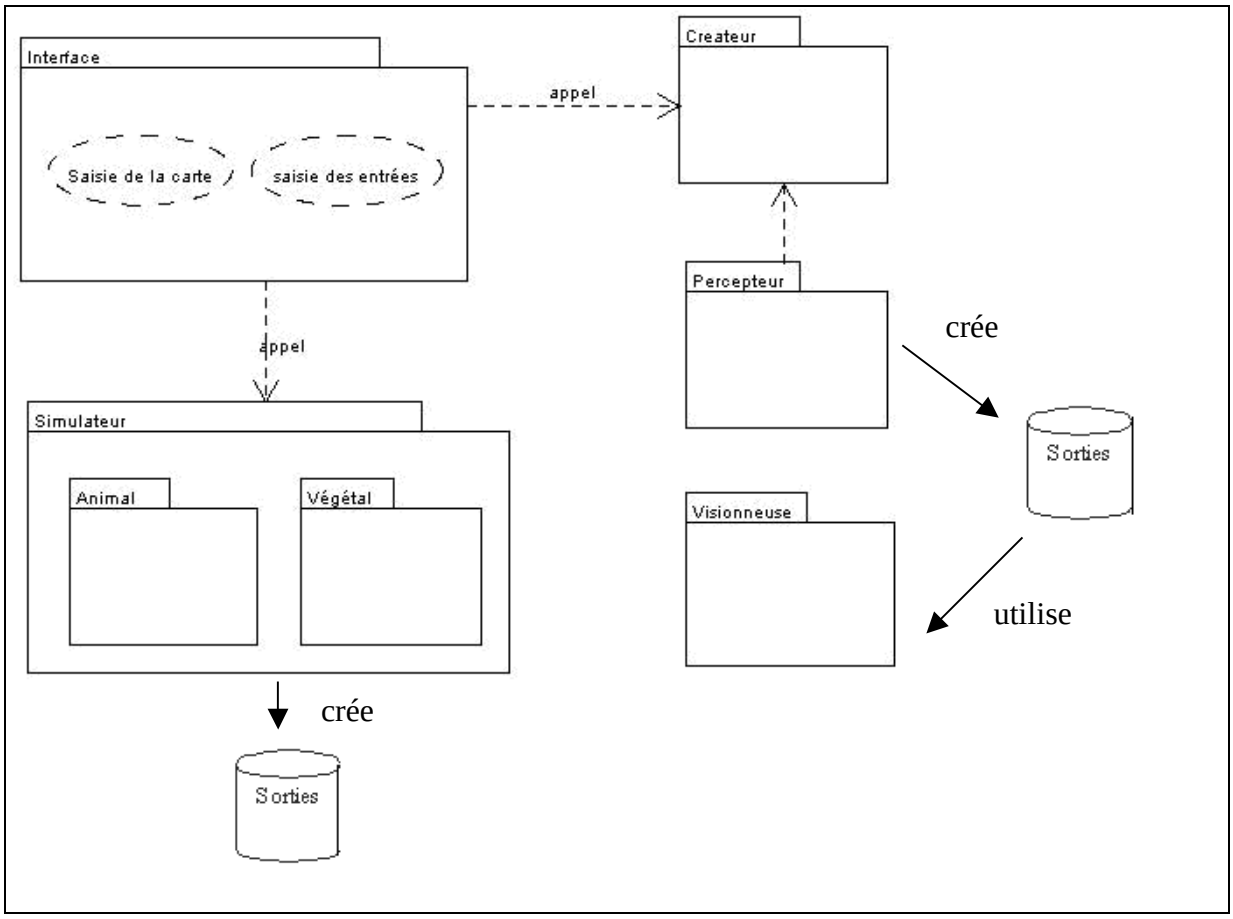


Figure 6 : Diagramme de paquetages montrant le fonctionnement général du simulateur

Partie II : Phase de conception

1. Modifications et corrections apportées au prototype animal-végétation

Comme un travail de validation se faisait en parallèle de mon stage, des erreurs de fonctionnement sont apparues, certaines ont pu être corrigées grâce à un travail commun avec Magali Jouven, la stagiaire chargée de la validation. Le travail effectué sur le prototype m'a permis de me familiariser rapidement avec le fonctionnement général et l'arborescence du simulateur qui comporte un grand nombre de fichiers source. Dans cette partie, je décrirais rapidement les modifications et corrections apportées.

1.1. Gestion du début et de la durée de simulation

Auparavant, le module animal et le module végétal avaient leur propre durée de simulation, pourtant seule celle du module animal était prise en compte et correspondait à la durée de simulation. Par contre, le jour de début de la simulation était paramétré dans le module végétal. Ce fonctionnement pouvait s'avérer un peu déroutant pour un utilisateur, c'est pourquoi il a été décidé de différencier les débuts et durées des simulations végétale et animale. Cette fonctionnalité s'avérait aussi très utile pour Pascal Carrère (centre INRA de Clermont-Ferrand – Crouël) qui est surtout intéressé dans un premier temps par la simulation de la partie végétale. Par contre, la simulation animale ne peut fonctionner sans la partie végétale puisqu'il faut que les mises à jour des biomasses de chaque compartiment aient lieu en fin de journée pour simuler la pousse de l'herbe et la sénescence et que l'impact sur la végétation d'une défoliation soit immédiatement pris en compte.

La simulation peut donc être commencée à n'importe quelle date, à condition que les paramètres de la végétation correspondent à la situation courante : si on commence la simulation en été, il faut que les paramètres tels que la hauteur de l'herbe ou les biomasses des compartiments verts et secs correspondent à ceux que l'on pourrait trouver en été.

Il y a donc plusieurs cas possibles de simulation, tels qu'ils sont illustrés à la Figure 7 :

- Une simulation de la végétation seule pendant toute la durée de simulation ;
- Une simulation avec animal pendant toute la durée de simulation ;
- Une simulation avec animal d'abord puis de la végétation seule ;
- Une simulation de la végétation seule d'abord, puis de l'animal à partir d'une date N ;
- Une simulation de la végétation seule d'abord, puis de l'animal à partir d'une date N, puis à nouveau de la végétation seule.

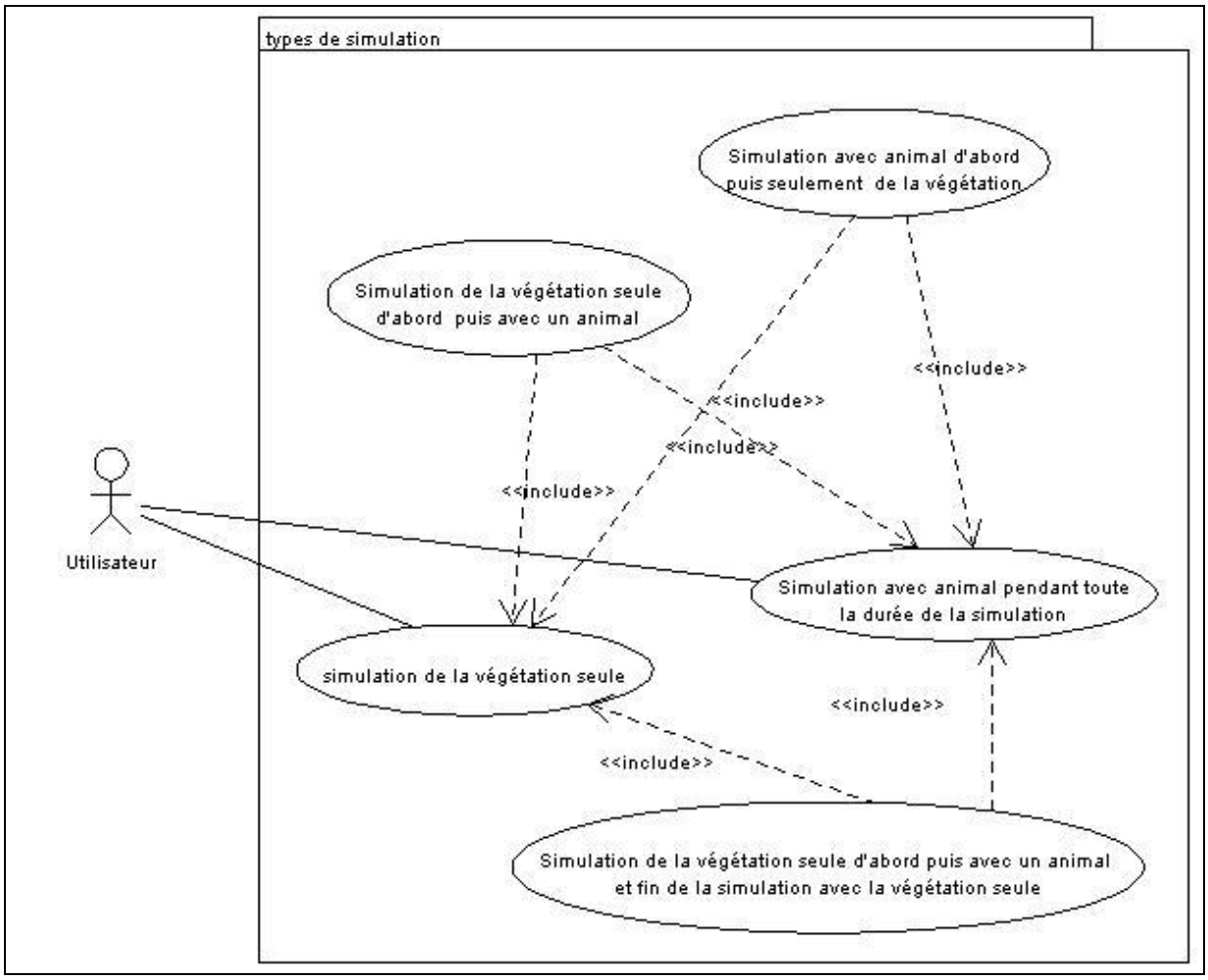


Figure 7 : Diagramme de cas d'utilisation des types de simulation

Dans le cas où on ferait d'abord tourner le module végétal seul, on a la possibilité de demander l'ensemble des sorties fichiers prévues pour cette période, ou seulement celles de la veille de l'arrivée de l'animal.

Le module végétal a été adapté à ces modifications pour la lecture des températures de chaque jour de l'année. Il n'y a désormais lecture que des températures nécessaires à la simulation, les autres températures du fichier *environnement.csv* sont ignorées. Aussi, la somme des températures au premier jour de simulation peut être renseignée mais si elle ne l'est pas, elle est calculée. L'annexe A décrit plus précisément l'ensemble des modifications apportées dans les deux fichiers en entrée *param.csv* et *environnement.csv*.

1.2. Erreurs dues à l'interconnexion des modules A et V

Les modules animal et végétal ayant été développés de manière parallèle, certaines incohérences sont apparues lors de leur connexion, notamment en ce qui concerne la défoliation des cellules par l'animal.

Dans un premier temps, les valeurs des quantités défoliées n'étaient pas les mêmes dans chaque module à cause d'unités différentes : le module animal traitait des g/cellule alors que le module végétal ne considérait que des g/m². Or la surface des cellules n'était pas toujours égale à 1 m², elle correspondait à une variable S paramétrée dans un fichier. Il fallait donc que le module végétal divise toutes les valeurs de quantités prélevées en g/cellule par S pour obtenir des valeurs en g/m².

D'autre part, lors de la défoliation d'une cellule par l'animal, la biomasse des compartiments de la cellule défoliée n'était pas mise à jour, c'est-à-dire que la quantité de matière disponible sur la cellule n'était pas diminuée. Cela se traduisait par le fait que l'animal mangeait beaucoup trop sur la même cellule puisqu'il pouvait manger plusieurs fois sans que la quantité diminue et donc la qualité de la cellule était constante. Aussi lorsqu'une mise à jour des cellules était effectuée en fin de journée, on obtenait fréquemment une quantité négative de matière disponible sur la cellule. Une mise à jour des compartiments a donc été effectuée après chaque défoliation et non plus seulement en fin de journée. La mise à jour de la parcelle chaque soir consiste donc seulement à gérer la pousse et la sénescence de l'herbe de la journée.
Aussi, un test a été rajouté pour vérifier que la quantité prélevée par l'animal ne dépasse pas la biomasse disponible sur la cellule.

