Benjamin Betti

Projet POO:
Modèle Proies-Prédateurs

Cahier de Spécifications

Historique des révisions

Date	Description et justification de la modification	Auteur	Pages/ chapitre	Edition/ Revision
12/02/2018	**Création**	**Betti**		**0.0**

Table des matières

1. Introduction

Les spécifications présentées dans ce document visent à décrire comment a été implémentée l'application demandée par le client, à savoir un *automate cellulaire* simulant un modèle d'évolution de populations suivant une relation proie-prédateur, sur la base de deux espèces: le Mouton symbolisant la proie, et le Loup symbolisant le prédateur.

Les fonctions principales de cette application sont la création des proies et prédateurs sur un terrain, ainsi que la simulation de l'évolution de leurs populations via un affichage au tour par tour. L'application pourra à tout moment être mise en pause, et des unités pourront alors à nouveau être ajoutées sur le terrain.

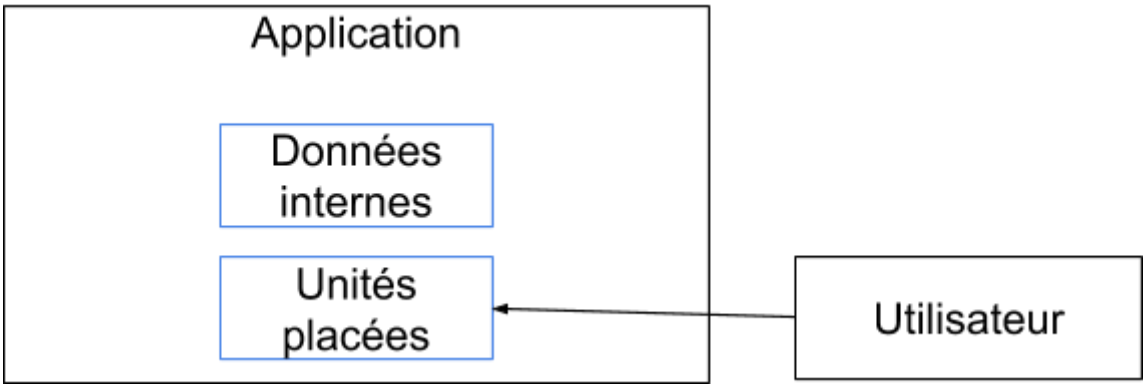
Le terrain est de taille fixe, carré et boucle sur lui-même (ainsi une unité sortant d'un côté se retrouve visuellement de l'autre côté).

2. Exigences opérationnelles

L'utilisateur se voit proposer une application *standalone*, simple d'utilisation.

Les contraintes de performance sont quasi inexistantes: l'application tourne sur un ordinateur *mainstream* (2 Go de Ram, 10 Go d'espace disque, processeur mono ou multicore à 2 Ghz) sans la moindre difficulté, attendu que la majeure partie du temps est passée à attendre entre deux tours successifs (ce point dépendant bien évidemment de la vitesse de la simulation).

Il n'y a pas de contraintes de sécurité, ni d'intégrité.



