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 II - ANNEXES

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 II - ANNEXES

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

Table des figures et illustrations

Table des matières

Tome I

Tome II

Annexe A : Modifications apportées aux fichiers en entrée

1. Module végétal

1.1. Fichier environnement.csv

Ce fichier utilisé par le module végétal sert à renseigner les températures de tous les jours de l'année ainsi que la valeur du paramètre PARi (rayonnement photosynthétique actif incident). Auparavant, si l'on voulait commencer la simulation en cours d'année, il fallait supprimer dans le fichier les valeurs de températures du début de l'année pour ne garder que celles qui correspondaient aux jours de la simulation. De plus il y avait trop de températures lues, plus que le nombre de jours de simulation. Cela imposait donc de toujours sauvegarder une version complète du fichier. Une modification a été apportée pour que seule la lecture des températures souhaitées soit effectuée et que les autres valeurs du fichier soient ignorées.

1.2. Fichier param.csv

Ce fichier de configuration est utilisé par le module végétal pour renseigner les caractéristiques générales de la simulation végétale. Les paramètres de ce fichier après les modifications que j'ai apportées au prototype de Guillaume Martin sont les suivants :

- Nombre de colonnes : largeur de la parcelle ;
- Nombre de facies : nombre de faciès présents dans la parcelle ;
- Nombre de lignes : hauteur de la parcelle ;
- Nombre de sites : nombre de sites présents dans la parcelle ;
- Rang de vision : rang de vision de l'animal ;
- Somme des temperatures au premier jour (facultatif) : cumul des températures au premier jour de la simulation végétale.

Le paramètre *Somme des températures au premier jour* est devenu facultatif. Si l'utilisateur le renseigne, il sera pris en compte et sinon il sera automatiquement calculé grâce à un cumul des températures présentes dans le fichier *environnement.csv* du premier jour de l'année au premier jour de la simulation végétale.

La durée et le jour du début de la simulation ont été supprimés puisqu'ils sont renseignés à l'aide de l'interface en début de simulation. L'écran de saisie est représenté par la Figure 1, il sert aussi à saisir le jour de début et la durée de la simulation animale si le nombre d'animaux saisi est positif. Une vérification est aussi faite dans l'interface pour que la simulation animale ne puisse pas se dérouler seule, sans la simulation végétale qui tourne en parallèle. Il faut donc que les valeurs saisies par l'utilisateur vérifient les deux conditions suivantes :

- debutVegetal < debutAnimal ;
- debutAnimal+dureeAnimal < debutVegetal+dureeVegetal.

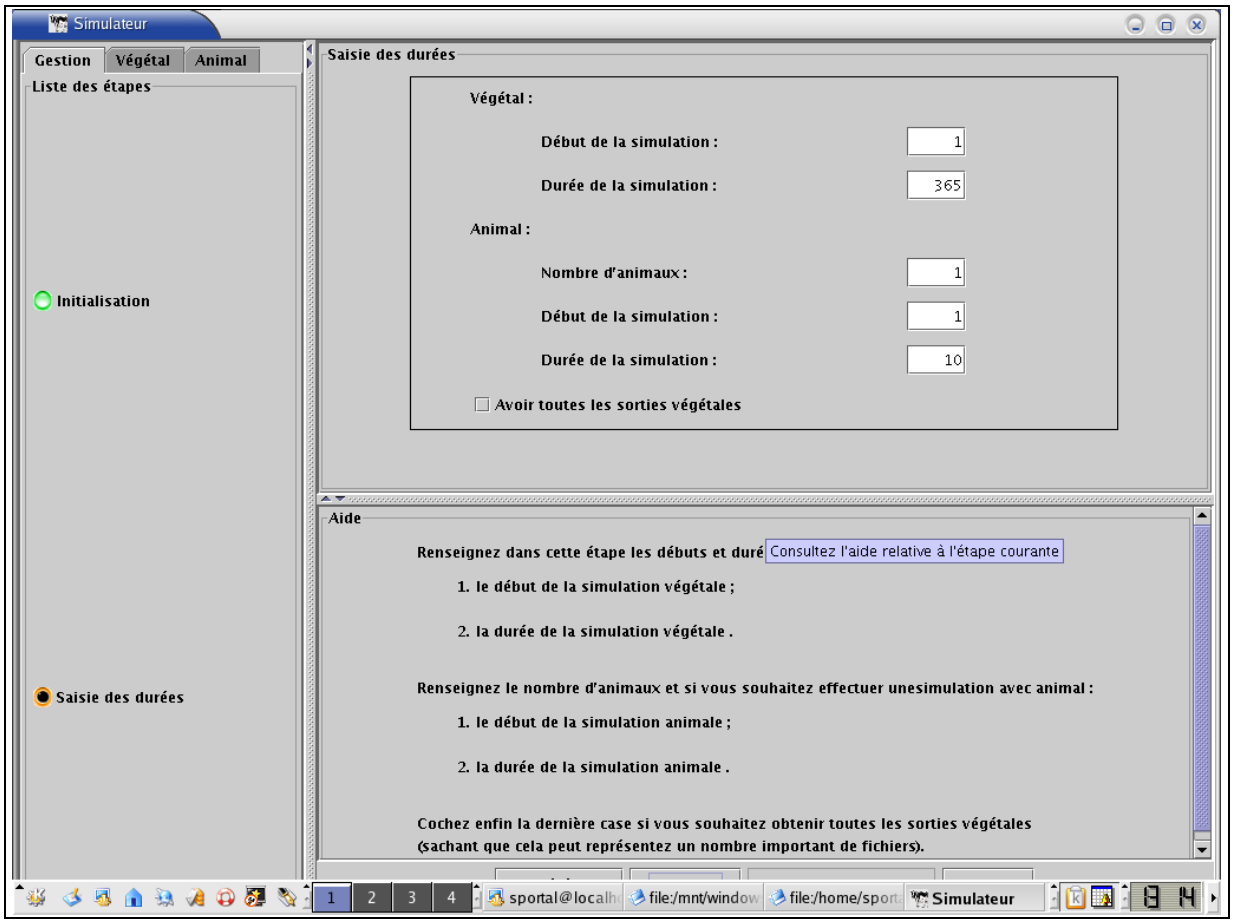


Figure 1 : Ecran de l'interface utilisateur permettant la saisie des débuts et durées de simulation

2. Module animal

2.1. Fichier ruminant.cfg

Ce fichier a été entièrement modifié au cours de mon stage puisqu'il intègre désormais toutes les caractéristiques des animaux du troupeau. Les paramètres de chaque animal contenus dans ce fichier sont les suivants :

- A, B : coefficients de probabilité ;
- APERCEPTION, BPERCEPTION : coefficients pour la fonction de perception ;
- EBC : bilan énergétique cumulé de la veille ;
- FCTDISTANCE ;
- NBPROCHES ;
- PARTICULIER ;
- LW : poids vif de l'animal ;
- MER : besoin énergétique quotidien ;
- SELECTIONS, PONDERATION : coefficients de sélectivité ;
- TEMPSBOIREMOY : durée d'une séquence de buvée ;
- TEMPSMANGERMOY : durée d'une séquence de défoliation ;
- TEMPSREPOSCOURTMOY : durée d'une séquence de repos court ;
- TEMPSREPOSLONGMOY : durée d'une séquence de repos long ;

- TEMPSRUMINATIONMOY : durée d'une séquence de rumination ;
- X_DEPART, YDEPART :coordonnées de la cellule de départ de l'animal.

Le paramètre *FCTDISTANCE* indique quelle fonction de distance sera considérée pour calculer la distance entre l'animal et le troupeau. Il y a trois fonctions de calcul de distance possibles :

- min ;

⇒ La distance est calculée par rapport à l'animal le plus proche.

- barycentre ;

⇒ La distance est calculée par rapport au barycentre des *NBPROCHES* animaux les plus proches (la distance calculée est alors égale à la moyenne des NBPROCHES distances).