1.3. Problème avec l'interface

Le travail de validation du prototype a aussi révélé que l'interface utilisateur codée lors du projet 2003 [PORTAL et SEGUY 2003] ne fonctionnait pas sur une durée de simulation importante (supérieure à vingt jours) à cause d'un problème de taille de pile. L'interface, codée en langage Java, utilise un système de librairie pour appeler le simulateur codé en C++. Après plusieurs tentatives infructueuses pour augmenter la taille de la pile réservée à l'exécution des librairies (avec l'option `-ss` de la commande `java -jar`), une nouvelle solution a été retenue : la suppression de l'appel de la librairie remplacé par l'appel de l'exécutable C++ avec la commande `Runtime.getRuntime().exec(« exécutable fic_archive rep_sorties »)`.

Le simulateur ne reçoit plus désormais tous les paramètres de la simulation mais les lit dans le fichier archive créé par l'interface qui répertorie tous les noms de fichiers en entrée et les paramètres généraux de la simulation tels que les dates de début et les durées des simulations animale et végétale. La classe *FichierConfigGestion*, qui fonctionne sur le même principe que les classes *FichierConfigRuminant* et *FichierConfigTroupeau* décrites plus loin, a été rajoutée pour lire tous les paramètres présents dans le fichier d'archive. De plus, l'exécutable doit aussi connaître le répertoire où seront stockées toutes les sorties pour le cas où il y ait plusieurs réplications : dans ce cas le simulateur appelle l'exécutable autant de fois qu'il y a de réplications, avec à chaque fois un répertoire de sortie différent.

Cette solution a résolu le problème mais la simulation s'arrête désormais au bout de quatre-vingt jours environ, le problème étant dû cette fois au simulateur lui-même. Cette erreur devrait être corrigée dans le cadre d'un travail de vérification.

2. De l'animal au troupeau

La prise en compte du module S dans le simulateur permettra d'effectuer une simulation avec un troupeau et non plus avec un seul animal. Ceci impose une analyse de ce module, ainsi que l'utilisation des programmes échéancier et mémorisation déjà développés. On en profitera pour améliorer le processus décisionnel grâce à l'automate.

2.1. Analyse biologique

Les herbivores se caractérisent par leur grégarisme : ils forment une entité et constituent un véritable groupe [DUMONT et BOISSY 1999]. Le groupe se définit par l'existence d'une attraction et d'une répulsion mutuelle entre les individus qui se concrétisent par le développement de nombreuses relations sociales. Les relations d'affinité assurent une plus grande tolérance entre animaux et se caractérisent par une diminution des conduites agressives et par une plus grande proximité spatiale.

La diminution de la ressource alimentaire par l'exploitation du site amène le groupe à le quitter. Un animal qui choisirait de quitter un site où il ne trouve plus une nourriture qui le satisfait, peut y être maintenu par les autres individus qui s'y alimentent encore. La répartition spatiale des animaux résulte de la cohésion du groupe qui détermine si l'attraction sociale l'emporte sur les préférences alimentaires individuelles.

Les partenaires sociaux influencent non seulement les préférences alimentaires de l'animal et son exploitation des sites alimentaires, mais ils orientent également la manière dont l'animal occupe l'espace.

Les relations sociales régulent les relations qu'entretiennent les animaux entre eux et assurent le fonctionnement du groupe, la reproduction et l'élevage des jeunes. Parmi ces relations, ce sont les relations de « dominance-subordination » qui structurent pour une grande part la vie au sein du groupe. Ainsi l'individu dominant accède aux différentes ressources de manière non contestée sans devoir diriger ni coups ni menaces envers le subordonné.

Les mouvements du groupe dans l'espace font appel à des relations de « leadership », l'initiation des déplacements étant généralement le fait des mêmes animaux.

Le leadership pourrait être la conséquence des affinités développées dans le groupe, les individus ayant tendance à suivre leurs partenaires familiers. En effet, la vie en groupe constitue une stratégie antiprédatrice efficace.

Le groupe assure ainsi aux herbivores une meilleure sécurité individuelle du fait du bénéfice apporté par une surveillance commune, et permet à chacun de ses membres de réduire les comportements de vigilance. La vie en groupe serait également un moyen efficace de maintenir la qualité des ressources pâturées. Enfin, l'organisation en groupe des herbivores résulterait de la répartition des ressources pâturées : les herbivores formeraient des groupes du fait de l'agrégation des zones de végétation qu'ils exploitent. Il est fréquemment observé que les distances inter-individuelles augmentent lorsque les disponibilités alimentaires diminuent.

Des différences inter-individuelles de sociabilité ont été évoquées pour expliquer le leadership. Les mouvements du groupe seraient initiés par les individus les moins grégaires qui pâturent fréquemment à une plus grande distance des autres.

Ainsi selon la nature du déplacement (volontaire ou provoqué), les leaders n'ont pas le même statut social. Chez les ovins et les bovins, ce sont plutôt des animaux âgés et donc plus expérimentés qui initient les déplacements du troupeau, et cela lors de déplacements volontaires ou provoqués. Ces animaux leader facilitent la découverte de nouveaux sites alimentaires par les autres animaux du groupe. Selon une étude réalisée dans un labyrinthe, dans 61 % des cas, le leader a déjà découvert un nouveau site alimentaire lorsqu'un des autres animaux l'atteint à son tour pour la première fois. Plus tard, ceux-ci retrouvent le site alimentaire sans l'aide du leader. Plus un leader est expérimenté, plus il pourrait faciliter la découverte de nouveaux sites par les autres membres de son groupe.

2.2. Différenciation des animaux

Dans un premier temps, il est important de pouvoir donner des paramètres en entrée différents pour chaque animal et aussi de pouvoir trier les sorties obtenues.

2.2.1. Paramètres en entrée

Parmi les variables paramétrées en entrée concernant l'animal et le générateur de trajectoires, il faut identifier les variables pour lesquelles la valeur sera commune au troupeau et celles dont la valeur sera propre à chaque individu.

Le fichier *troupeau.cfg* donne les caractéristiques communes au troupeau. On peut voir que la plupart sont issues du fichier de configuration du générateur de trajectoires à part la *fonction DN* qui n'était auparavant paramétrée dans aucun fichier. Cette fonction est utilisée pour moduler la motivation à ingérer de l'animal en fonction de l'heure de la journée, ainsi l'animal mangera beaucoup en début de matinée et en fin d'après-midi mais très peu la nuit. Il pouvait donc s'avérer nécessaire de modifier sa valeur. Ces variables sont lues par la classe *FichierConfigTroupeau* qui remplace la classe *FichierConfigChoix* devenue obsolète.

Les variables propres à chaque animal sont quant à elles listées dans le fichier *ruminant.cfg*. Les caractéristiques de chaque animal sont séparées par des caractères ✶ dans le fichier. Le parcours du fichier doit se faire de manière séquentielle, en lisant les caractéristiques animal par animal, selon le numéro de l'animal dans le troupeau. L'algorithme de la classe *FichierConfigRuminant* permettant le parcours de ce fichier arrête la lecture dès qu'il trouve un caractère ✶ en début de ligne.

Certaines variables qui étaient présentes dans le fichier *ruminant.cfg* ne se retrouvent dans aucun des deux fichiers de configuration. Ces variables de rumen dépendent du poids de l'animal et sont donc désormais calculées grâce à des formules appropriées.

Le contenu des fichiers *ruminant.cfg* et *troupeau.cfg* est décrit plus précisément en Annexe A.

2.2.2. Tri des sorties animales

Pour que le simulateur soit utilisable avec plusieurs animaux, il était important de pouvoir trier les sorties par animal. Ainsi, si les paramètres en entrée varient selon les animaux, leur influence pourra être observée dans les sorties.

Le choix retenu a été de créer un répertoire par animal. Mais le répertoire conçu pour chaque animal ne doit contenir que les sorties animales, les sorties végétales étant regroupées dans un répertoire *Vegetation* prévu à cet effet. Dans l'arborescence illustrée par la Figure 8, la simulation a été lancée avec deux animaux. Les sorties animales sont contenues dans les deux sous-répertoires *Animal1* et *Animal2* du répertoire *Troupeau* tandis que les sorties végétales sont stockées dans le répertoire *Vegetation*.

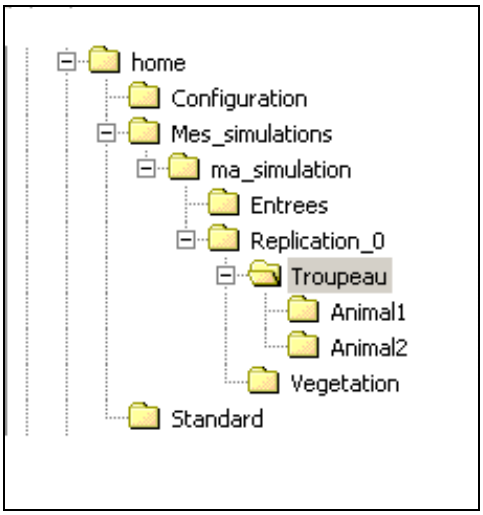


Figure 8 : Arborescence choisie pour trier les sorties

La création des sorties animales dans le répertoire approprié s'est avérée moins simple que prévue. L'ajout de l'interface utilisateur dans le prototype avait conduit à la création d'une nouvelle classe dans le simulateur, la classe *Parametres*. Cette classe consiste à stocker tous les paramètres généraux de la simulation, le nom, les durées, mais aussi les chemins de tous les fichiers d'entrée/sortie. Cette classe est donc utilisée par toutes les classes du module animal et du module végétal qui écrivent ou lisent dans un fichier.

Pourtant, il ne suffisait pas de changer un nom de répertoire dans la classe *Parametres* pour que tout fonctionne correctement. Puisque de nombreuses classes du module *Animal* créaient des fichiers en sortie, il fallait qu'à chaque accès à un nom de fichier de la classe *Parametres*, cette classe sache quel animal était traité pour obtenir le nom de répertoire approprié.

Une solution aurait pu être d'appeler chaque méthode de la classe avec le numéro de l'animal en tant que paramètre. Pourtant, cette solution ne paraissait pas satisfaisante puisqu'elle nécessitait un grand nombre de modifications dans le module *Animal*.

Une autre solution a donc été retenue, elle consiste à rajouter dans la classe *Ruminant* le numéro de l'animal en cours de traitement en tant qu'attribut de classe, cet attribut étant mis à jour lorsque le traitement d'un nouvel animal commence. Ainsi, lors de chaque accès à un des noms de fichiers de la classe *Parametres*, l'accès au nom de fichier dépend de ce numéro.

Le diagramme de séquences de la Figure 9 illustre la solution retenue. Le simulateur crée le troupeau puis le troupeau crée chaque ruminant. Avant de créer chaque animal, il faut que le constructeur de la classe *Troupeau* mette à jour le numéro de l'animal courant, ce qui consiste à appeler la fonction `Ruminant ::setAniCour(numero)`, l'attribut `numero` correspondant au numéro de l'animal dans le troupeau. Cette méthode est également appelée dans le constructeur de la classe *Ruminant* pour permettre l'écriture des entêtes des fichiers de sortie.

Ensuite la simulation peut commencer, le simulateur appelle la méthode `traiteJournee()` de la classe *Echeancier*. Cette méthode, comme son nom l'indique, permet de gérer un jour de simulation. Elle récupère le premier événement de l'échéancier, une action à effectuer par un des animaux, et l'exécute. Le numéro de l'animal doit donc être mis à jour avant l'exécution de cet événement.