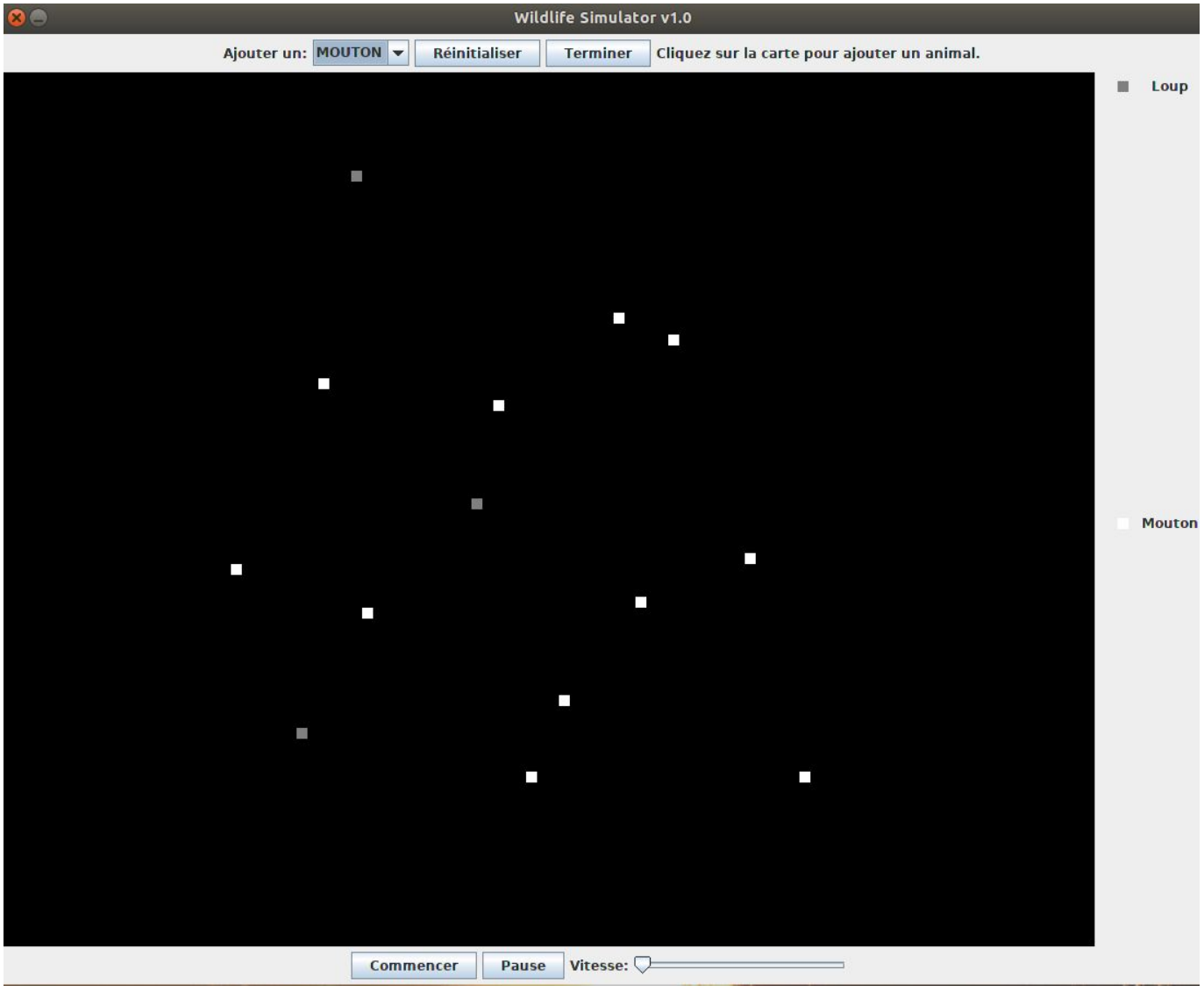
La *simulation* en elle-même est autonome, mais l'on garde la possibilité de la mettre en pause/fonction, d'ajouter des unités et de changer sa vitesse à tout moment.

3. Interfaces

L'interface homme-machine est simple, intuitive et la plus explicite possible.

L'intégralité des donnés générées par la simulation est transitoire et il n'y a donc pas d'entrées/sorties de fichiers ni de connexions à des éléments externes.

L'application ne dispose en fonctionnement normal que d'une seule fenêtre, qui permet à la fois de placer des unités, de contrôler la simulation et de l'observer.



Fenêtre principale de l'application

4. Conception

L'application est réalisée en langage *Java*, pour son paradigme objet et sa grande portabilité.

L' interface homme-machine est créée à partir de la librairie *Swing* du langage Java.

Le coeur de l'application est la simulation d'un modèle de prédation, incluant différentes espèces (en l'état actuel 2 seulement: des loups et des mouton) se déplaçant dans un monde fermé. La simulation se termine dès lors que toutes les unités sont décédées.

4.1 Initialisation d'une simulation

Comme dans toute simulation, la première étape est bien évidemment l'initialisation, qui se déroule très simplement dans la fenêtre de l'application. Bien que l'on considère la présence d'une phase de simulation et d'une phase d'initialisation, il est en fait possible d'ajouter des unités sur la simulation à tout moment.

4.2 Éléments d'une simulation

Une simulation fait interagir les différents éléments suivants:

4.2.1 Monde

Le monde est constitué d'un ensemble de cases, sur lesquelles peut se tenir au plus une unité. Le monde "boucle" sur lui même, ainsi toute unité sortant par exemple à droite réapparaître à à gauche.