- particulier.

⇒ La distance est calculée par rapport à un animal particulier du troupeau, celui dont le numéro est *PARTICULIER* (ce numéro est compris entre 1 et le nombre d'animaux du troupeau).

Certains paramètres sont utilisés dans le générateur de trajectoires, les autres sont des paramètres physiologiques de l'animal. Le fichier *genTraject.cfg* qui était utile pour le générateur de trajectoires a donc été supprimé.

2.2. Fichier troupeau.cfg

Ce fichier regroupe les valeurs des paramètres communs à tous les animaux du troupeau.

Paramètres utilisés par le générateur de trajectoires :

- DIR_DEPART : direction de départ des animaux ;
- DIRMILIEU, DIRDROITE, DIRGAUCHE : probabilités de direction ;
- RG1, RG2, RG3 : probabilités de distance.

Paramètres servant à calculer la vitesse de déplacement d'un animal :

- DISTMIN, DISTMAX ;
- TEMPSMIN, TEMPSMAX.

Les deux premiers paramètres *DISTMIN* et *DISTMAX* concernent le déplacement tête haute, un tirage aléatoire entre ces deux valeurs donne la distance parcourue par l'animal tête haute en une seconde, c'est-à-dire la vitesse de l'animal lors d'un déplacement tête haute.

Les deux paramètres suivants *TEMPSMIN* et *TEMPSMAX* concernent le déplacement tête basse, un tirage aléatoire entre ces deux valeurs donne le temps nécessaire pour que l'animal parcoure un mètre tête basse.

Paramètres servant à tester l'éloignement d'un animal par rapport au reste du troupeau :

- D1TROUPEAU, D2TROUPEAU, D3TROUPEAU ;
- TEMPSELOIGNEMENTTROUPEAU.

Les trois distances *D1TROUPEAU*, *D2TROUPEAU* et *D3TROUPEAU* servent de référence pour savoir si l'animal est trop proche d'un autre animal, trop éloigné de ses congénères ou bien placé.

Le paramètre *TEMPSELOIGNEMENTTROUPEAU* représente la période des tests d'éloignement de chaque animal par rapport au troupeau.

Paramètres servant à tester l'éloignement d'un animal par rapport au leader :

- D1LEADER, D2LEADER, D3LEADER ;
- TEMPSCALCULTRAJECTOIRE ;
- TEMPSELOIGNEMENTLEADER ;
- TEMPSMAJPOSITION.

Les trois distances *D1LEADER, D2LEADER* et *D3LEADER* servent de référence pour savoir si l'animal est trop proche de l'animal leader, trop éloigné ou bien placé.

Le paramètre *TEMPSCALCULTRAJECTOIRE* intervient lorsqu'un animal est trop loin du leader et qu'il se rapproche. 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 risquerait d'avoir à se déplacer à nouveau lorsqu'il y serait 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.

Le paramètre *TEMPSELOIGNEMENTLEADER* représente la période des tests d'éloignement de chaque animal par rapport au leader.

Comme le leader peut être amené à se déplacer sur une longue distance, il faut que sa position sur la parcelle soit mise à jour régulièrement pour que la trace de sa trajectoire soit correcte et observable par la visionneuse. L'animal recalcule donc sa nouvelle position toutes les *TEMPSMAJPOSITION* secondes.

Paramètres utilisés par la classe *CentreDecisionnel* :

- DN_i : fonction d'effet jour/nuit ;
- FACTEURREPOSCOURT : facteur utilisé dans l'automate décisionnel pour donner une probabilité non nulle à l'activité repos court.

Annexe B : Modifications apportées aux fichiers en sortie

Fichiers de trajectoire des animaux

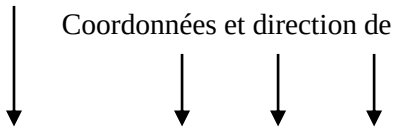
Dans le prototype animal-végétation, seul le déplacement tête basse était implémenté donc un animal ne pouvait se déplacer sur la parcelle que pour pâturer. Avec l'ajout de la gestion des relations sociales, le déplacement tête haute a été implémenté pour permettre aux animaux d'effectuer un déplacement long, de rejoindre le groupe ou encore de suivre le leader. Il fallait donc différencier ces différents types de déplacement dans les fichiers de trajectoire des animaux.

Dans le cas du pâturage, la colonne *choix* du fichier de trajectoire servait à indiquer le numéro de la cellule choisie parmi les quinze cellules du champ de vision. Comme pour les autres types de déplacement la cellule choisie peut être très éloignée de l'animal, on ne peut pas donner le numéro de cette cellule. La colonne *choix* contient alors un entier relatif indiquant la raison du déplacement de l'animal :

- -1 si l'animal s'est déplacé pour s'éloigner du leader ou pour le suivre;
- -2 si l'animal s'est déplacé pour s'éloigner du troupeau ou pour le rejoindre ;
- -3 si l'animal a effectué un déplacement long.

La colonne leader doit également être modifiée pour le cas du déplacement long, sa valeur est 1 dans ce cas et 0 dans tous les autres types de déplacement. Enfin, ce fichier montre aussi les collisions de pâturage grâce à la colonne *collision*. La valeur de cette colonne est égale à 1 en cas de collision et à 0 sinon. L'exemple suivant est un exemple de trajectoire suivie par l'animal.

Date de simulation en secondes



Date	abs	ord	dir	choix	raison	leader	collision
955949 42	48	4	2	1		0	1

collision de pâturage => nouveau choix de cellule effectué

955953 43	48	4	2	1	0	0	

****

995768 14	5	1	-1	1	0	0	

déplacement pour suivre le leader

1000938	33	31	0	-2	1	0	0

déplacement pour suivre le troupeau

1005358	10	40	0	-3	1	1	0

déplacement long (animal leader)

Annexe C : Description des évènements

1. L'exécution des évènements

Dans l'approche par évènements discrets, un échéancier se charge du stockage et de l'exécution des évènements. L'échéancier consulte le premier évènement et l'exécute, l'entité associée est alors activée, elle va déclencher une action dépendant du type d'événement. L'entité (un ruminant dans la Figure 2) pourra ensuite insérer un nouvel évènement dans l'échéancier pour continuer à participer à la simulation.

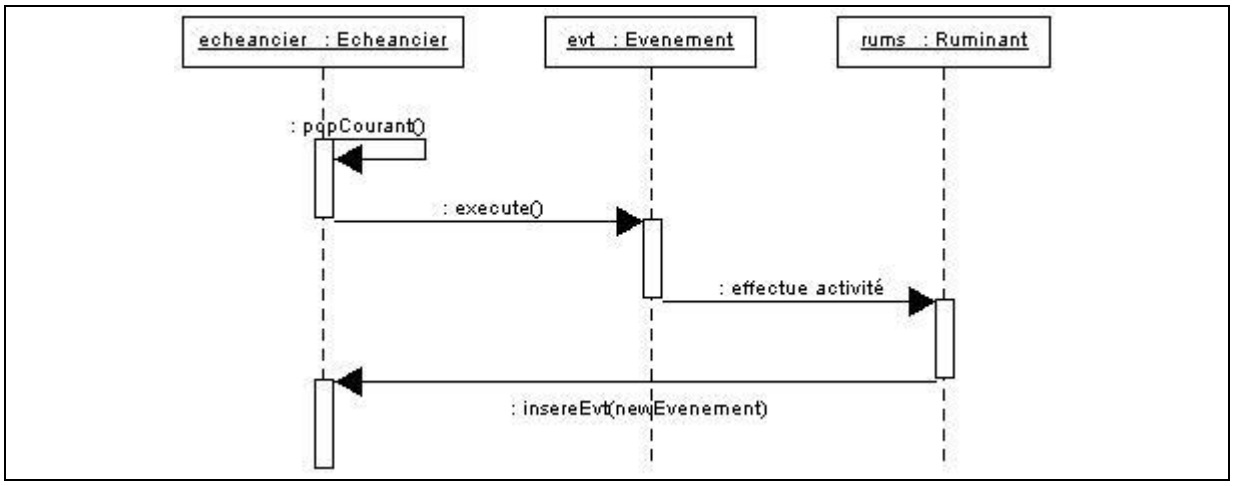


Figure 2 : Diagramme de séquences d'exécution d'un événement

2. Les types d'évènements

Tous les évènements qui sont décrits se rapportent à un ruminant, leur exécution fait donc appel à une méthode de la classe *Ruminant*.