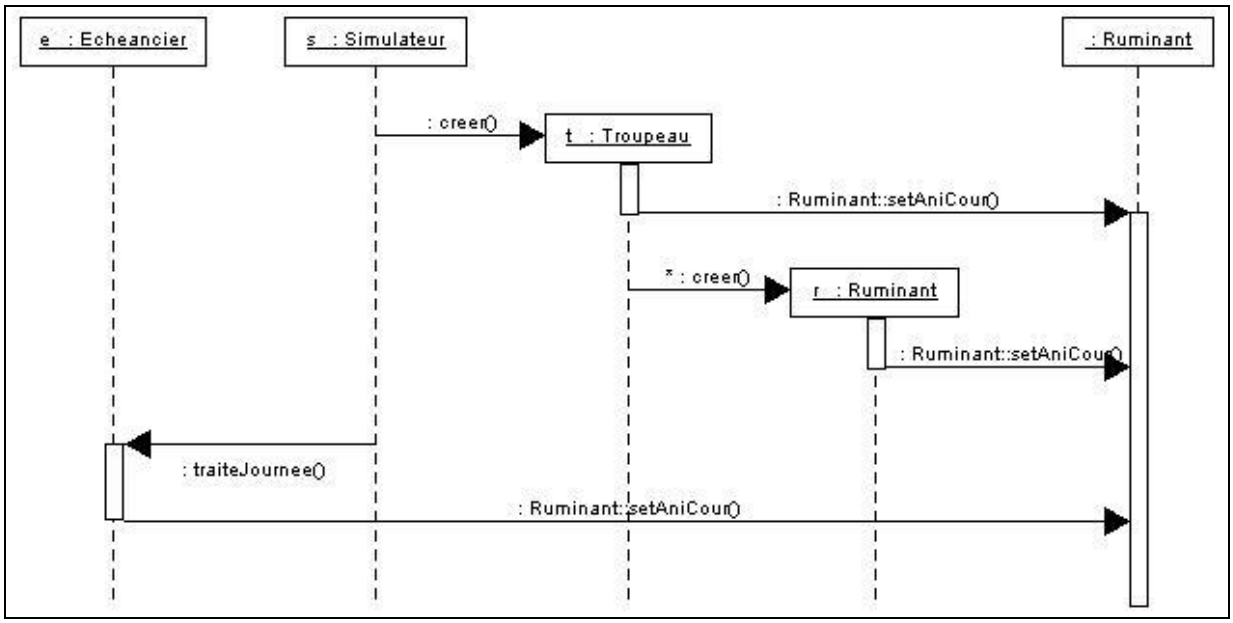


Figure 9 : Diagramme de séquences de tri des sorties animales

2.3. Gestion des collisions sur une cellule

La présence de plusieurs animaux sur la parcelle impose de gérer les collisions, afin que deux animaux ne puissent pas pâturer la même cellule simultanément. Les actions de buvée et de repos sont particulières, les collisions ne posent pas de problème puisque les animaux peuvent boire ou se reposer en étant très proches, les aires de repos et de couchage étant peu nombreuses sur une parcelle.

Comme c'est un événement rare, nous avons décidé dans un premier temps de ne considérer le problème qu'à l'arrivée des animaux sur une cellule. Ainsi lorsqu'un animal termine son déplacement et arrive sur une cellule, il doit vérifier que cette dernière ne soit pas déjà occupée. Si un animal s'y trouve déjà, il choisit une nouvelle cellule parmi les quinze cellules voisines de son champ de vision.

Le premier animal arrivé aura donc accès à la cellule convoitée mais pour cela il faut tenir compte des temps de déplacement qui étaient jusqu'alors instantanés. On distingue deux vitesses de déplacement : tête haute et tête basse. L'animal se déplace tête haute en cas de déplacement long ou pour rejoindre le troupeau ou l'animal leader. Le déplacement tête basse est un déplacement de proximité en pâturant, sa durée est prise en compte dans le temps de pâturage.

La Figure 10 montre l'enchaînement des opérations effectuées dans un déplacement de proximité. L'animal choisit dans un premier temps une cellule grâce au générateur de trajectoires et calcule le temps nécessaire pour se déplacer jusqu'à cette cellule. Un événement *FinDeplacement* est inséré dans l'échéancier à la date courante augmentée du temps de déplacement. Lorsque cet événement est activé, la position de l'animal est d'abord modifiée puis il demande au troupeau si sa nouvelle cellule est occupée par un autre animal. Si elle l'est, il effectue un nouveau choix de cellule en insérant l'événement *ChoixCellule* dans l'échéancier, sinon il défolie la cellule puis effectue un nouveau choix d'activité lorsque la séquence de défoliation est terminée. Tous les temps de déplacements consécutifs de l'animal pour trouver une cellule libre sont recensés dans l'échéancier et dans la durée de la séquence de défoliation. Une description plus complète du fonctionnement de l'événement *FinDeplacement* est donnée en annexe B.

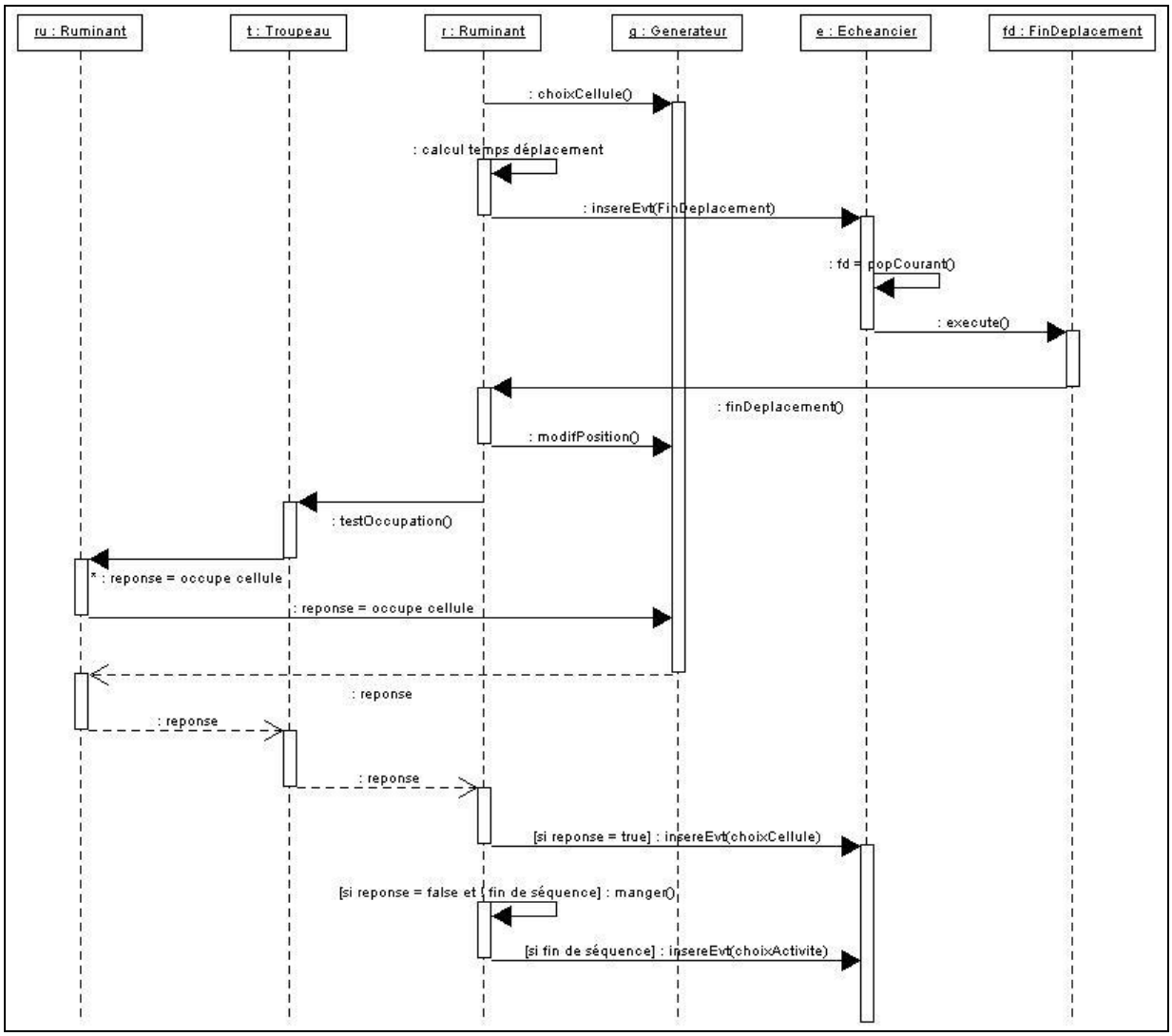


Figure 10 : Diagramme de séquences du déplacement de proximité

2.4. Cohésion sociale

2.4.1. Tests d'éloignement

Les herbivores domestiques étant grégaires, nous avons dû modéliser des règles assurant la cohésion sociale du troupeau . Aussi les animaux ne doivent pas trop s'éloigner les uns des autres, ni être trop proches. Chaque animal vérifiera donc à intervalles de temps réguliers s'il n'est pas trop loin du reste du troupeau ni trop proche d'un autre animal du groupe afin de limiter les phénomènes de compétition alimentaire. La période de vérification de l'éloignement est de l'ordre de deux minutes, c'est celle que j'ai utilisée pour les tests du prototype intégrant la gestion de la cohésion sociale.

Ceci se fera en deux temps, tout d'abord l'animal vérifie qu'il n'est pas trop près de l'animal le plus proche, puis il vérifie son éloignement par rapport au troupeau. Il existe trois manières de calculer la distance entre un animal et le reste du troupeau, la méthode de calcul choisie est paramétrée dans le fichier de configuration *troupeau.cfg* :

- Par rapport à l'animal le plus proche ;
- Par rapport au barycentre des N animaux les plus proches (la distance calculée est alors égale à la moyenne des N distances) ;
- Par rapport à un animal particulier du troupeau, ce qui permet de prendre en compte l'existence d'affinités particulières au sein du groupe, par exemple entre une mère et son jeune.

Pour cela, on se sert de l'événement *TestEloignementTroupeau* qui est détaillé en annexe B. On connaît la position de la cellule sur laquelle se trouve chaque animal. Puisque les cellules sont de forme hexagonale, le calcul de distance est effectué d'un centre d'hexagone à l'autre.

Le test d'éloignement se fait en quatre étapes et utilise une fonction de motivation à se déplacer qui dépend de trois distances *d1Troupeau*, *d2Troupeau* et *d3Troupeau* (Figure 11). Ces distances sont elles aussi paramétrées dans le fichier *troupeau.cfg* et pour les tests j'ai utilisé respectivement les trois valeurs suivantes : 1, 8 et 25 mètres.

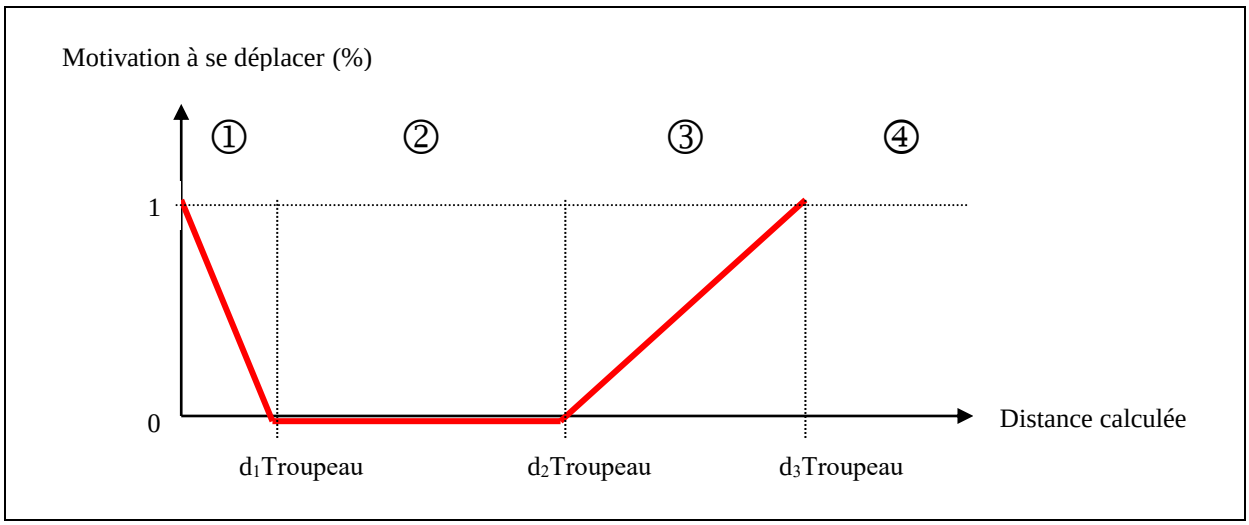


Figure 11 : Différentes situations du test d'éloignement

Dans un premier temps, on calcule la distance entre l'animal et son congénère le plus proche que l'on appelle *dproche*. Si cette distance est inférieure à *d1Troupeau* (situation ① de la Figure 11), l'animal est trop proche, il risque de se déplacer. On calcule alors la motivation à se déplacer de l'animal, qui sera comprise entre 0 et 1 et calculée en fonction de *dproche*. Un tirage aléatoire permettra finalement de déterminer si l'animal s'éloigne ou pas.