4.2.2 Unités

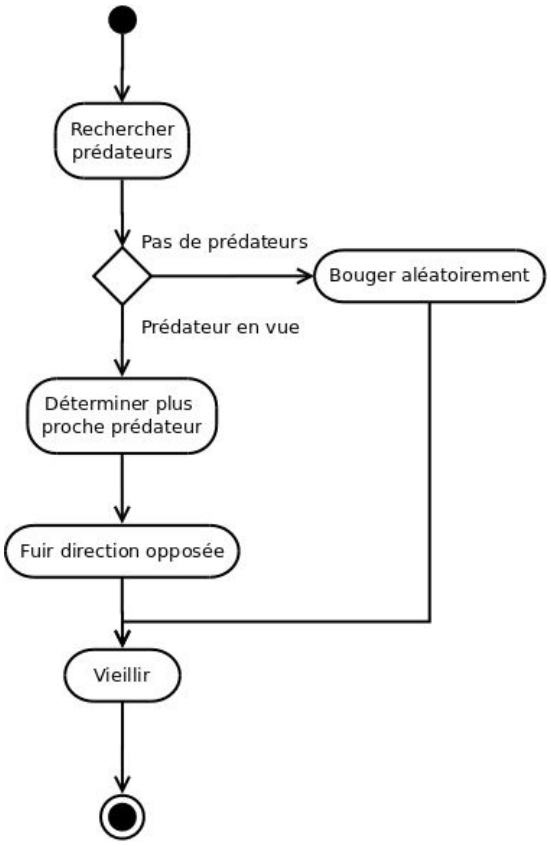
Une unité est un animal, présent sur une case, pouvant agir par lui-même et interagir avec les unités environnantes. Chaque *espèce* dispose d'un ensemble de Proies et/ou de Prédateurs, d'une vitesse de déplacement, d'une durée de vie qui peut être rallongée en chassant d'autres unités (pour les Prédateurs uniquement), d'une valeur nutritive pour ses prédateurs et d'un champ de vision limité.

4.3 Déroulement d'une simulation

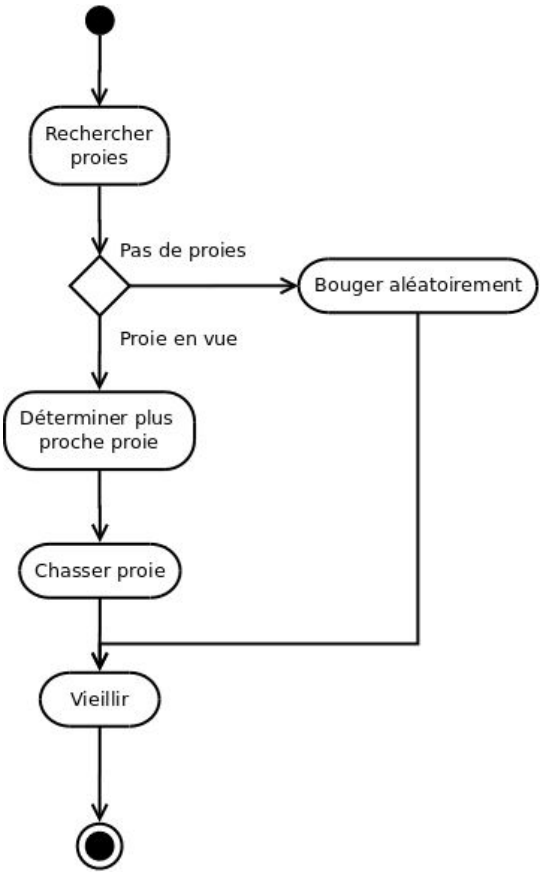
La simulation se déroule en tour par tour, de manière autonome. Elle s'achève au clic sur le bouton "Terminer" ou lorsque toutes les unités sont mortes. Les unités peuvent interagir de différentes manières. Un tour d'une unité passe par les étapes suivantes:

- détection des unités (proies et/ou prédateurs) présentes dans son champ de vision
- mouvement adapté à cette détection:
    - fuite dans la direction opposée si un prédateur est présent
    - chasse d'une proie si l'une d'entre elles est à portée
- si rien n'a été détecté, mouvement aléatoire à la découverte du monde