2.1. Evènement ChoixActivite

L'événement *ChoixActivite* exécute la méthode *choixActivite* qui permet de choisir l'activité suivante de l'animal par appel à l'automate décisionnel. La méthode appelle ensuite la fonction associée à l'activité choisie à part l'action manger qui est particulière. Les activités possibles sont les suivantes :

- Manger ;
- Boire ;
- Rumine ;
- ReposLong ;
- ReposCourt ;

- LeadershipAlimentaire : l'animal est devenu leader, il va effectuer un déplacement long vers le meilleur site alimentaire de sa mémoire ;
- LeadershipBuvee : l'animal est devenu leader, il va effectuer un déplacement long vers le site de buvée le plus proche ;
- LeadershipRepos : l'animal est devenu leader, il va effectuer un déplacement long vers le site de repos le plus proche.

Si l'animal a choisi de manger, dans un premier temps on remet à zéro ses paramètres d'ingestion puis on insère un événement *choixCellule* pour choisir la première cellule qu'il va défolier.

2.2. Evènement ChoixCellule

Cet événement exécute la méthode *choixCellule* qui permet à l'animal de choisir une cellule parmi les quinze cellules de son champ de vision en faisant appel au générateur de trajectoires (Figure 3). Par contre, la position de l'animal n'est pas mise à jour, les coordonnées de la cellule sont enregistrées dans les attributs de l'animal *xChoix_*, *yChoix_*, sa direction dans *dirChoix_* et les informations telles que les différentes biomasses des quinze cellules dans *info*.

La méthode *choixCellule* insère ensuite un événement *FinDeplacement* pour tenir compte du déplacement de l'animal jusqu'à la cellule choisie. Cet événement sera activé *temps_de_deplacement* secondes plus tard.

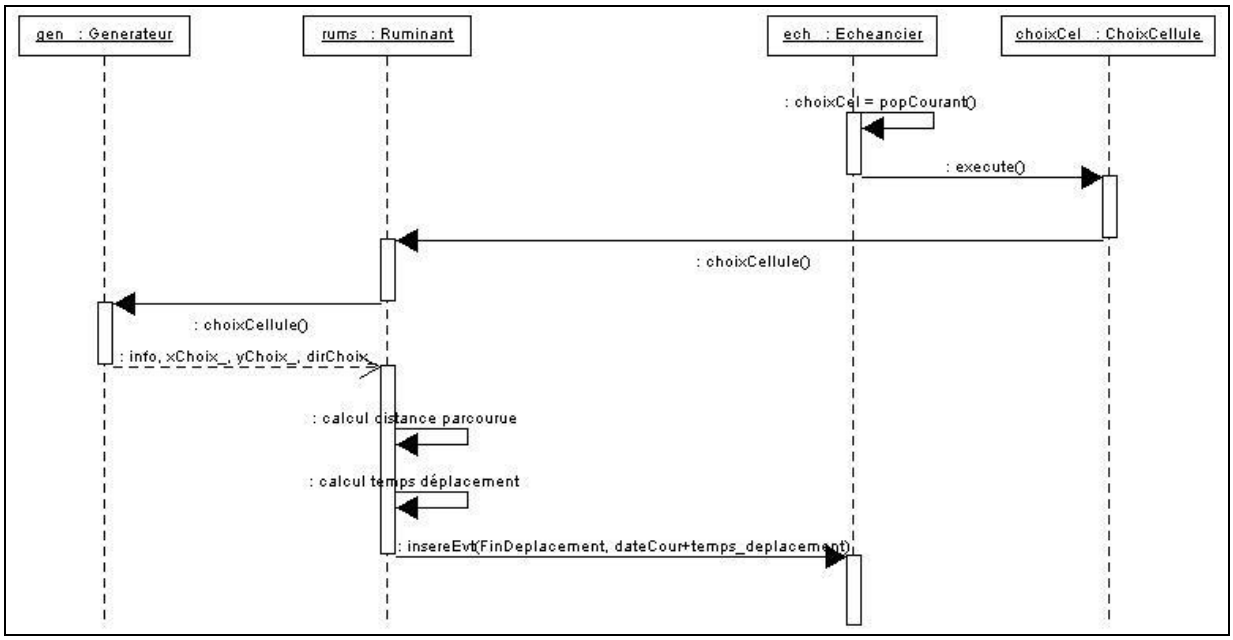


Figure 3 : Diagramme de séquences d'exécution de l'événement *ChoixCellule*

2.3. Evènement FinDeplacement

Cet événement est activé lorsque l'animal a terminé un déplacement de n'importe quel type, que ce soit un déplacement de proximité en pâturant, un déplacement pour rejoindre le troupeau ou le leader ou encore un déplacement long vers un autre site de la parcelle.

La méthode *FinDeplacement* qui lui est associée met à jour la position et la direction de l'animal. Sa nouvelle position est (*xChoix_*, *yChoix_*), sa nouvelle direction *dirChoix_*. Elle teste ensuite s'il n'y pas de collision sur cette cellule, c'est-à-dire si un animal ne l'occupe pas déjà. Si c'est le cas, l'animal fait un nouveau choix de cellule par l'insertion d'un événement *ChoixCellule* à la date courante. Par contre, si la cellule est libre, l'animal va continuer son activité ou enchaîner sur l'activité suivante, comme le montre la Figure 4.