Si *dproche* est supérieure à *d1Troupeau*, on calcule une nouvelle distance *dtroupeau* entre l'animal et le reste du troupeau à partir d'une des trois fonctions de calcul (animal le plus proche, barycentre des N animaux les plus proches ou animal particulier). Si la distance *dtroupeau* est comprise entre *d1Troupeau* et *d2Troupeau* (situation ②), l'animal est à une distance raisonnable du reste du troupeau et ne cherche ni à s'éloigner, ni à se rapprocher de ses congénères.

Si *dtroupeau* est supérieure à *d3Troupeau* (situation ④), l'animal se rapproche du groupe. Enfin, si *dtroupeau* est comprise entre *d2Troupeau* et *d3Troupeau* (situation ③), on calcule la motivation à se déplacer de l'animal puis un tirage aléatoire a lieu pour savoir si l'animal va se rapprocher.

L'éloignement ou le rapprochement se font par rapport à la position de l'animal le plus proche, du barycentre des N animaux les plus proches ou d'un animal particulier, selon la fonction de calcul de distance choisie, on appellera par la suite cette position P. Il faut aussi mettre à jour la direction vers laquelle l'animal regarde pour le générateur de trajectoires. L'éloignement et le rapprochement sont des déplacements tête haute, ils interrompent toute autre activité commencée. L'animal reprend son activité de pâturage après le déplacement ou choisit une nouvelle activité si la séquence de défoliation est terminée ou si le déplacement a interrompu une action de repos ou de buvée. Chaque activité a une durée, et lorsque cette durée est écoulée, on dit que l'animal a fini une séquence, il doit donc choisir une nouvelle action.

2.4.2.L'éloignement

Les cellules du champ de vision de l'animal se présentent de la façon suivante (cf. Figure 12). L'animal se situe sur la cellule 0 et voit les quinze cellules voisines réparties sur trois rangs de vision.

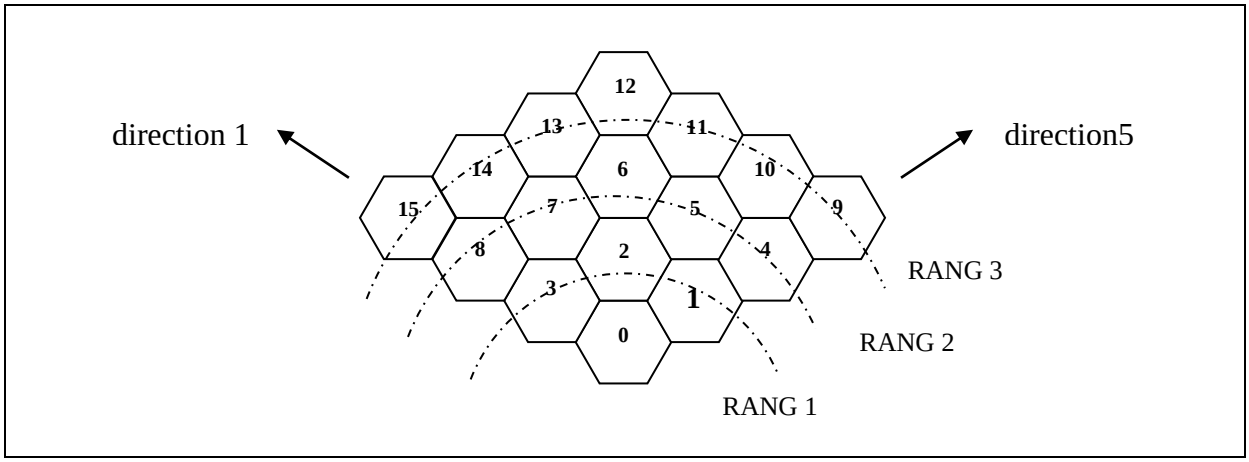


Figure 12 : Représentation des quinze cellules du champ de vision de l'animal

Lorsque l'animal est trop proche d'un autre et qu'il doit s'éloigner, il va choisir une des deux cellules les plus éloignées de lui, c'est-à-dire la cellule 9 ou la cellule 15. Il demande donc à la parcelle les positions de ces deux cellules et choisit celle qui est la plus éloignée de la position P. S'il choisit la cellule 9, il va emprunter la direction 5 et sinon la direction 1.

Par contre, il peut arriver que les cellules 9 et 15 se trouvent en dehors de la parcelle si l'animal est situé sur un des bords et qu'il regarde en direction de l'extérieur de la parcelle. Il faut donc dans ce cas que l'animal change dans un premier temps sa direction pour la direction opposée et choisisse ensuite la cellule 9 ou la cellule 15 pour sa destination. L'algorithme de la fonction d'éloignement est fourni en annexe D.

2.4.3.Le rapprochement

L'animal se rapproche du point P en déterminant la position de toutes les cellules se situant sur la trajectoire de P et dans le cercle dont le centre est le point P et le rayon la distance *d2Troupeau* (en rouge dans la Figure 13). On part du point P et on se rapproche de l'animal en calculant au fur et à mesure la distance entre le point courant et P, on garde la dernière position pour laquelle cette distance est inférieure à *d2Troupeau*. Ce parcours est effectué en faisant varier d'abord les abscisses, puis les ordonnées et on garde la position pour laquelle la distance calculée est la plus proche de *d2Troupeau*. L'algorithme de la fonction de rapprochement est présenté en annexe D.

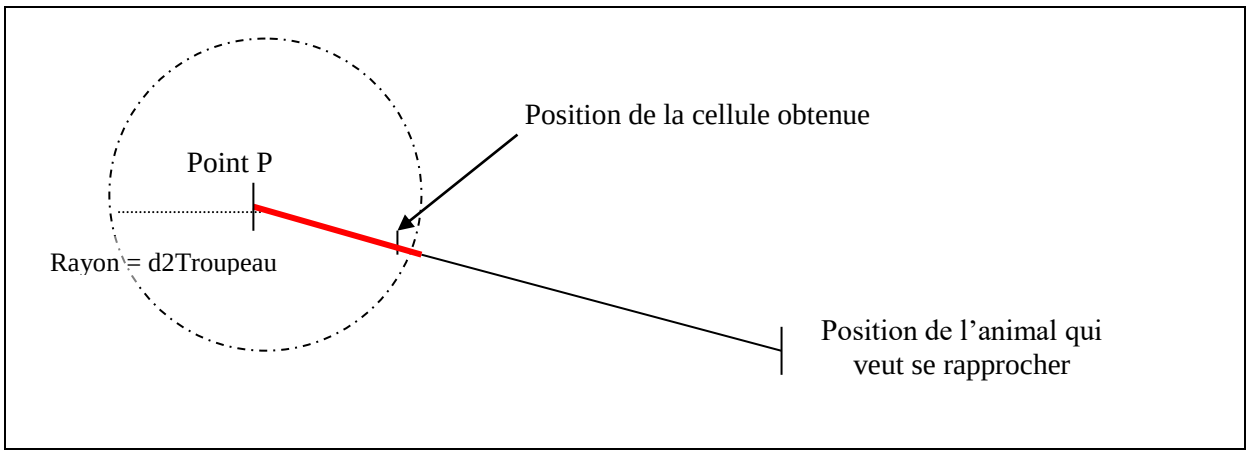


Figure 13 : Schéma explicatif du rapprochement de l'animal par rapport au troupeau

La mise à jour de la direction de l'animal était plus difficile que pour l'éloignement puisqu'il s'agissait de déterminer de quelle direction la droite reliant l'animal au point P était la plus proche. Pour cela, il suffisait d'utiliser la pente de cette droite, que l'on appelle *a*, après avoir déterminé dans quel cadran on allait se situer (cf. Figure 14).

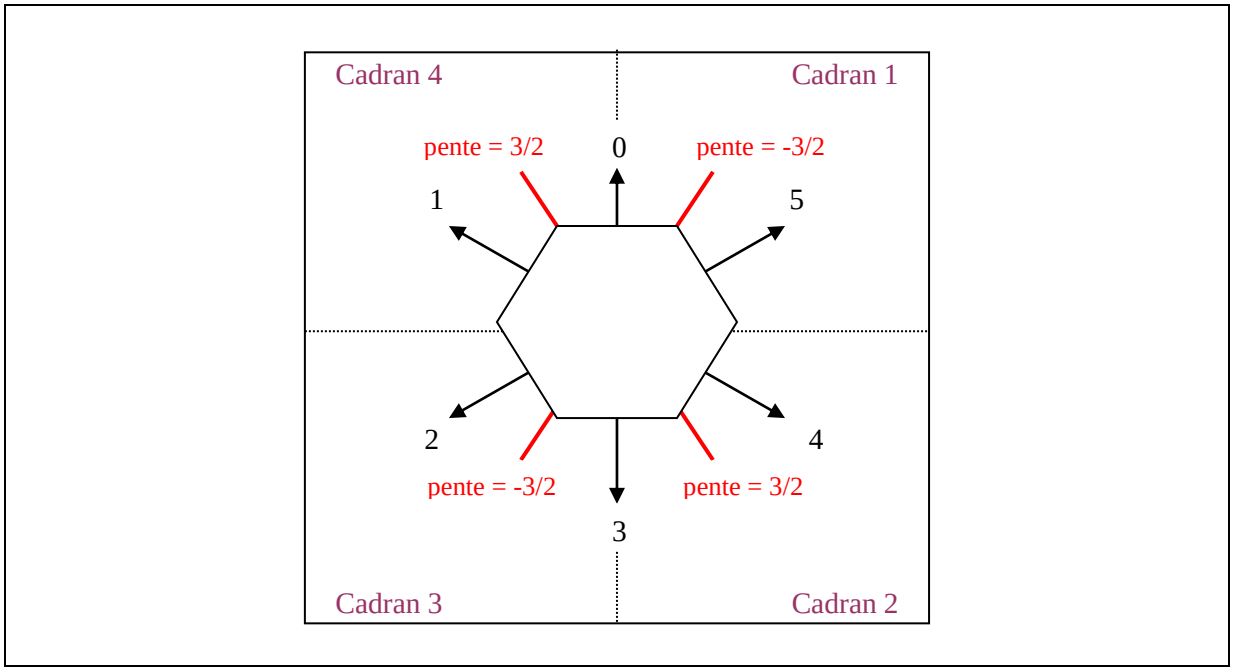


Figure 14 : Fonctionnement du calcul de direction après rapprochement

Soit (x1, y1) les coordonnées de l'animal et (x2, y2) les coordonnées du point P. Sachant que les abscisses augmentent vers la droite et les ordonnées augmentent vers le bas, on obtient la répartition suivante :

- Si x1 < x2 et y1 > y2 ⇒ cadran 1 ;
- Si x1 < x2 et y1 < y2 ⇒ cadran 2 ;
- Si x1 > x2 et y1 < y2 ⇒ cadran 3 ;
- Si x1 > x2 et y1 > y2 ⇒ cadran 4.