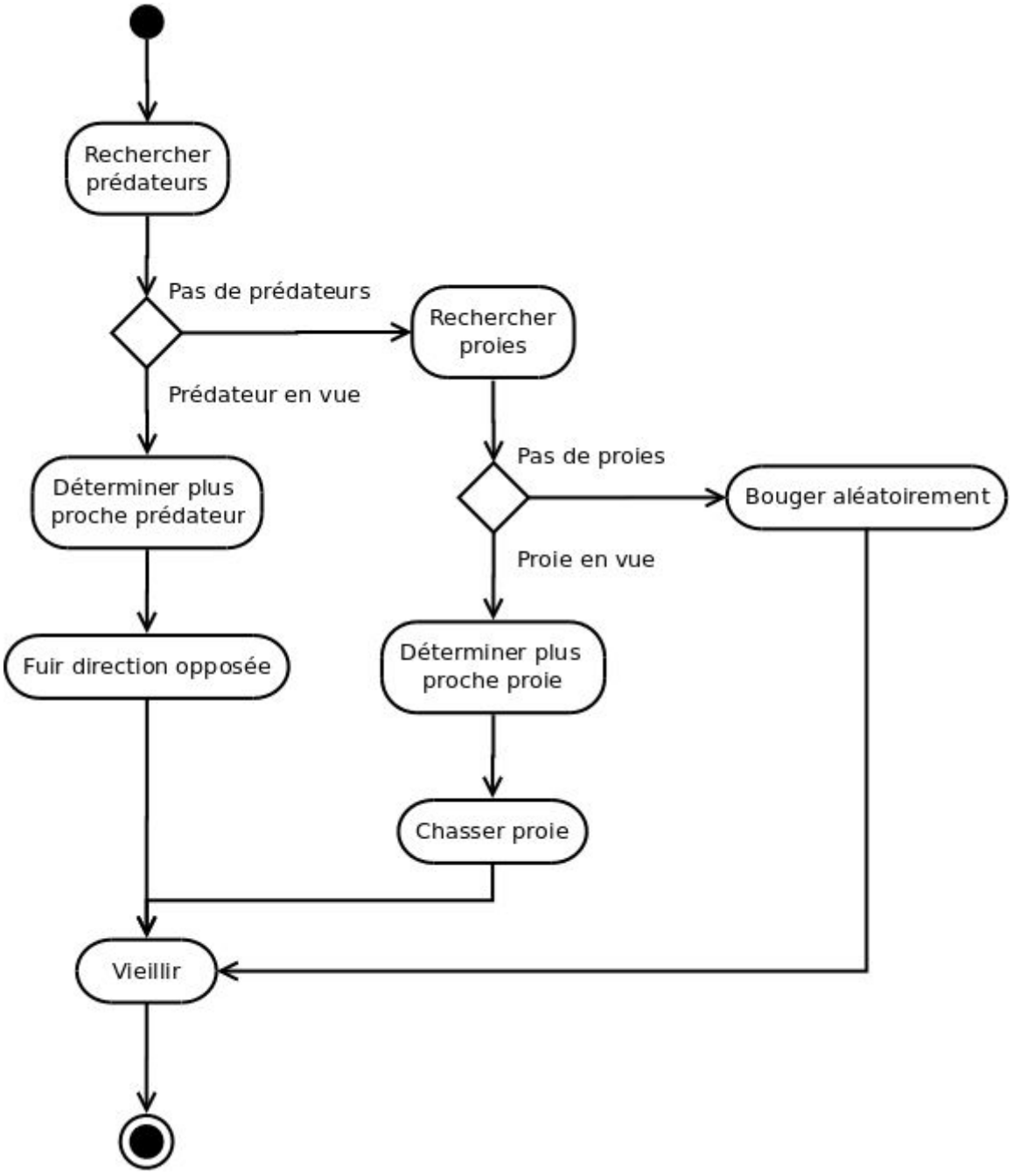
Pour une proie:



Pour un prédateur:



Pour un animal étant à la foie proie et prédateur:

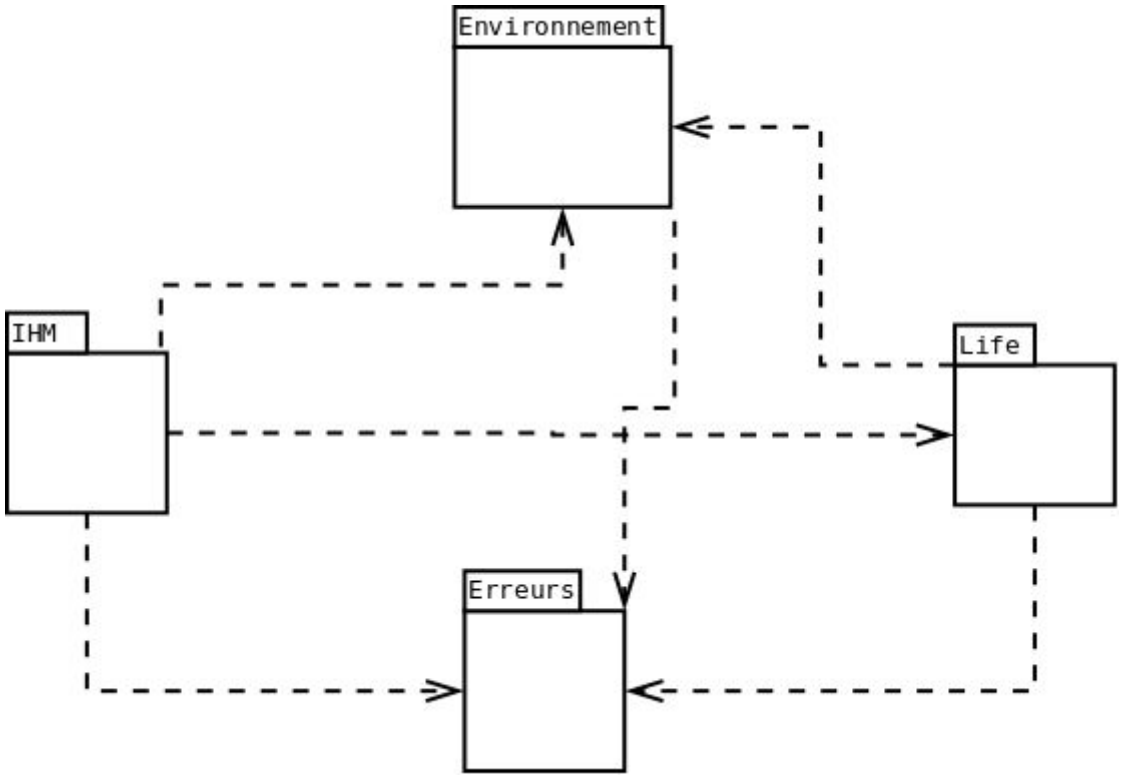


5. Implémentation

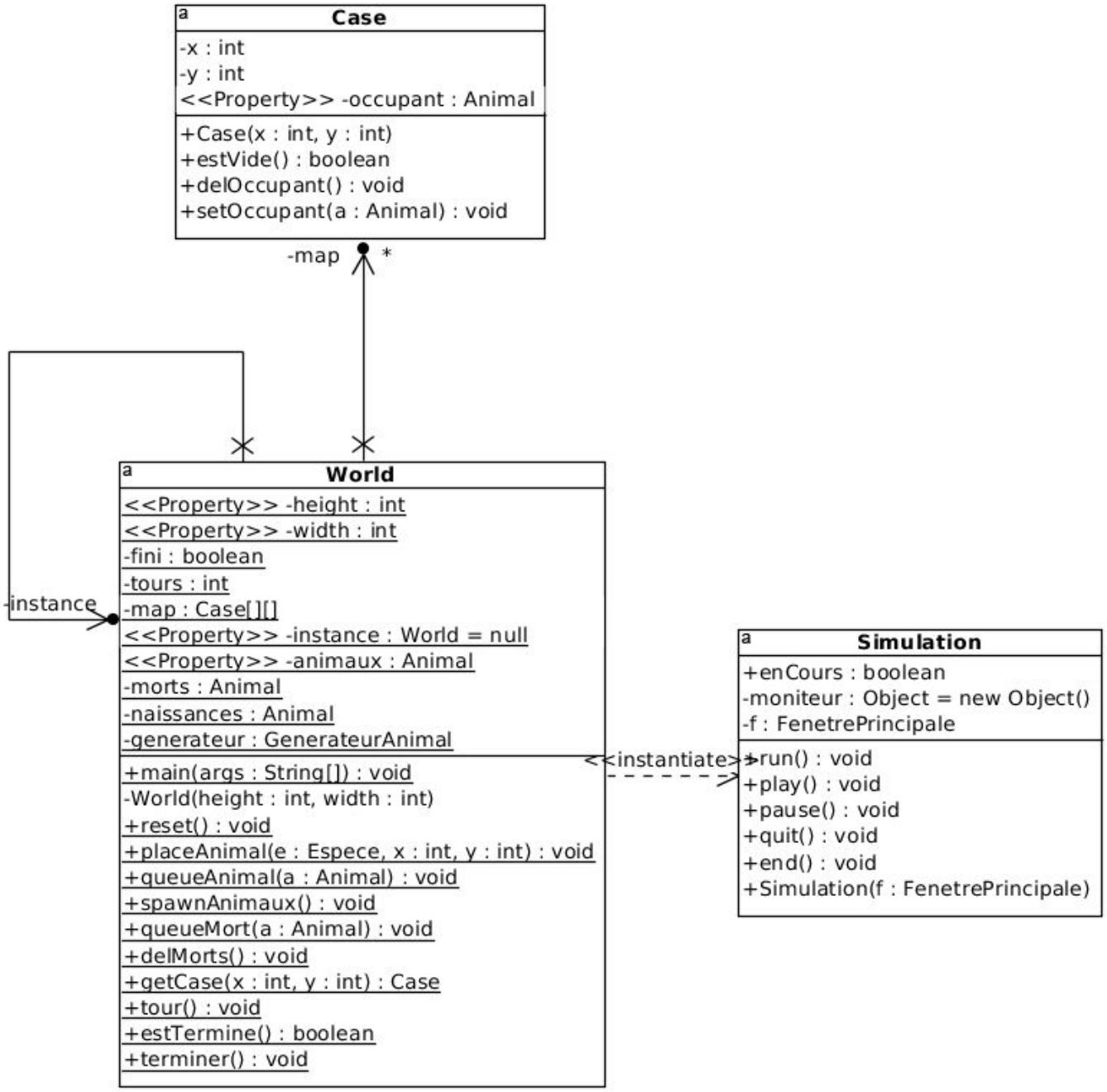
5.1 Découpage logique

Tout d'abord, on a le découpage en packages suivant:

- Environnement: Ensemble de classes servant à représenter le Monde et la Simulation.
- IHM: Ensemble de classes servant à l'interaction utilisateur.
- Erreurs: Classes d'erreurs et classes permettant leur affichage.
- Life: Ensemble de classes permettant la gestion des comportements et des unités, et leur création et évolution.



5.2 Package Environnement



5.2.1 Classe World

Cette classe est un singleton regroupant l'ensemble des variables et/ou éléments devant être uniques. Elle sert de point d'entrée et s'occupe de l'apparition/disparition des unités, ainsi que de l'instanciation du monde dans lequel elles évoluent.

Ses attributs sont les suivants:

- height de type int: Indique la hauteur du monde initialisé,
- width de type int: indique la largeur du monde initialisé,
- fini de type booleen: indique à la fenêtre et à la simulation que la simulation est terminée,
- tours de type int: permet de comptabiliser le nombre de tours effectués par la simulation,
- map de type tableau 2D de Case: représente la "carte" du monde, c'est à dire l'ensemble des cases qui le composent,
- instance de type World qui permet de garder l'unique instance de cette classe,
- morts et naissance, listes d'objets de type Animal: gardent en mémoire les unités devant être détruites et/ou créées à la fin du tour,
- animaux, liste de type Animal: liste les unités présentes et vivantes, qui doivent donc jouer le prochain tour,
- generateur, de type GenerateurAnimal: contient la fabrique qui permettra d'instancier les unités.

Ses méthodes:

- main: point d'entrée du programme,
- un constructeur permettant de créer un monde du taille définie,
- reset: méthode permettant une réinitialisation du monde si demandé par l'utilisateur,
- placeAnimal: méthode permettant d'insérer un Animal à une position donnée,
- queueAnimal: méthode permettant d'ajouter un Animal à la liste d'Animaux devant être spawnés pour le prochain tour,
- spawnAnimaux: méthode appelée à la fin du tour pour insérer les Animaux mis en attente,
- queueMort: contrepartie de queueAnimal pour les animaux morts,
- delMorts: méthode supprimant les animaux morts du monde à la fin du tour,
- getCase: permet d'accéder à une case définie par ses coordonnées,
- tour: méthode implémentant un tour, c'est à dire appelant les uns après les autres les unités devant participer à ce tour,
- estTermine: indicateur de fin de simulation,
- terminer: méthode permettant d'arrêter la simulation sur demande de l'utilisateur.

5.2.2 Classe Case

Classe permettant de constituer la carte sur laquelle évolueront les unités. Elle permet de placer les unités, et d'avoir un moyen de lancer des actions à portée géographique (par exemple la détection). Par souci de simplicité, il a été choisi d'utiliser une distance dite de Manhattan pour ce type d'actions.

Attributs:

- x de type int: position en largeur de la case considérée,
- y de type int; position en hauteur de la case considérée,

- occupant de type animal: occupant actuel de la case (si il y en a un).

Méthodes:

- estVide: méthode permettant de vérifier qu'une case est disponible pour y placer une unité,
- delOccupant: méthode permettant de supprimer un occupant, par exemple si il se déplace ou meurt,
- setOccupant: méthode permettant d'ajouter un occupant à la case.

5.2.3 Classe Simulation

Classe permettant de créer un thread et de le contrôler. Se charge d'appeler la méthode tour de la classe World.