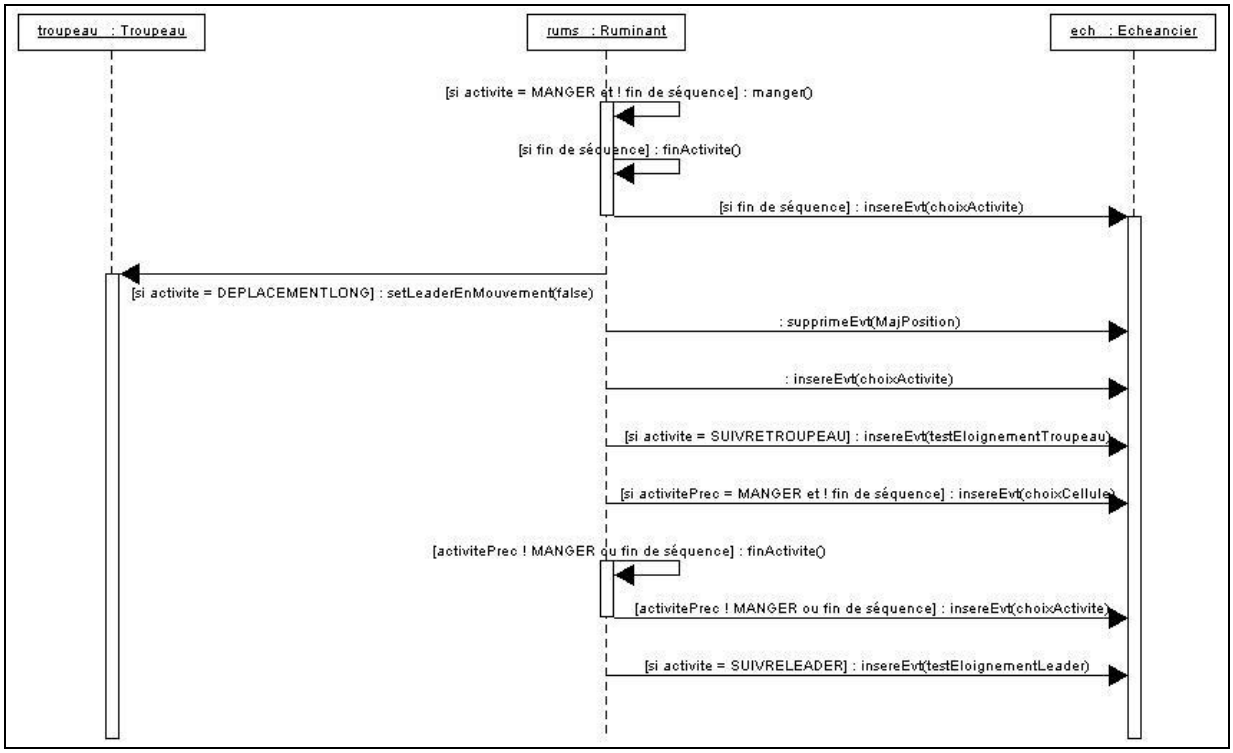


Figure 4 : Diagramme de séquences de la fin d'un déplacement de l'animal

Pour savoir ce que l'animal va faire maintenant qu'il a fini son déplacement, on regarde quelle activité était exécutée par l'animal lors de son déplacement. S'il était en train de manger, il s'agit d'un déplacement de proximité en pâturant, il va donc manger et défolier la cellule choisie. Par contre, si la séquence de défoliation est terminée, une mise à jour de ses paramètres de rumen est effectuée avec la fonction *finActivite* puis il choisit une nouvelle activité en insérant l'événement *choixActivite*. La séquence est terminée lorsqu'il mange depuis plus de 1200 secondes, soit la durée d'une séquence de défoliation paramétrée dans le fichier *ruminant.cfg*.

Si l'animal était en train d'effectuer un déplacement long, il est arrivé sur sa cellule de destination, il prévient donc le troupeau qu'il n'est plus en mouvement. Il supprime ensuite l'événement *MajPosition* qui servait à calculer sa trajectoire pendant son déplacement puis il choisit une nouvelle activité.

Si l'animal s'est rapproché ou éloigné du troupeau, il insère un événement *TestEloignementTroupeau* pour le test d'éloignement suivant. Ensuite il recommence à manger si son déplacement a interrompu une séquence de défoliation ou il fait un nouveau choix d'activité s'il était en train de boire, se reposer ou si la séquence de défoliation est terminée.

Enfin, si l'animal suivait le leader, il réinsère un test d'éloignement par rapport au leader pour continuer à s'en rapprocher.

2.4. Evènement MajPosition

Cet événement concerne uniquement l'animal leader en cours de déplacement long. Il sert à mettre à jour la position du leader régulièrement pour que les autres animaux sachent à tout moment où il se trouve et qu'ils puissent ainsi le suivre et pour les fichiers de trajectoire utilisés par la visionneuse.

La méthode *majPosition* fait d'abord appel à la méthode *calculPosition* pour déterminer la position actuelle de l'animal calculée à partir de sa position initiale et du temps depuis lequel il se déplace. Ensuite, les coordonnées de l'animal sont modifiées puis cette méthode insère l'événement *MajPosition* suivant.

2.5. Evènement TestEloignementLeader

Cet événement sert à vérifier qu'un animal ne soit ni trop proche ni trop éloigné de l'animal leader. La méthode *testEloignementLeader*, dont le fonctionnement est illustré par la Figure 5, demande au leader de calculer sa position courante puis calcule la distance entre l'animal et le leader. Si cette distance est inférieure à *d1Leader*, l'animal est trop proche du leader et va donc s'en éloigner. Si elle est supérieure à *d2Leader*, il teste s'il doit se rapprocher ou pas en calculant sa motivation à se déplacer. Par contre, si cette distance est inférieure à *d2Leader*, l'animal est bien positionné. Dans ce cas, si le leader est encore en mouvement, il doit continuer à se rapprocher au fur et à mesure, c'est pourquoi on insère un événement *TestEloignementLeader* dans l'échéancier. Par contre, si le leader a fini son déplacement long, l'animal arrête de le suivre et reprend une activité normale : il choisit une nouvelle activité et recommence les test d'éloignement par rapport au troupeau.

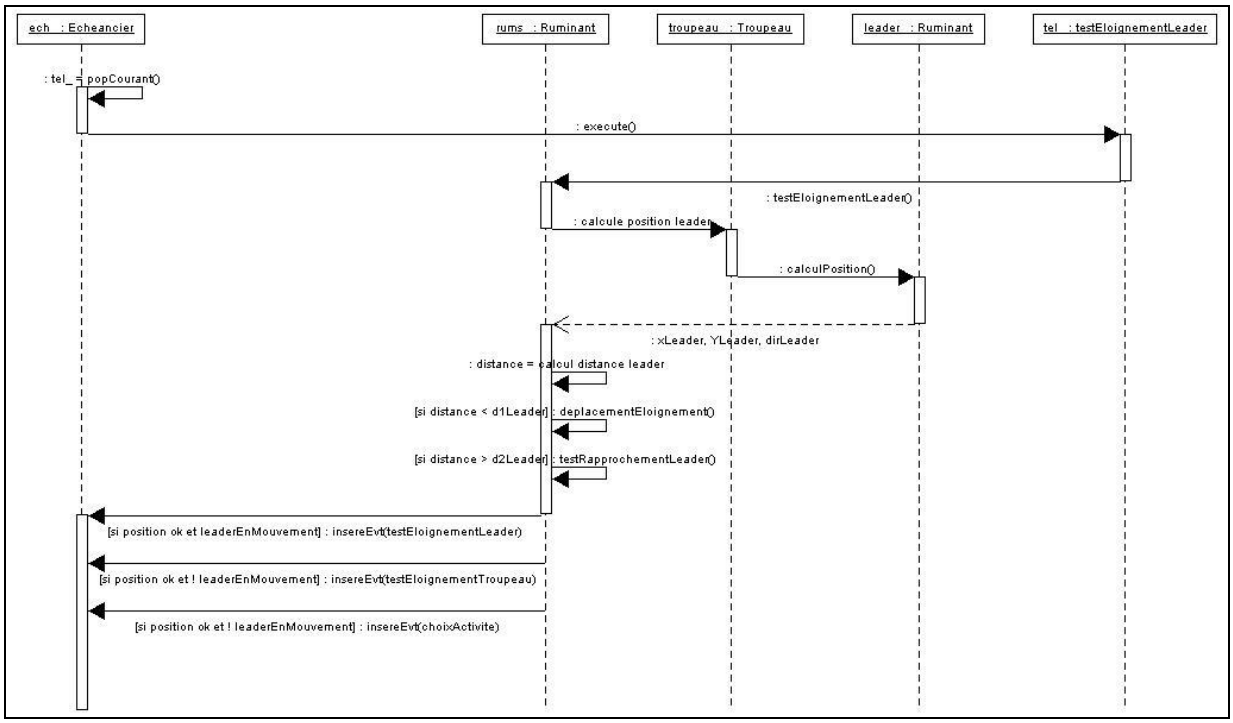


Figure 5 : Diagramme de séquences du test d'éloignement par rapport au leader

Si cette distance est supérieure à *d2Leader*, l'animal risque de se rapprocher du leader. Le rapprochement du leader ne se fait pas en une seule étape puisque le leader peut encore être en train de se déplacer. Il est expliqué dans la Figure 6. L'animal se déplace toutes les *tempsCalculTrajectoire* secondes, la valeur de ce paramètre étant paramétrée à 10 secondes pour les tests. L'animal demande donc au leader de calculer sa position actuelle, détermine quelle distance il peut parcourir en *tempsCalculTrajectoire* secondes et calcule une position intermédiaire entre le leader et lui qu'il pourra atteindre en *tempsCalculTrajectoire* secondes.