Il y a aussi plusieurs cas particuliers à envisager :

- Si x1 = x2 et y1 < y2 ⇒ direction 3 ;
- Si x1 = x2 et y1 >= y2 ⇒ direction 0 ;
- Si y1 = y2 et x1 < x2 ⇒ direction 4 ou 5 (déterminée par tirage aléatoire) ;
- Si y1 = y2 et x1 > x2 ⇒ direction 1 ou 2 (déterminée par tirage aléatoire).

Ensuite, pour chaque cadran, on détermine de quelle direction la droite reliant l'animal au point P est la plus proche en raisonnant par rapport aux pentes des droites en rouge dans la Figure 14.

- Cadran 1 : si a < -3/2 ⇒ direction 0, sinon direction 5 avec tirage aléatoire si a = -3/2 ;
- Cadran 2 : si a > 3/2 ⇒ direction 3, sinon direction 4 avec tirage aléatoire si a = 3/2 ;
- Cadran 3 : si a < -3/2 ⇒ direction 3, sinon direction 2 avec tirage aléatoire si a = -3/2 ;
- Cadran 4 : si a > 3/2 ⇒ direction 0, sinon direction 1 avec tirage aléatoire si a = 3/2.

2.5. Le leadership

2.5.1. Début de leadership

Le leader est un animal susceptible d'effectuer un déplacement long pour changer de site alimentaire, aller boire ou se coucher, en étant suivi par les autres animaux du groupe. Chaque animal a une probabilité d'être leader non nulle, qui lui est affectée en début de simulation et qui reste constante. Il ne peut pas y avoir plusieurs animaux leader en même temps puisque lorsqu'un animal devient leader, les autres le suivent et annulent toutes les actions en cours qui auraient pu les amener à devenir eux-mêmes leader.

Le fonctionnement de l'automate décisionnel ajouté dans le simulateur est décrit dans la Figure 15, il intègre l'action de leadership. Les résultats 1, 2, 3 et 4 correspondent respectivement aux actions manger, boire, repos long et repos court.

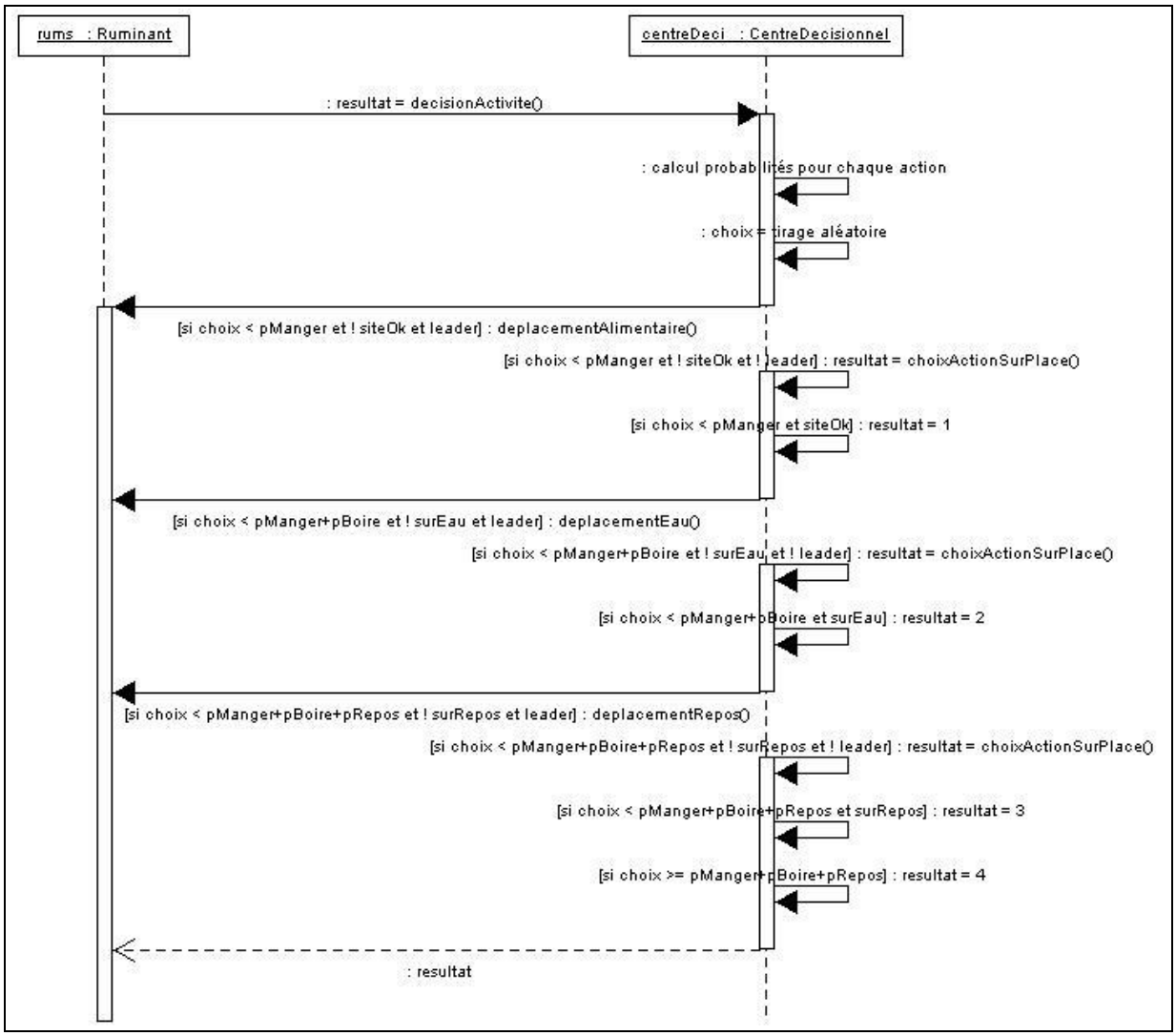


Figure 15 : Diagramme de séquences du fonctionnement de l'automate décisionnel

Dès qu'une action est choisie, on vérifie que la position de l'animal est adaptée à l'activité. Si l'animal veut manger, on regarde s'il est situé sur un site alimentaire qui lui convient. S'il veut boire, on regarde s'il se trouve près d'un point d'eau. Enfin, s'il veut effectuer un repos long, on regarde s'il est proche d'une aire de couchage. Pour le moment, nous n'avons pas défini comment on considérait qu'un site alimentaire convient à un animal. Nous avons donc donné une probabilité de satisfaction égale à 0.8 pour les tests et l'animal effectue un tirage aléatoire à chaque action pour savoir s'il est satisfait de sa position ou pas.

Si sa position n'est pas satisfaisante, il effectue un tirage aléatoire pour déterminer s'il est leader. S'il devient leader, il peut explorer sa mémoire et éventuellement se déplacer vers un nouveau site. S'il n'est pas leader, il effectue un nouveau choix d'action qu'il pourra effectuer sur place (manger sur une des quinze cellules de son champ de vision, boire s'il est sur un site de buvée, se reposer s'il est sur un site de repos ou effectuer un repos court).

2.5.2.Les règles de déplacement

Voyons le cas où l'animal devient leader et où il souhaite manger. Il faut noter que l'animal leader ne consulte pas toujours sa mémoire, il peut aussi avoir envie de partir en exploration. Plus le coefficient 1 / pourcentage de la surface de parcelle connue est petit, plus la probabilité de consulter sa mémoire est importante, puisque l'animal part en exploration en début de simulation, lorsqu'il a encore mémorisé peu de sites de la parcelle. Dans le cas où l'animal consulte sa mémoire, il choisit toujours le meilleur site mémorisé (celui dont le coefficient qualite*surface/distance est le plus élevé) vers lequel il effectuera un déplacement long. S'il part en exploration, il choisit une des quinze cellules de son champ de vision pour un déplacement de proximité. Le déplacement de proximité lui permet alors d'augmenter le nombre de sites connus de sa mémoire qui est mise à jour à chaque défoliation.

La Figure 16 montre l'enchaînement des opérations lorsqu'un animal devient leader et qu'il commence un déplacement long pour se rendre sur un site alimentaire.

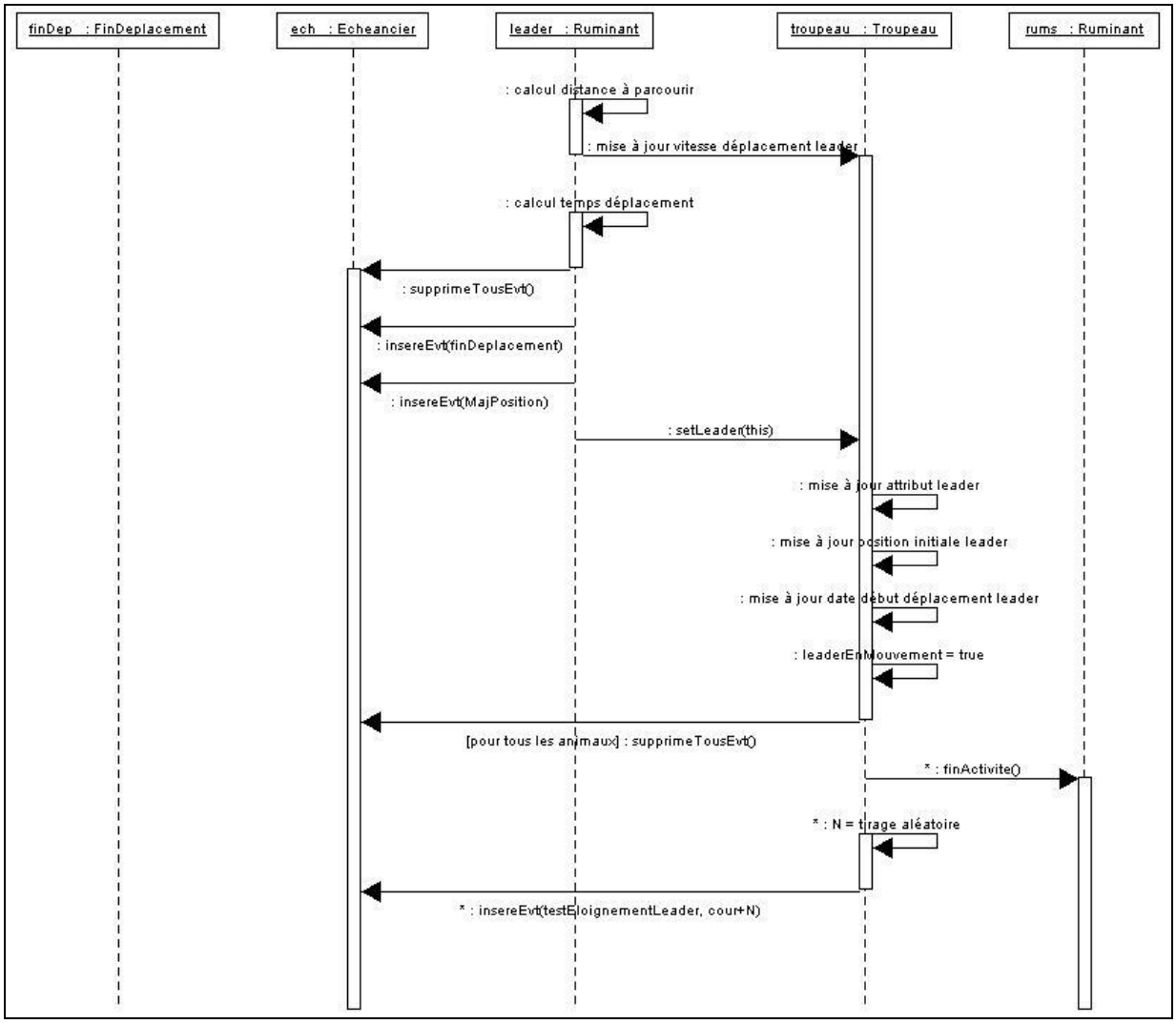


Figure 16 : Diagramme de séquences de début du déplacement long

Lorsque l'animal consulte sa mémoire pour choisir le meilleur site, il détermine aussi le point du site le plus proche de sa position actuelle et c'est donc à cet endroit qu'il va se rendre. Il lui faut tenir compte du temps de déplacement et il annule toute autre activité pour se consacrer à son déplacement, c'est le but de la fonction `deleteTousEvt()` qui supprime tous les évènements insérés dans l'échéancier pour l'animal qui appelle cette fonction. L'événement *CalculPosition*, quant à lui, permet de mettre à jour la position de l'animal leader régulièrement pour que sa trajectoire puisse être observée grâce à la visionneuse et que les autres animaux du troupeau puissent le suivre. La position initiale, le début et la vitesse de déplacement du leader sont sauvegardés pour qu'il puisse calculer à tout moment sa position à partir du temps de déplacement écoulé.