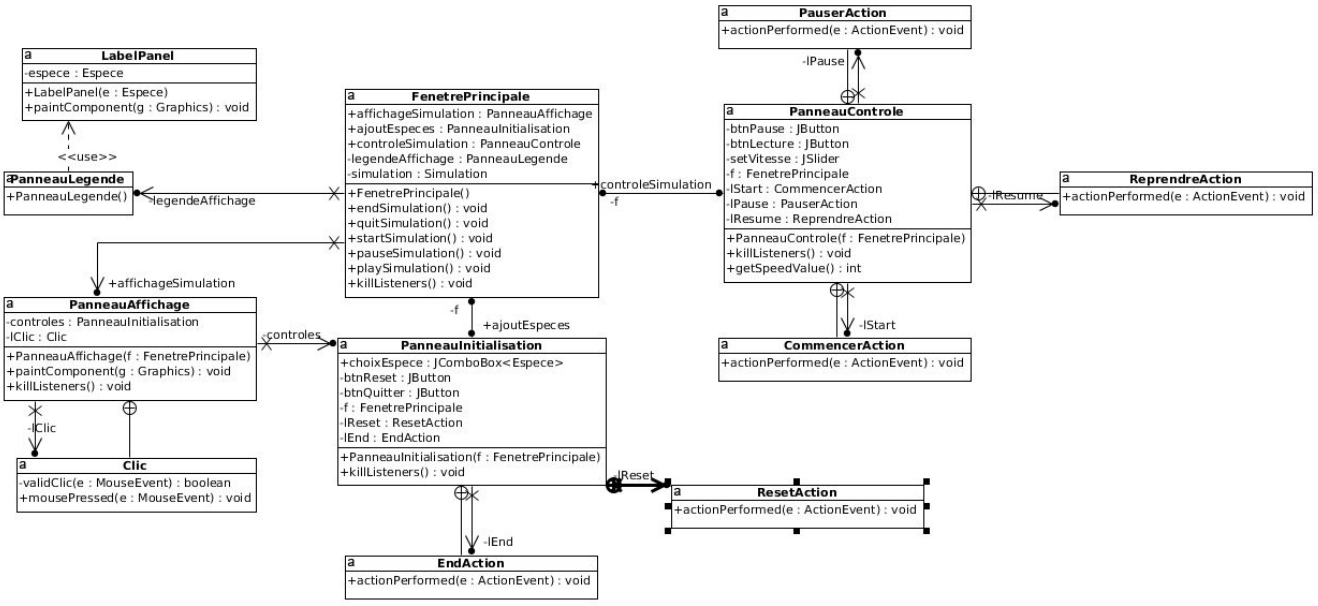
Attributs:

- enCours: booléen permettant de déclencher/pauser la simulation quand nécessaire,
- moniteur: objet servant de moniteur pour pauser/reprendre la simulation depuis les contrôles de la fenêtre de l'application,
- f: attribut permettant de se référer à la fenêtre ayant lancé cette simulation.

Méthodes:

- run: méthode permettant de lancer les actions à effectuer durant la simulation, point d'entrée du thread,
- play: méthode permettant de reprendre la simulation après une pause, ou de la déclencher si cela n'a pas encore été fait,
- pause: méthode permettant de pauser la simulation,
- quit: méthode permettant à l'utilisateur de demander l'arrêt de la simulation,
- end: méthode appelée en cas de fin naturelle de la simulation.

5.3 Package IHM



5.3.1 Classe FenetrePrincipale

Cette classe représente la fenêtre de l'application avec laquelle interagira l'utilisateur. Elle dérive de la classe JFrame de la bibliothèque Swing.

Ses attributs sont:

- affichageSimulation, ajoutEspeces, controleSimulation, legendeAffichage: les différentes parties constituant cette fenêtre.
- simulation: une référence vers la simulation qu'elle va déclencher, contrôler et afficher.

Ses méthodes:

- endSimulation et quitSimulation: mettent fin à la simulation, respectivement de manière naturelle et imposée par l'utilisateur,
- startSimulation, pauseSimulation, playSimulation: permettent dans l'ordre de lancer, mettre en pause et reprendre la simulation sur demande de l'utilisateur,
- killListeners: permet de désactiver un certain nombre d'interactions à la fin de la simulation, ou lorsque celle-ci est en pause.

5.3.2 Classe PanneauInitialisation

Cette classe représente le panneau de l'application qui permet d'initialiser la simulation.

Ses attributs sont:

- choixEspece: un sélecteur permettant de choisir quel type d'unité on souhaite placer,
- btnReset: un bouton permettant de réinitialiser le monde,
- btnQuitter: un boutton permettant de quitter l'application à tout moment,
- resetAction et endAction: des listeners permettant d'implémenter les actions des boutons décrits ci-dessus.

5.3.3 Classe PanneauAffichage

Ce panneau affiche une représentation de l'état actuel du monde.

Son listener clic lui permet de placer l'unité sélectionnée dans le PanneauInitialisation à l'endroit où l'utilisateur effectue un simple clic gauche, et ce à n'importe quel moment de la simulation, tant qu'elle n'est pas terminée naturellement ou artificiellement.