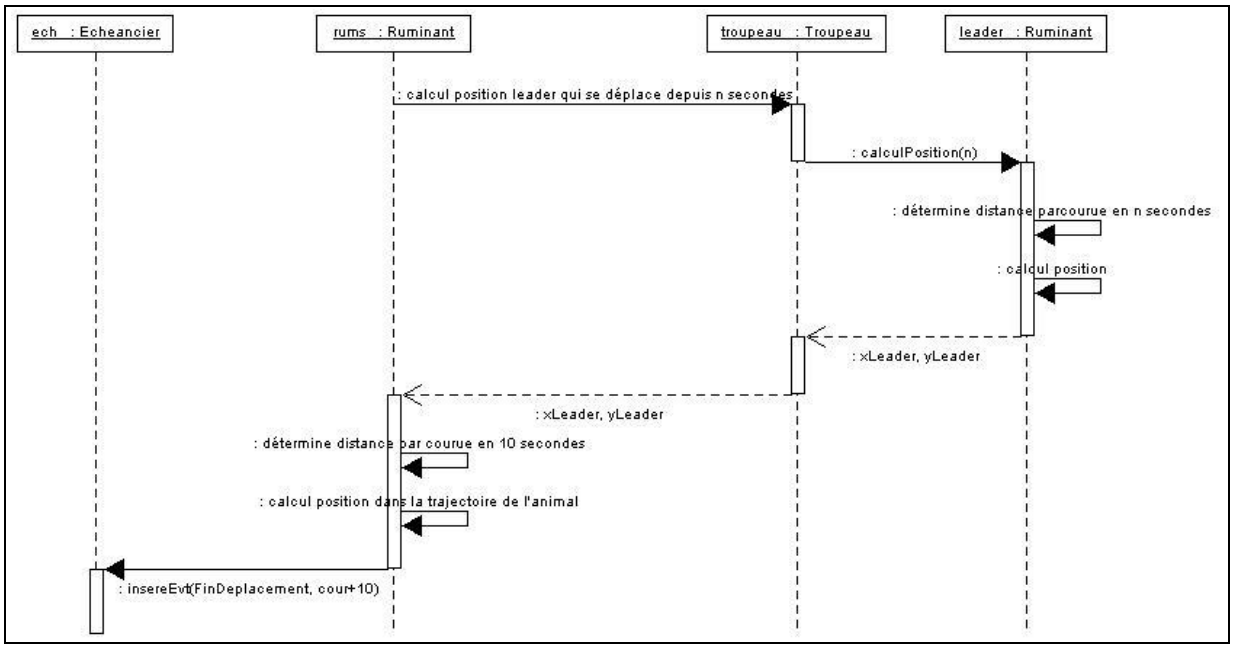


Figure 6 : Diagramme de séquences du rapprochement par rapport au leader

2.6. Evènement TestEloignementTroupeau

Cet événement sert à déterminer si l'animal n'est pas trop proche ni trop loin du troupeau. Comme le montre la Figure 7, la fonction *testEloignementTroupeau* calcule d'abord la distance *dproche* entre l'animal et son congénère le plus proche afin de vérifier si l'animal ne devra pas s'éloigner, ce qui est le cas si *dproche* est inférieure à la distance *d1Troupeau*. Pour déterminer si l'animal doit s'éloigner, on calcule une motivation à se déplacer pour l'animal puis on effectue un tirage aléatoire.

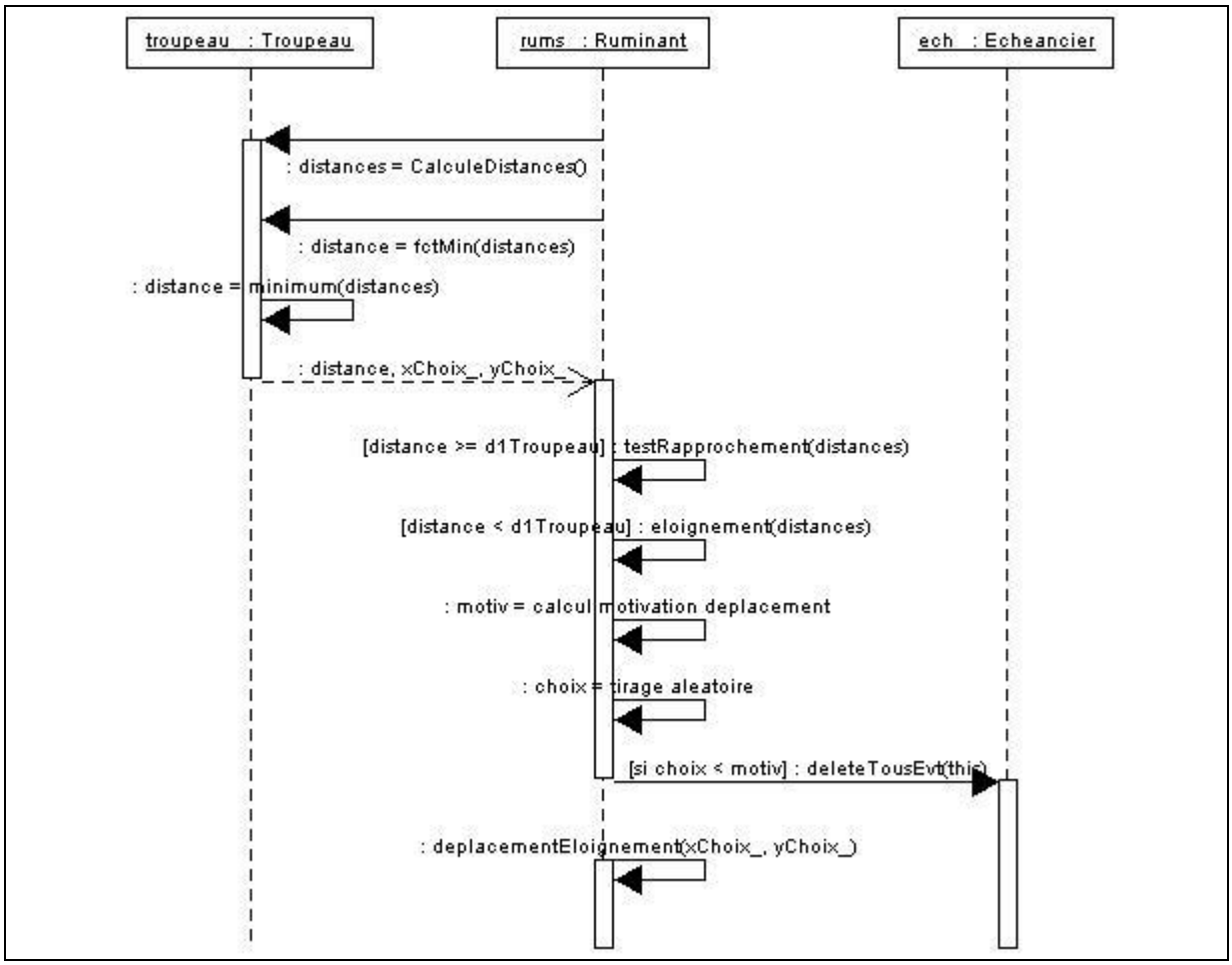


Figure 7 : Diagramme de séquences du test d'éloignement par rapport au troupeau

Si *dproche* est supérieure à *d1Troupeau*, on vérifie si l'animal doit se rapprocher ou pas. Pour cela, on calcule une distance *dtroupeau* entre l'animal et le reste du troupeau à partir de la fonction de calcul choisie, la fonction barycentre dans la Figure 8. L'animal se rapproche si *dtroupeau* est supérieure à *d3Troupeau*, calcule une motivation à se déplacer et fait un tirage aléatoire si *dtroupeau* est comprise entre *d2Troupeau* et *d3Troupeau*. Il faut remarquer que les déplacements, que ce soient l'éloignement ou le rapprochement, annulent toutes les autres actions en cours. L'animal se consacre au déplacement et supprime donc tous les évènements de l'échéancier qui le concernent.