L'animal indique ensuite au troupeau qu'il est devenu leader, pour que le troupeau le communique à tous les animaux. Ceci est fait par insertion pour chaque animal d'un événement *TestEloignementLeader* (cf. Annexe B) qui permettra au troupeau de ne pas trop s'éloigner du leader. Cet événement n'est pas inséré à la même date pour tous les animaux pour éviter que tout le groupe parte en même temps, les dates d'insertion sont uniformément réparties entre deux dates maximale et minimale. Quand cet événement est exécuté, l'animal concerné interrompt l'activité en cours pour se consacrer à suivre le leader.

2.5.3. Test de rapprochement par rapport au leader

Le test d'éloignement par rapport au leader, illustré par la Figure 17, est presque semblable au test de rapprochement par rapport au troupeau. Il est juste effectué plus fréquemment, toutes les minutes. On demande dans un premier temps au leader de calculer sa position actuelle puis on calcule la distance entre l'animal et le leader avant d'effectuer des comparaisons par rapport aux distances de référence *d1Leader*, *d2Leader* et *d3Leader*.

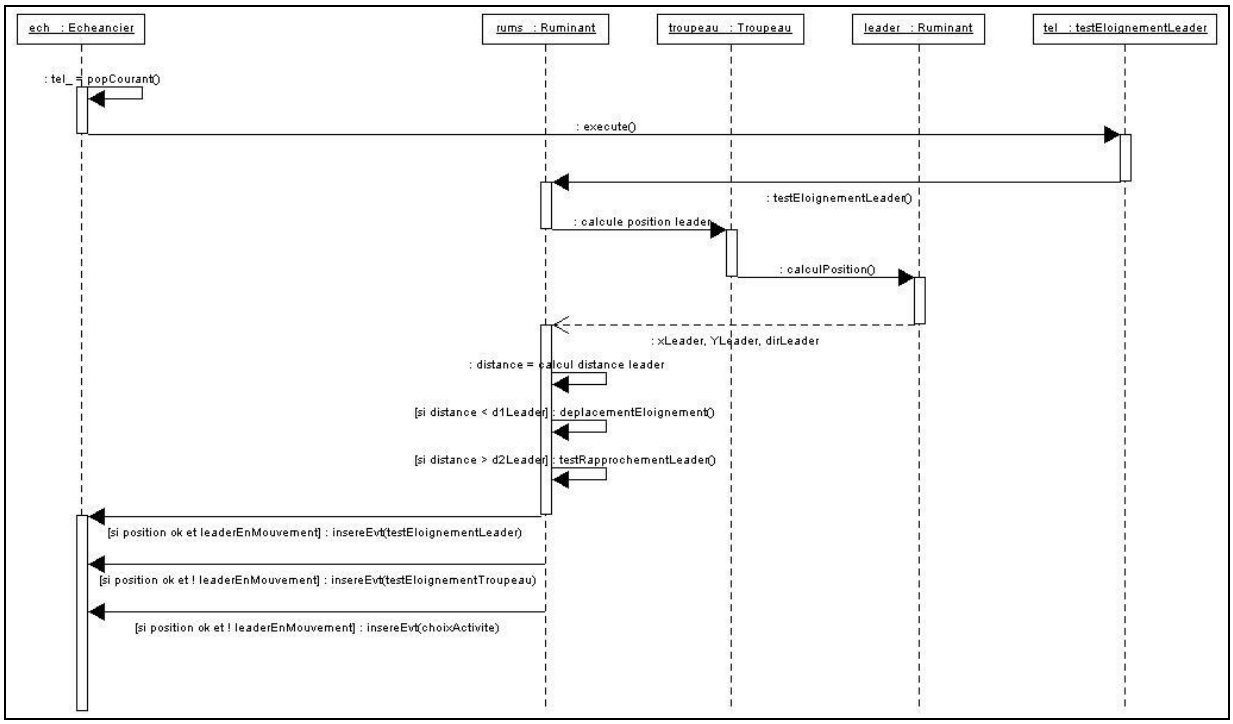


Figure 17 : Diagramme de séquences du test de rapprochement par rapport au leader

Si un animal est trop loin du leader et qu'il doit se rapprocher, il demande la position courante du leader et calcule une position intermédiaire qu'il peut atteindre en *TempsCalculTrajectoire* secondes et s'y rend. Comme le leader peut être aussi en train se déplacer, l'animal ne peut pas se déplacer directement sur la position du leader sinon il risquera d'avoir à se déplacer à nouveau lorsqu'il y sera parce que le leader aura bougé. On décide donc que l'animal se déplace petit à petit, pendant *TempsCalculTrajectoire* secondes puis il redemande la nouvelle position de l'animal et continue à se déplacer. Ce paramètre est donc une période de réajustement de la trajectoire de l'animal sur celle du leader.

Lorsque l'animal s'est déplacé sur cette position intermédiaire, il refait un test d'éloignement et ainsi de suite. La trajectoire du leader sera donc recalculée toutes les *TempsCalculTrajectoire* secondes. L'animal arrête de se rapprocher lorsque la distance est comprise entre *d2Leader* et *d3Leader*. Il insère alors un nouvel événement *TestEloignementLeader* dans l'échéancier une minute plus tard si le leader est en cours de déplacement. Par contre, si le leader a arrêté son déplacement, l'animal effectue un nouveau choix d'activité et recommence les tests d'éloignement par rapport au troupeau, jusqu'à ce qu'il y ait un nouveau leader.

Le rapprochement par rapport au leader ressemble au rapprochement par rapport au troupeau, son algorithme se trouve en annexe D. J'ai donc voulu réutiliser la fonction de rapprochement mais pour cela il m'a fallu l'adapter un peu. Lorsque l'on appelle la méthode, il faut préciser une position de départ et une position d'arrivée et aussi une distance maximale entre la nouvelle position et la destination à atteindre. Dans le cas du rapprochement par rapport au troupeau, la position de départ est celle de l'animal, celle d'arrivée celle du troupeau et la distance limite la distance *d2Troupeau*, comme expliqué dans le paragraphe 2.4.3.

Dans le cas du rapprochement par rapport au leader illustré par la Figure 18, le raisonnement est inversé. On calcule la distance appelée *dist* que peut parcourir l'animal en direction du leader en *TempsCalculTrajectoire* secondes. On part des coordonnées de l'animal et on calcule les coordonnées de toutes les cellules en direction de la position actuelle du leader (en rouge sur la Figure 18). Le calcul s'arrête lorsque la distance entre la cellule courante et l'animal est supérieure à *dist*. L'appel de la fonction de rapprochement se fait donc avec les paramètres inversés, la position de départ est celle de l'animal leader, la position d'arrivée celle de l'animal qui souhaite se rapprocher et la distance limite la distance *dist*.

Par contre, il faut faire attention que l'animal ne dépasse pas le leader s'il a besoin de moins de *TempsCalculTrajectoire* secondes pour rejoindre le leader. Dans ce cas-là, le rapprochement est un rapprochement classique, comme celui qui a été fait pour se rapprocher du troupeau.

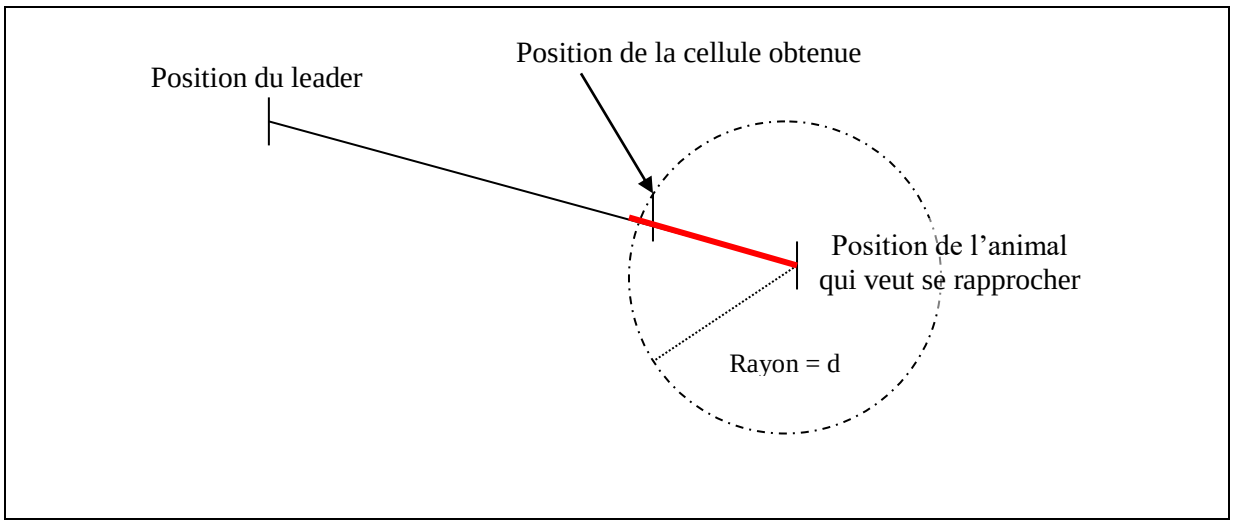


Figure 18 : Schéma explicatif du rapprochement de l'animal par rapport au leader

Partie III : Résultats et discussion

1. Outils utilisés

Lors de mon stage, j'ai travaillé sur un PC basé sur un processeur AMD Athlon, cadencé à 700 Mhz, avec 256 Mo de mémoire vive.

Le simulateur a été entièrement codé en langage C++, en utilisant le compilateur g++ de GNU. Le travail de développement s'est fait à partir du système d'exploitation Linux que j'ai installé sur le PC à mon arrivée à l'INRA, avec la distribution Mandrake 9.1 pour avoir un meilleur compilateur.
Le simulateur ne tourne actuellement que sous Linux mais il devrait pouvoir être porté sans trop de difficultés sous Windows en modifiant le nom les répertoires de fichiers d'entrée/sortie, même si les performances devraient théoriquement être moins bonnes. Par contre, la visionneuse ne fonctionne que sous Linux. Le portage sous Windows n'a donc pas encore été envisagé, il faut attendre d'avoir des résultats de tests du simulateur complet sous Linux.

L'implémentation du module social a nécessité un travail d'analyse important, avec de fréquentes modifications au fur et à mesure des réunions avec les biologistes. Pour la phase d'analyse, le langage de modélisation UML a été retenu parce que c'est la norme dans le domaine objet, il est compris et utilisé par un grand nombre de personnes, notamment les personnes qui travaillent sur le simulateur. Les diagrammes UML présentés dans ce rapport ont tous été réalisé avec le logiciel Poséidon for UML Community Edition 1.51.

2. Déroulement du travail

La première étape de mon travail a consisté à faire une étude de l'existant et aussi un travail de bibliographie afin de mieux connaître les relations sociales au sein d'un troupeau et leur influence sur le comportement des ruminants. L'étude de l'existant a aussi été complétée par la correction des erreurs découvertes lors du travail de vérification puis de validation et qui gênaient pour le bon fonctionnement du prototype réalisé par Guillaume Martin.