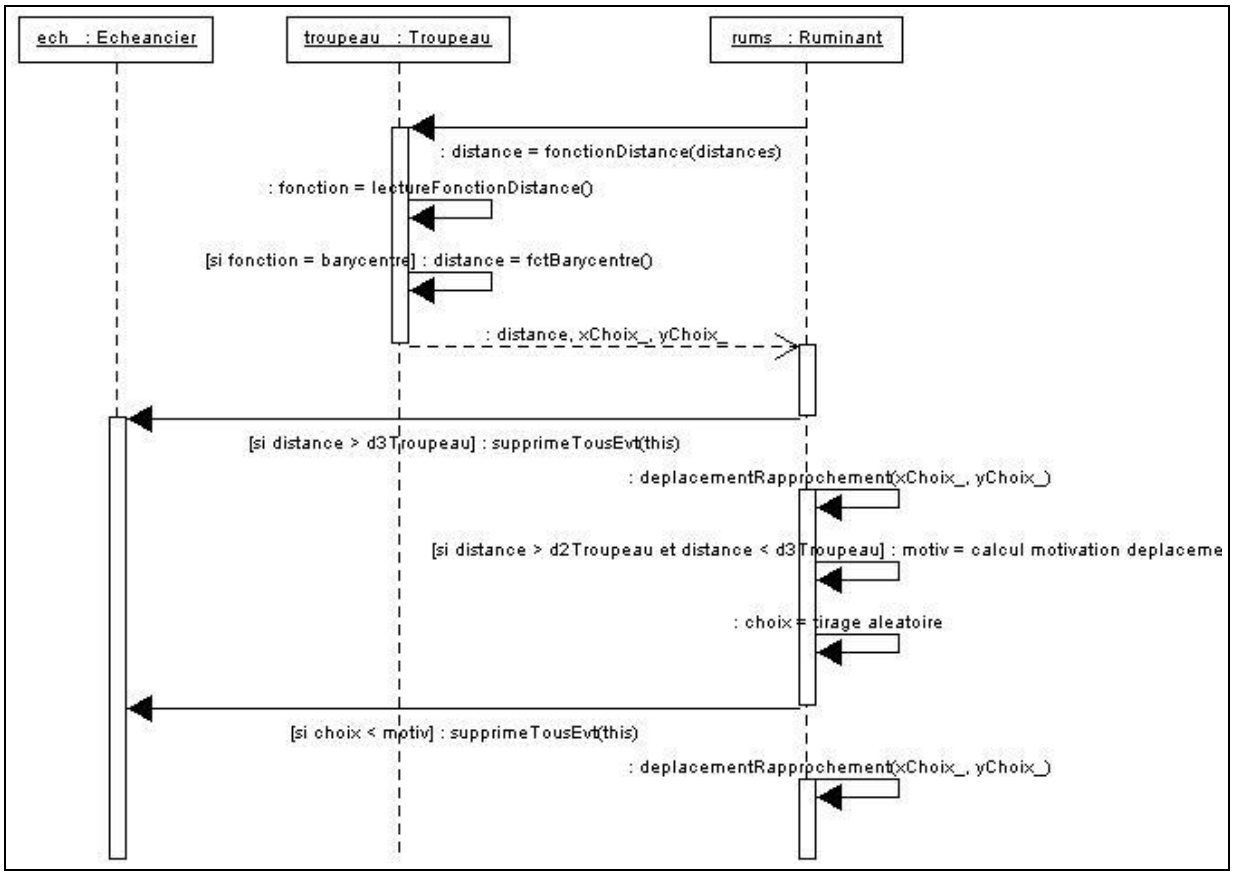


Figure 8 : Diagramme de séquences du test de rapprochement par rapport au troupeau