Le développement du module social et spatial a ensuite pu commencer. Pour permettre de tester efficacement toutes les parties du module S, chaque partie a été d'abord analysée puis développée et testée de manière indépendante, sur le principe des prototypes précédemment adopté.

Le premier prototype consistait d'abord à ajouter les programmes précédemment codés, l'échéancier et l'automate décisionnel, puis à différencier les animaux du troupeau, en paramétrant les entrées et classant les sorties par animal. Le deuxième ajoutait la gestion des distances entre chaque animal et le reste du troupeau, pour obtenir une cohésion entre les animaux et aussi la gestion des collisions sur les cellules. La gestion des relations de grégarisme imposait d'effectuer de nombreux tests pour voir comment réagissaient les animaux en cas de grande proximité ou en au contraire en cas d'éloignement important. Il fallait donc observer le fonctionnement d'un groupe d'animaux à l'intérieur d'une séquence de pâturage, plusieurs animaux pâturant de manière simultanée, et vérifier à l'aide des fichiers de trace de la trajectoire que les animaux évitent les collisions et ne pâturent pas simultanément la même cellule.

Le troisième prototype consistait à rajouter la présence d'un animal leader dans le troupeau et effectuer des tests d'éloignement par rapport au leader, pour voir si tous les animaux le suivent en empruntant une trajectoire de déplacement correcte. Le dernier prototype, qui n'est pas implémenté, consiste enfin à ajouter la mémoire spatiale pour permettre les déplacements longs en testant les deux méthodes, méthode des contours et carte pixélisée, et éventuellement choisir la meilleure des deux.

3. Résultats, fonctionnement du programme

Actuellement, le programme est opérationnel et la simulation se déroule sans interruption. Le prototype intègre l'échéancier, l'automate décisionnel mais la mémoire spatiale n'a pas encore été ajoutée au simulateur, elle devrait l'être avant la fin de mon stage.

La simulation peut donc être lancée avec plusieurs animaux et les relations sociales de grégarisme et de leadership sont gérées. Les animaux réagissent à l'éloignement ou à la trop grande proximité en se rapprochant ou s'éloignant. Par contre, comme la mémoire ne fait pas encore partie du simulateur, l'animal leader commence un déplacement long toujours vers la même cellule de la parcelle, il s'agit d'une cellule que j'ai fixée au hasard. Les tests se sont avérés concluants mais un travail de validation permettrait de déterminer de manière plus précise si les valeurs sont cohérentes.

Les fichiers en entrée sont pratiquement les mêmes que pour l'ancien prototype, seuls les fichiers de configuration du troupeau et de chaque ruminant, *troupeau.cfg* et *ruminant.cfg* ont été modifiés de façon notable. Les fichiers en sortie n'ont pas évolué, j'ai seulement rajouté un nouveau fichier nommé *eloignement.txt* qui précise les résultats des test d'éloignement, que ce soit par rapport au troupeau ou par rapport à l'animal leader. Ce fichier donne également les cordonnées de chaque animal au moment des vérifications de distance ainsi que la position et la destination de l'animal en cas de déplacement.

D'après les tests que j'ai pu effectuer sur un troupeau de quatre animaux pour une durée de dix jours, les tests d'éloignement fonctionnent correctement, même s'il est difficile de tester tous les cas particuliers (animal se trouvant sur les bords…). J'ai observé le comportement des animaux en cas de grande proximité et aussi en cas d'éloignement important. Pour cela, il suffisait de modifier les coordonnées de départ de chaque animal dans le fichier de configuration *ruminant.cfg*.

J'ai également effectué des tests de performances du simulateur avec un troupeau de quinze animaux sur quinze jours de simulation. Tous les animaux du troupeau ont les mêmes caractéristiques, un poids de 80 kg et un besoin énergétique quotidien de 150. Ce sont donc des animaux qui mangent beaucoup. Les animaux ont été placés plutôt en bord de parcelle, de façon très rapprochée pour observer les comportements du simulateur en cas de fréquentes collisions de pâturage.

3.1. Tests avec une parcelle de 1 million de cellules

Nous avons d'abord voulu tester le programme avec une grande parcelle de 1 million de cellules. Cette simulation a pris beaucoup de temps (environ 45 minutes) et la mémoire occupée est très importante (environ 550 Mo). C'est pourquoi nous avons essayé de supprimer les sorties fichiers qui prenaient le plus de place, c'est-à-dire les fichiers de végétation créés en fin de journée et récapitulant la mise à jour de la parcelle pour tenir compte de la pousse de l'herbe et de la sénescence.

Sans ces sorties végétales, on peut observer que la taille disque occupée est beaucoup plus faible (cf. Figure 19) et la simulation dure moins longtemps (cf. Figure 20). Cela est dû au fait que l'écriture dans les fichiers pour la mise à jour de la parcelle est très coûteuse. Par contre, la mémoire occupée lors de l'exécution de l'application est sensiblement la même (cf. Figure 21).

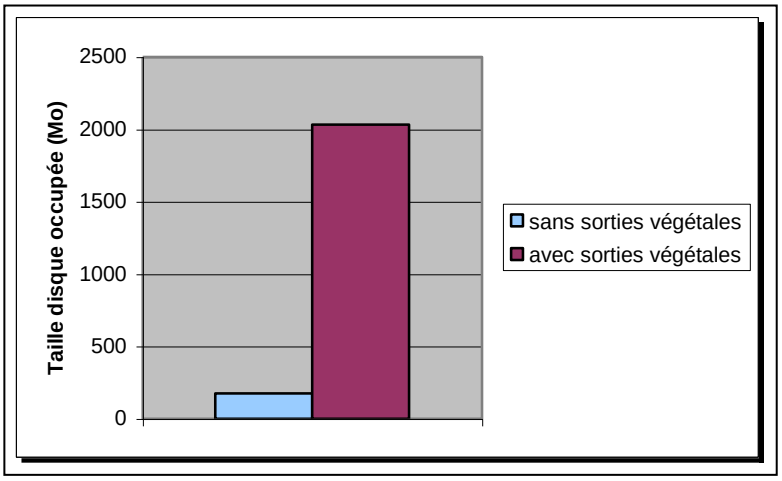


Figure 19 : Diagramme montrant la taille disque occupée avec et sans les sorties végétales

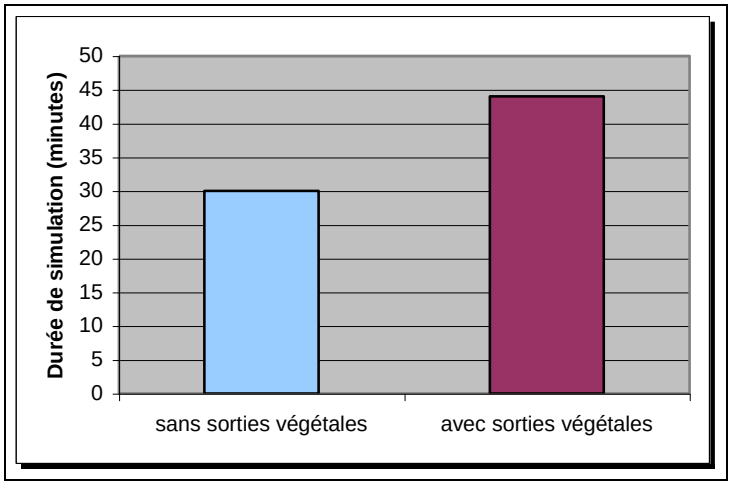


Figure 20 : Diagramme montrant la durée de simulation avec et sans les sorties végétales

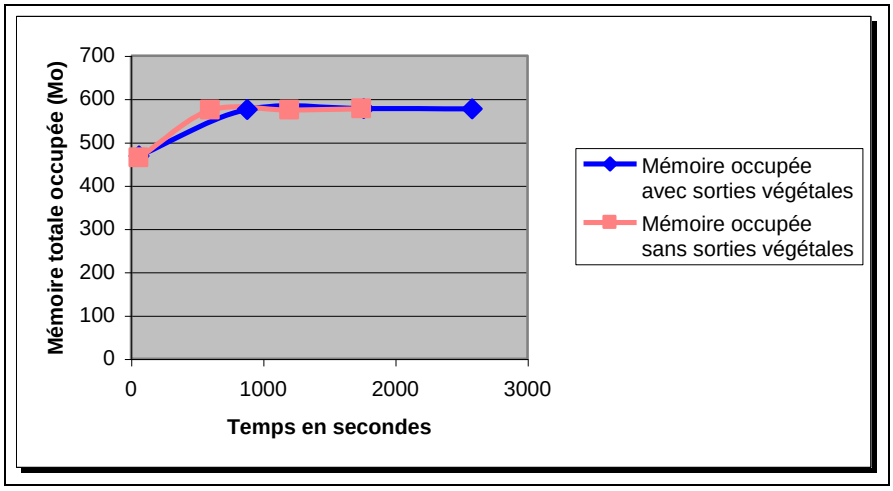


Figure 21 : Diagramme montrant la mémoire occupée avec et sans les sorties végétales

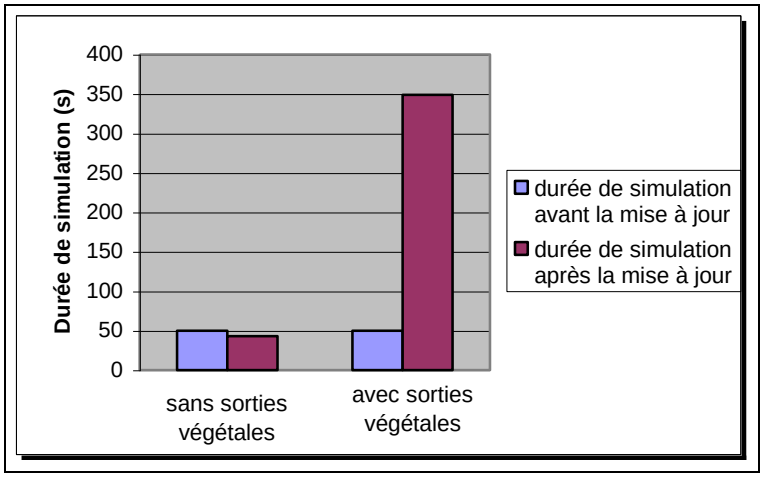


Figure 22 : Diagramme montrant les durées de simulation avec et sans sorties végétales

Pour montrer l'importance de l'écriture des fichiers de sortie lors de la mise à jour de la parcelle sur la durée de simulation, j'ai réalisé une simulation sur un jour en mesurant le temps écoulé pour effectuer la simulation avant la mise à jour de la parcelle. Puis j'ai mesuré le temps nécessaire pour effectuer la mise à jour de la parcelle. La simulation a été réalisée avec les sorties végétales de mise à jour et sans ces mêmes sorties végétales. La Figure 22 illustre les résultats obtenus. On peut observer que la mise à jour de la parcelle prend beaucoup plus de temps avec les sorties végétales, ce qui explique la différence de durée obtenue sur une simulation de quinze jours.

On a également vu que la mémoire occupée par le simulateur est très importante, une partie conséquente de la zone de swap est utilisée. C'est pourquoi nous avons voulu réduire la mémoire utilisée en diminuant le nombre de cellules de la parcelle. Nous avons effectué des tests avec 400 000 cellules puis avec 300 000 sans les sorties végétales puis nous avons comparé avec les données obtenues avec 1 million de cellules.

3.2. Comparaison des performances selon le nombre de cellules de la parcelle

Plus la simulation est effectuée sur une grande parcelle, plus la durée de simulation (cf. Figure 23) et la mémoire occupée (cf. Figure 24) sont importantes. Cela est dû au fait que plus la parcelle est grande, plus il faut de temps pour lire toutes les valeurs de la carte et aussi pour mettre à jour toutes les cellules en fin de journée.

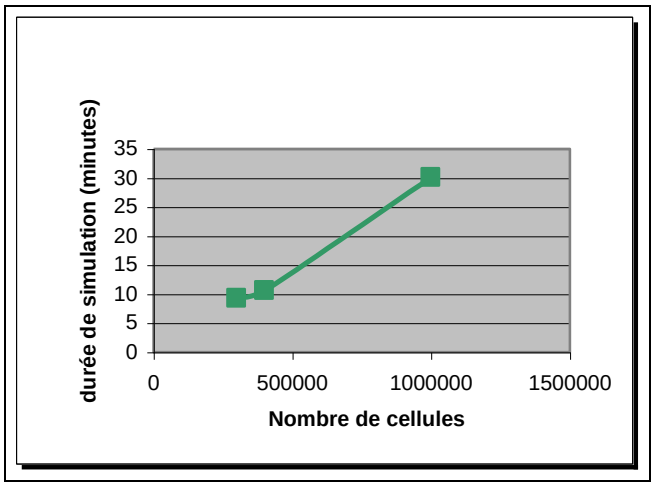


Figure 23 : Diagramme montrant la durée de simulation en fonction de la taille de la parcelle

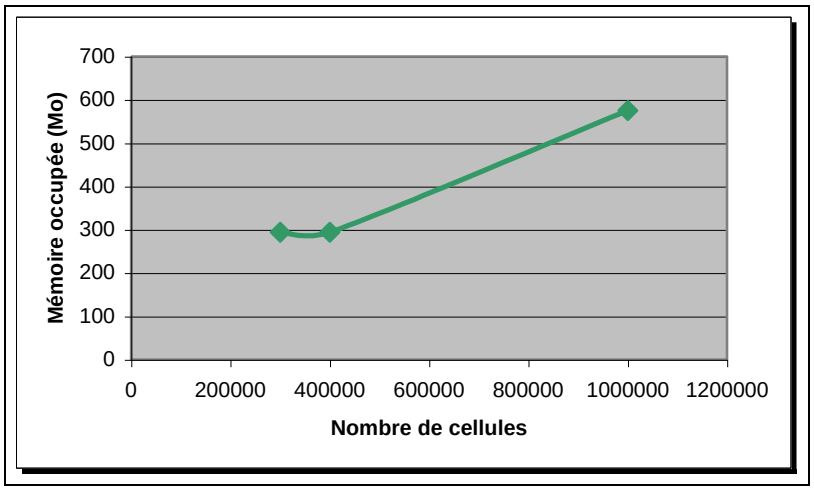


Figure 24 : Diagramme montrant la mémoire totale occupée en fonction de la taille de la parcelle

Par contre, le pourcentage de cpu utilisée (cf. Figure 25) a tendance à diminuer lorsque le nombre de cellules augmente. On peut supposer que cela s'explique par une utilisation de la zone de swap plus importante.

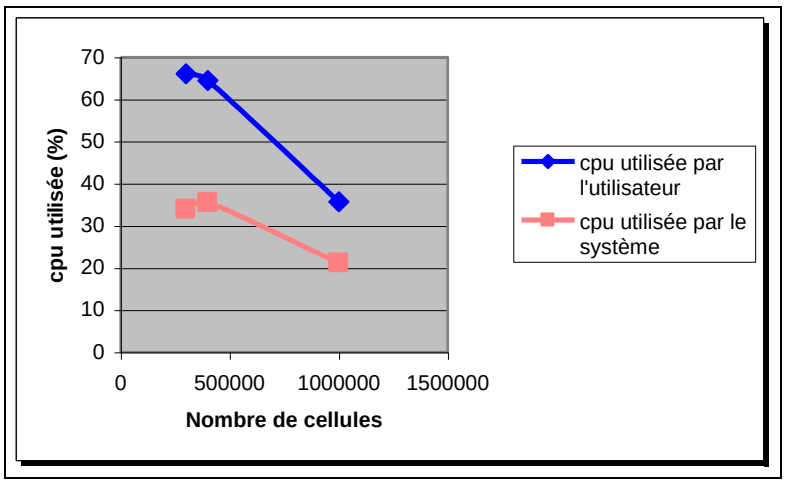


Figure 25 : Diagramme montrant le pourcentage de cpu utilisée en fonction du nombre de cellules

4. Difficultés rencontrées

Les difficultés que j'ai pu rencontrer tout au long du stage ont plutôt été de l'ordre de l'implémentation. L'analyse n'a pas posé vraiment de problèmes, même s'il a fallu modifier les diagrammes UML assez fréquemment, les discussions avec les biologistes nous permettant de formuler nos idées réciproques et de corriger au fur et à mesure notre vision de la solution.

Une des difficultés majeures a concerné l'ajout de l'échéancier dans le simulateur. A chaque fois qu'un événement est inséré dans l'échéancier, il faut préciser sa date d'activation. Au début, il m'était difficile de voir dans quel ordre les évènements devaient être insérés, peut-être par manque de recul par rapport à l'enchaînement des étapes. Aussi, le temps était considéré en secondes alors que je manipulais des minutes, une conversion des durées s'avérait donc nécessaire. Par conséquent, j'insérais les évènements avec une date d'activation incorrecte, ce qui occasionnait un mauvais fonctionnement de la simulation. La correction de ce genre de problèmes m'a pris un certain temps.

Une autre difficulté concernait l'éloignement. Au départ, je n'avais pas pensé que la cellule 9 ou la cellule 15 pouvaient faire sortir l'animal de la parcelle. Aussi, j'obtenais souvent un message d'erreur me signalant que le module végétal avait reçu une coordonnée négative. En consultant le fichier de sortie des tests d'éloignement, je me suis aperçue qu'une des deux cellules 9 et 15 avait des coordonnées négatives donc lorsque l'animal s'y déplaçait, ses coordonnées devenaient négatives. C'est pourquoi j'ai dû rajouter des tests et un changement de direction de l'animal s'il était tourné vers l'extérieur de la parcelle.

5. Futur du simulateur

Le simulateur est un projet qui dure déjà depuis plusieurs années mais qui n'est pas encore terminé. Il fera encore l'objet de stages et projets.

D’abord prévu pour des moutons, le simulateur devrait pouvoir être utilisé pour n’importe quel ruminant, des vaches par exemple, sous réserve de modifier les équations du modèle. Une version future devra donc passer les constantes des équations comme les équations du rumen en variables stockées dans un fichier.

La simplification des sorties est indispensable. Il y a actuellement beaucoup de sorties qui sont très lourdes à gérer. Le choix des paramètres en sortie devra être rajouté côté animal dans l’interface, comme ce qui a été fait pour le végétal et les sorties végétales pourront être agrégées par sites, par faciès…
Aussi, les paramètres redondants dans les entrées dus au développement de chaque module séparément devront être supprimés, tels que la taille de la parcelle.

Certains outils comme la visionneuse ou l’interface sont désormais inutilisables avec plusieurs animaux. Il faudrait donc modifier ces programmes pour qu’ils puissent gérer un troupeau entier. L’adaptation de l’interface ne devrait pas demander beaucoup de modifications, il suffira de modifier les répertoires des fichiers de sortie pour avoir un répertoire par animal et aussi de faire saisir dans une des étapes de l’interface le nombre d’animaux du troupeau.

Il faudrait aussi intégrer ces deux programmes au simulateur pour que l’utilisateur n’ait pas à lancer successivement chaque partie de programme en ligne de commande mais que le simulateur les appelle directement.

La validation a été faite au fur et à mesure pour chaque module, une validation du module social devra donc être effectuée avant l’ajout de nouvelles fonctionnalités. Ce travail de validation permettrait ainsi de corriger les erreurs qui persistent et qui sont dues à l’augmentation de la taille du code du simulateur.

Conclusion

Trois prototypes du simulateur sont actuellement opérationnels, même si le dernier est plus fiable grâce à une meilleure gestion des cas particuliers. Le premier permet de lancer une simulation avec plusieurs animaux et intègre l'automate décisionnel et l'échéancier. Le deuxième gère les collisions de pâturage sur les cellules ainsi que les relations de grégarisme qui amènent les ruminants à rester groupés. Enfin, le troisième prototype intègre les relations de leadership qui apportent une nette tendance à suivre l'animal leader.

Le quatrième et dernier prototype n'est pas encore achevé, il consiste à ajouter la mémoire spatiale. L'analyse et l'implémentation de cette application sont déjà effectuées, seules manquent les étapes d'ajout au simulateur et de tests. Cette partie devrait tout de même être achevée à la fin du stage.

Ce stage m'a beaucoup apporté, autant sur le plan technique que relationnel. En ce qui concerne la programmation, je pense pouvoir corriger plus rapidement les erreurs d'exécution et j'ai surtout appris à développer de manière incrémentale, en réalisant plusieurs prototypes et les testant au fur et à mesure. J'ai aussi eu l'occasion de rencontrer à plusieurs reprises les biologistes qui m'ont expliqué leurs attentes pour la réalisation du module spatial et social du simulateur. Je me suis alors rapidement rendue compte qu'il n'est pas toujours facile de comprendre et de se faire comprendre lorsque l'on n'a pas le même domaine d'activité que son interlocuteur. J'ai donc appris à m'adapter à la terminologie biologique et aussi essayé d'intégrer le point de vue des utilisateurs, en oubliant un peu la vision informatique. Enfin, j'ai appris qu'il était très important de rédiger les comptes rendus de réunion au fur et à mesure, pas seulement pour la rédaction du rapport mais aussi pour revenir plus facilement sur un aspect non compris.

Même si le quatrième prototype constitue une première version complète du simulateur, il faudra encore le modifier pour qu'il puisse être utilisable facilement et qu'il fournisse des résultats fiables. Une première étape sera sûrement la validation du prototype intégrant le module social et spatial. Une simplification des entrées et des sorties devra être également envisagée puisque certaines sont redondantes et ne présentent donc pas un réel intérêt. Enfin, la visionneuse, l'interface utilisateur ou le générateur de statistiques devront être adaptés à cette nouvelle version. Et puis, surtout, le but final du simulateur est d'être utilisable aussi bien avec des bovins que des ovins. Pour le moment, le développement est fait pour des ovins, ce qui va nécessiter des modifications importantes pour rendre le simulateur générique, notamment en ce qui concerne les équations du rumen.

Références bibliographiques

[BEECHAM et FARNSWORTH 1998], BEECHAM J. A., FARNSWORTH K. D., *Animal foraging from an individual perspective : an object orientated model*, Ecological Modeling, n° 113, pages 141-156, 1998

[DUMONT et col. 2001] DUMONT B., HILL D. R. C., BAUMONT R., CARRERE P., MASSON L., MAZEL C., PEROCHON L., Modélisation des processus spatiaux d'utilisation de couverts herbacés hétérogènes par les herbivores: Représentation de leur mémoire spatiale, Actes du séminaire INRA-CIRAD Modélisation du fonctionnement des troupeaux, Millau, pages 25-32, 2001

[DUMONT et BOISSY 1999] DUMONT B., BOISSY A., *Relations sociales et comportement alimentaire au pâturage*, INRA Prod. Anim., n°12, pages 3-10, 1999

[DUMONT et HILL 2003] DUMONT B., HILL D. R. C., *Spatialized models of group foraging by herbivores : What can Multi-Agent Systems offer ?*, Accepté pour publication dans Animal Research, 2003

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

[FERRAND 1998] FERRAND N., *Modélisation et systèmes multi-agents*, http://wwwlisc.clermont.cemagref.fr/Labo/activite_recherche/projets/Projets_en_cours/MultiAgents/pressma.htm, Présentation Cemagref Clermont-Ferrand 1998

[LEMLOUMA et BOUDINA 2001], LEMLOUMA T., BOUDINA A., *L'intelligence artificielle distribuée et les systèmes multi-agents*, http://opera.inrialpes.fr/people/Tayeb.Lemlouma/Papers/IAD_Presentation.pdf, publication INRIA Rhône-Alpes

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

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

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

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