Annexe D : Diagrammes de classes du modèle

1. Module végétal

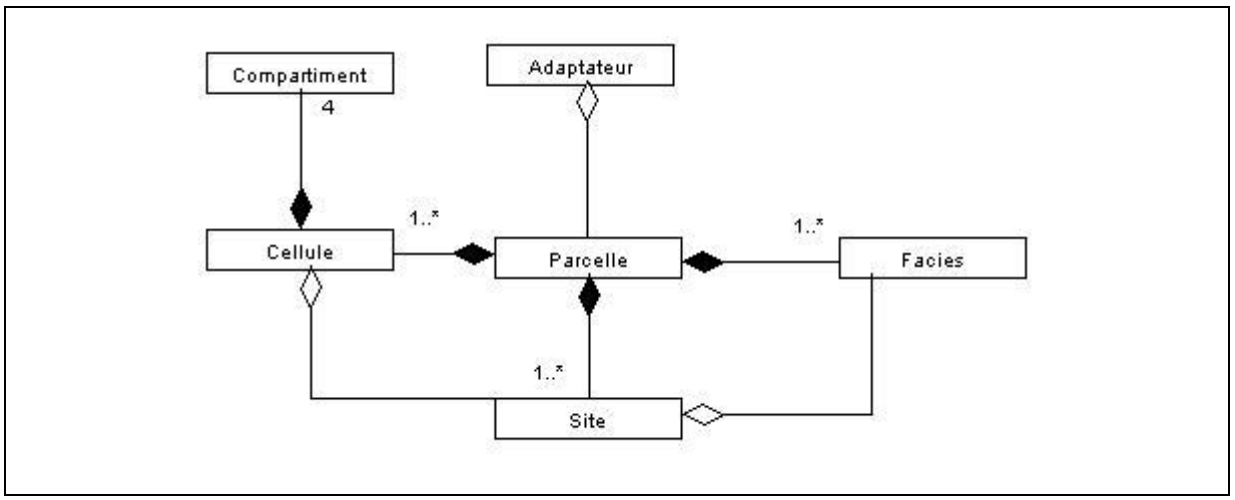


Figure 9 : Diagramme de classes général du module végétal

2. Module animal

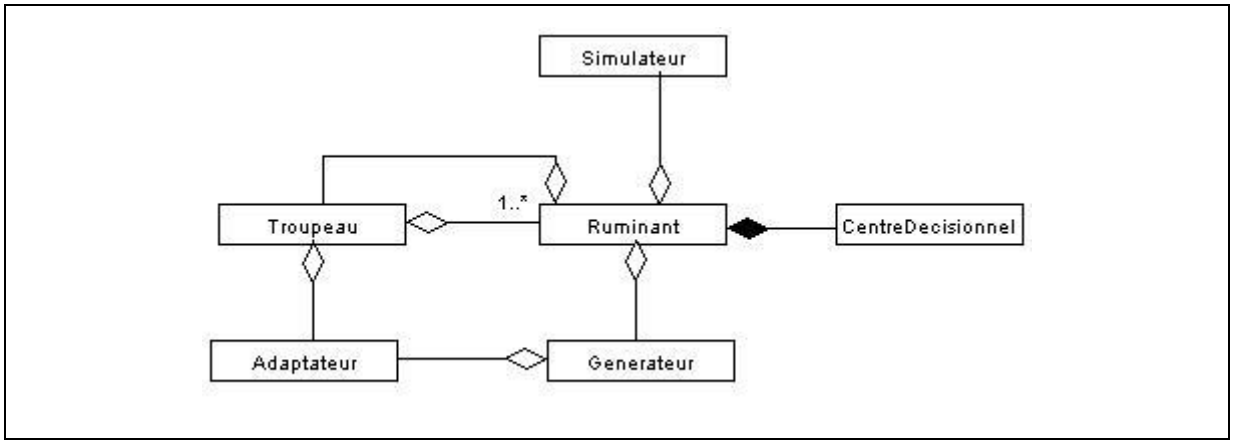


Figure 10 : Diagramme de classes général du module animal

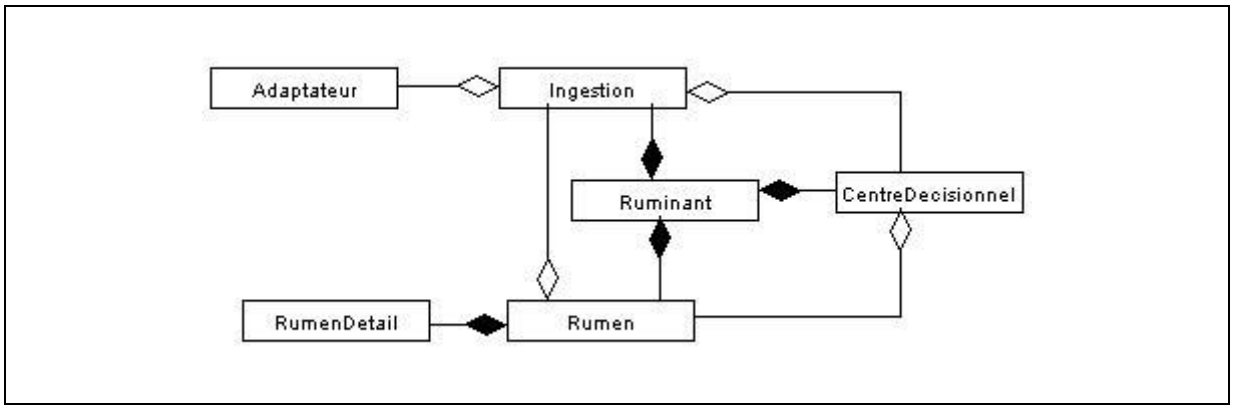


Figure 11 : Diagramme de classes particulier à l'animal

3. Module technique

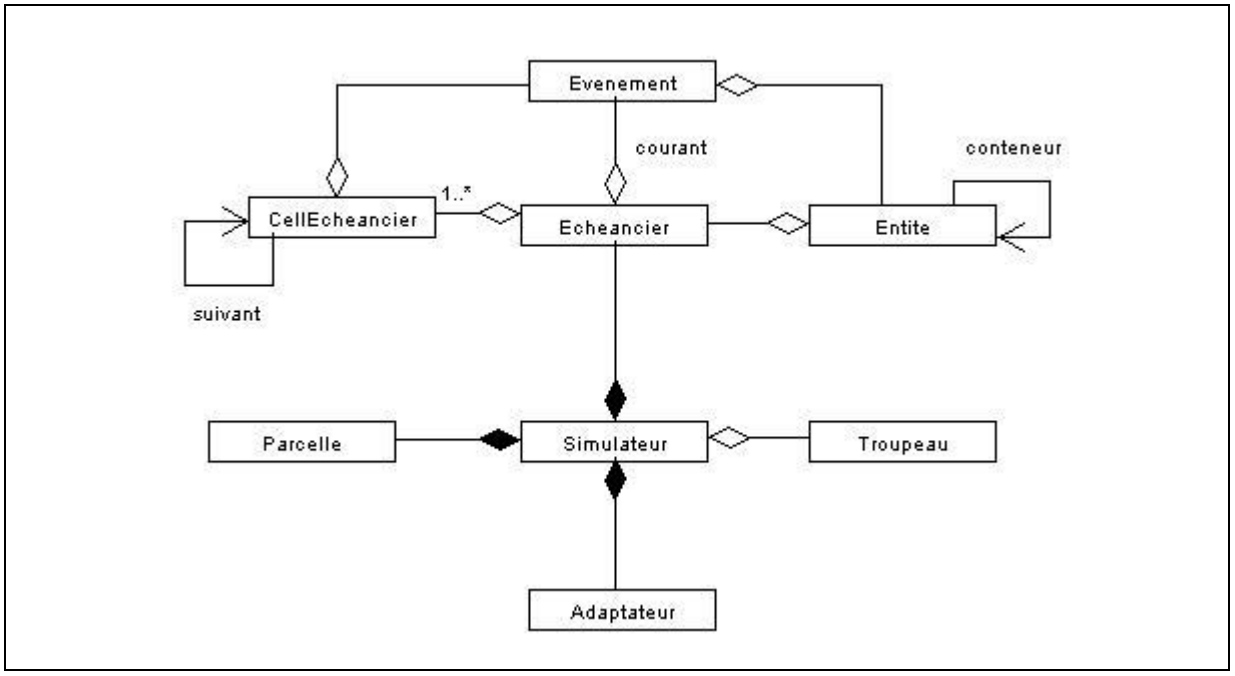


Figure 12 : Diagramme de classes général du module technique

Annexe E : Algorithmes

1. Eloignement d'un animal

Cet algorithme permet à un animal situé à la position (*x*, *y*) et regardant dans la direction *dir* de s'éloigner d'un autre animal de coordonnées (*xDest*, *yDest*). *Hauteur* et *largeur* correspondent aux dimensions de la parcelle.

```
# on demande à la parcelle les 15 cellules avec la direction dir
# les caractéristiques des cellules sont stockées dans info
gen_->getValeurs(&info, x, y, dir);

tant que((info[9].GetJ() < 0 || info[9].GetI() < 0 ||
          info[9].GetJ() >= largeur ||
          info[9].GetI() >= hauteur) &&
         (info[15].GetJ() < 0 || info[15].GetI() < 0 ||
          info[15].GetJ() >= largeur ||
          info[15].GetI() >= hauteur)) faire
# l'animal est tourné vers l'extérieur de la parcelle,
# on change sa direction pour la direction opposée

  si dir == 0 alors
    nDir = 3;
  sinon si dir == 1 alors
    nDir = 4;
  sinon si dir == 2 alors
    nDir = 5;
  sinon si dir == 3 alors
    nDir = 0;
  sinon si dir == 4 alors
    nDir = 1;
  sinon si dir == 5 alors
    nDir = 2;
  fin si

  dir = nDir;   # appel de la fonction gen_->setDir(nDir);
  # et on recommence
  gen_->getValeurs(&info, x, y, dir);
fait

# on calcule la distance entre la cellule n°9/n°15 et (xDest, yDest)
# on garde la cellule qui permettra de s'éloigner le plus possible
dist1 = distance(xDest, yDest, info[9].GetJ(), info[9].GetI());
dist2 = distance(xDest, yDest, info[15].GetJ(), info[15].GetI());

# si une des cellules sort de la parcelle, on choisit l'autre
si(info[9].GetJ() < 0 || info[9].GetI() < 0 ||
   info[9].GetJ() >= largeur ||
   info[9].GetI() >= hauteur ||
```

```
   (dist1 < dist2 &&
    info[15].GetJ() >= 0 && info[15].GetI() >= 0 &&
    info[15].GetJ() < largeur &&
     info[15].GetI() < hauteur)) alors  # cellule 15 choisie
    xChoix_ = info[15].GetJ();
    yChoix_ = info[15].GetI();
    dirChoix_  = 1;
sinon si(info[15].GetJ() < 0 ||
         info[15].GetI() < 0 ||
         info[15].GetJ() >= largeur ||
         info[15].GetI() >= hauteur ||
         dist1 > dist2) alors  # cellule 9 choisie
  xChoix_ = info[9].GetJ();
  yChoix_ = info[9].GetI();
  dirChoix_  = 5;
sinon
# on fait un tirage aléatoire pour savoir quelle cellule est choisie
  si(tirageAleatoire(0, 1) < 0.5) alors  # cellule 9 choisie
    xChoix_ = info[9].GetJ();
    yChoix_ = info[9].GetI();
    dirChoix_   = 5;
  sinon    # cellule 15 choisie
    xChoix_ = info[15].GetJ();
    yChoix_ = info[15].GetI();
    dirChoix_  = 1;
  fin si
fin si

# On calcule le temps du deplacement qui se fait tête haute
distanceParcourue = distance(x, y, xChoix_, yChoix_);
# temps de déplacement = distance / distance parcourue en 1 s
# et distance parcourue en 1 s est un tirage aléatoire dans
# l'intervalle [distMin, distMax]
tempsDeplacement = distanceParcourue /
(tirageAleatoire(Ruminant::distMin, Ruminant::distMax));

ech_->insereEvt(FinDeplacement::newEvt(this), dateCourante + (int)
tempsDeplacement);
```

2. Rapprochement d'un point de la parcelle

2.1.1. Algorithme de la fonction deplacementRapprochement

Cet algorithme est utilisé aussi bien pour qu'un animal puisse se rapprocher du leader, du reste du troupeau ou encore pour calculer la position actuelle du leader lorsqu'il effectue un déplacement long (cf. 1). On connaît les coordonnées des deux points de départ et de destination, A (*xDepart*, *yDepart*) et B (*xDest*, *ydest*). L'algorithme se déroule de façon suivante : on commence le parcours du point B et on calcule les coordonnées de toutes les cellules entre B et A jusqu'à obtenir une distance entre le point courant et B supérieure à *dist*. Les coordonnées sont calculées d'abord en faisant varier les abscisses, puis les ordonnées, et on choisit le meilleur point obtenu, celui qui se rapproche le plus de B.

```
xPrec1 = -1, yPrec1 = -1;
xPrec2 = -1, yPrec2 = -1;

# on fait d'abord varier les x
si xDepart != xDest alors
  rapprochementX(xPrec1, yPrec1, xDepart, yDepart, xDest, yDest,
dist);
fin si
# puis en faisant varier y
si yDepart != yDest alors
  rapprochementY(xPrec2, yPrec2, xDepart, yDepart, xDest, yDest,
dist);
fin si
# on choisit entre les deux point (xPrec1, yPrec1) et (xPrec2,
yPrec2) celui qui respecte le mieux la distance désirée
si((xPrec2 == -1 && yPrec2 == -1) ||
   (xPrec1 != -1 && yPrec1 != -1 &&
   (dist - distance(xDest, yDest, xPrec1, yPrec1)) <
   (dist - distance(xDest, yDest, xPrec2, yPrec2)))) alors
    xChoix_ = xPrec1;
    yChoix_ = yPrec1;
sinon si((xPrec1 == -1 && yPrec1 == -1) ||
         ((dist - distance(xDest, yDest, xPrec1, yPrec1)) >
          (dist - distance(xDest, yDest, xPrec2, yPrec2)))) alors
  xChoix_ = xPrec2;
  yChoix_ = yPrec2;
sinon      # deux distances égales => tirage aléatoire
  si(tirageAleatoire(0, 1) < 0.5) alors  # (xPrec1, yPrec1) choisi
    xChoix_ = xPrec1;
    yChoix_ = yPrec1;
  sinon    # (xPrec2, yPrec2) choisi
    xChoix_ = xPrec2;
    yChoix_ = yPrec2;
  fin si
fin si

# calcul de la nouvelle direction de l'animal
calculDirection();
```

2.1.2. Algorithme de la fonction rapprochementX

Cet algorithme sert à déterminer les coordonnées de toutes les cellules intermédiaires entre deux points A (*xDepart*, *yDepart*) et B (*xDest*, *yDest*) jusqu'à obtenir une distance par rapport à B supérieure à la distance *dist* en faisant varier les abscisses. *Largeur* et *hauteur* désignent les dimensions de la parcelle.

```
si(xDest - xDepart != 0 && yDest - yDepart != 0) alors
  a = (yDest - yDepart) / (xDest - xDepart);
  b  = yDepart - (xDepart * a);
fin si

si(xDepart < xDest) alors
  modif = -1;
```

```
sinon
  modif = 1;
fin si

si(yDepart != yDest) alors
  xCour = inXDest;
  yCour = inYDest;
  dCour = 0.0;
  tant que(dCour < dist &&
           (xCour >= 0 && xCour < largeur) &&
           (yCour >= 0 && yCour < hauteur)) faire
    xPrec = xCour;
    yPrec = yCour;
    xCour += modif;
    # (xCour, yCour) est situé sur la droite liant les points A et B
    yCour = a * xCour) + b;
    dCour = distance(xDest, yDest, xCour, yCour);
  fait
sinon  # droite d'équation y = yDepart
  xCour = xDest;
  yCour = yDest;
  dCour = 0.0;
  tant que(dCour < dist &&
            (xCour >= 0 && xCour < largeur)) faire
    xPrec = xCour;
    yPrec = yCour;
    xCour += modif;
    dCour = distance(inXDest, inYDest, xCour, yCour);
  fait
fin si
```

L'algorithme de la fonction *rapprochementY* est presque semblable à celui de la fonction *rapprochementX*, c'est pourquoi il ne sera pas donné dans ce rapport.

2.1.3. Algorithme du rapprochement par rapport au leader

L'animal de coordonnées (*x*, *y*) cherche à se rapprocher le plus possible du leader dont les coordonnées sont (*xLeader*, *yLeader*). L'animal se rapproche petit à petit, dans le cas où le leader soit encore en train de se déplacer, pour ne pas avoir à changer complètement de trajectoire une fois qu'il aurait fini son déplacement.

```
# L'animal se rapproche tous les tempsCalculTrajectoire
# distance de déplacement = tempsCalculTrajectoire * vitesse
# et vitesse est un tirage aléatoire dans l'intervalle
# [distMin, distMax]
vitesse = (tirageAleatoire(Ruminant::distMin, Ruminant::distMax));
distanceParcourue = Ruminant ::tempsCalculTrajectoire * vitesse;

si(distanceParcourue < distance(x, y, xLeader, yLeader)) alors
  # L'animal ne finira pas son rapprochement
  deplacementRapprochement(xLeader, yLeader, x, y,
distanceParcourue);
```

```
  ech_->insereEvt(FinDeplacement::newEvt(this), dateCourante +
Ruminant::tempsCalculTrajectoire);
sinon
  # L'animal finira son rapprochement
  deplacementRapprochement(x, y, xLeader, yLeader,
Ruminant::d2Leader);
  # On calcule le temps du deplacement tête haute
  dist = distance(x, y, xChoix_, yChoix_);
  tempsDeplacement = dist / vitesse;

  ech_->insereEvt(FinDeplacement::newEvt(this), dateCourante +
tempsDeplacement);
fin si
```

3. Calcul de la position du leader à un instant donné

Cet algorithme permet de déterminer la position du leader à un instant donné sachant qu'il se déplace dans la direction *dirChoix_* vers la position (*xChoix_*, *yChoix_*). On sait aussi que le leader se déplace depuis *N* secondes à la vitesse *vitesseDeplacementLeader* et que sa position actuelle est (*xLeader*, *yLeader*) et sa direction *dirLeader*. La nouvelle position du leader, celle que l'on calcule, est (*xDest*, *yDest*), sa nouvelle direction *dirDest*.

```
si leaderEnMouvement alors
# on calcule la position du leader en cours de déplacement
   tempX = xChoix_;
   tempY = yChoix_;
   tempDir = dirChoix_;
   # on récupère la position du leader avant son déplacement dans
   # (xInit, yInit)
   troupeau_->getPositionInitLeader(&xInit, &yInit);

   # calcul de la distance parcourue = dureeDeplacement * vitesse
  distanceParcourue = inDureeDeplacement * vitesseDeplacementLeader;

  deplacementRapprochement(tempX, tempY, xInit, yInit,
distanceParcourue);
  xDest    = xChoix_;
  yDest    = yChoix_;
  dirDest  = dirChoix_;
  # on remet à jour la destination de l'animal qui a été modifiée
  xChoix_    = tempX;
  yChoix_    = tempY;
  dirChoix_  = tempDir;
sinon      # le leader est arrivé à destination
  xDest = xLeader;
  yDest = yLeader;
  dirDest  = dirLeader;
fin si
```

Il faut noter que *xChoix_*, *yChoix_* et *dirChoix_* qui caractérisent la position de destination du leader doivent être sauvegardés dans des variables temporaires parce que ces attributs sont modifiés par la méthode *deplacementRapprochement*, les nouvelles valeurs caractérisent la position où l'animal se trouve au moment des calculs.