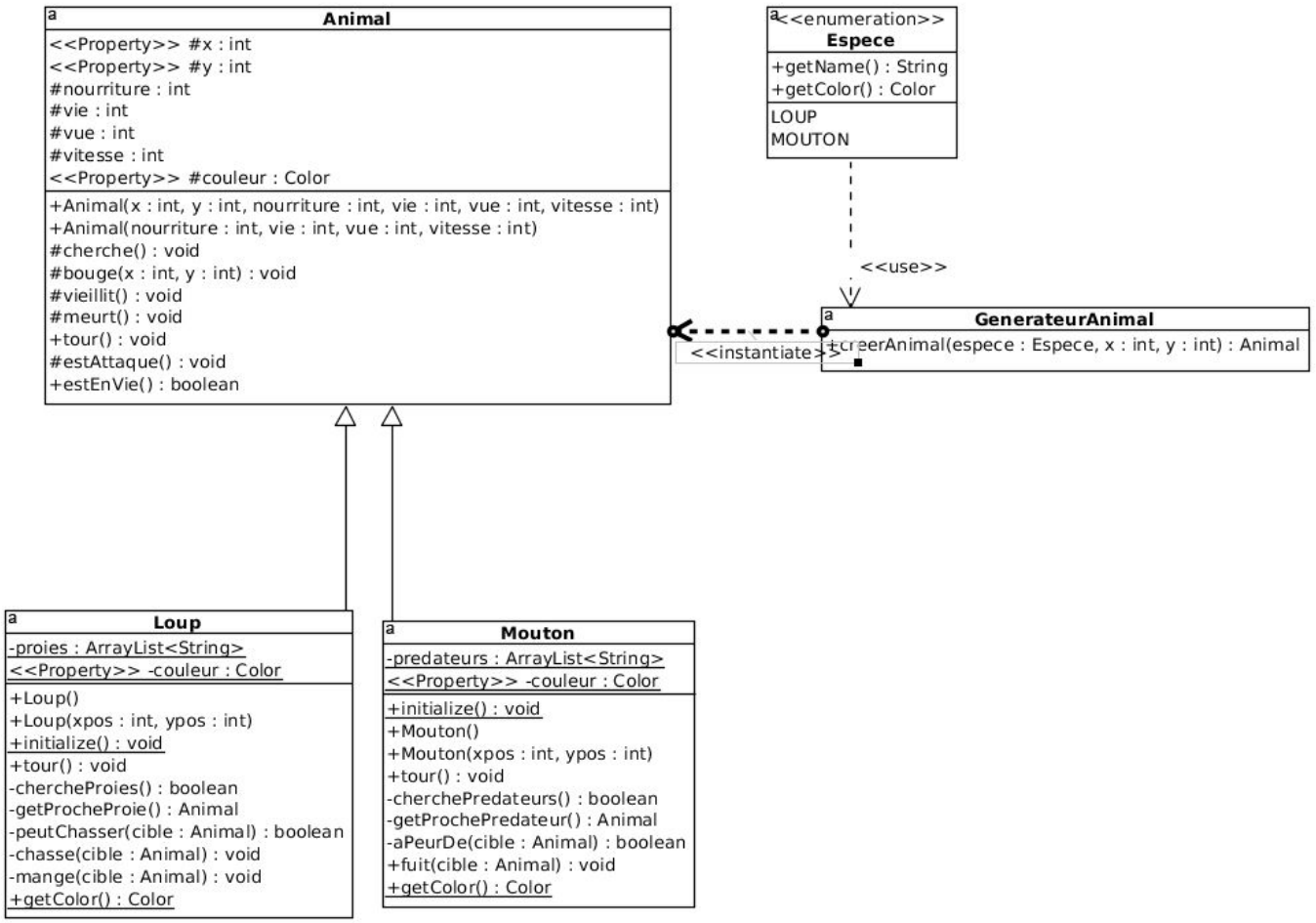
5.3.4 Classe PanneauContrôle

Ce panneau permet de contrôler le déroulement de la simulation.

Ses attributs:

- btnPause: permet de mettre en pause la simulation,
- btnLecture: permet de lancer une première fois puis de relancer après chaque pause la simulation,
- setVitesse: un slider permettant d'accélérer ou de ralentir la vitesse d'évolution de la simulation.

5.4 Package Life



5.4.1 Classe Animal

Cette classe est une abstraction des différents types d'unités disponibles. Elle implémente des versions par défaut de leurs comportements, qui sont éventuellement redéfinis dans les classes qui en héritent, selon un Design Pattern Specialisation.

Ses attributs:

- x de type int: position en x de l'animal considéré,
- y de type int; position en y de l'animal considéré,
- nourriture: apport nutritif de l'animal lorsqu'il est mangé,
- vie: durée de vie de l'animal, peut évoluer en fonction de son alimentation,
- vue: distance à laquelle l'animal peut détecter des proies ou des prédateurs,
- vitesse: vitesse de déplacement de l'animal, c'est à dire distance maximale qu'il peut parcourir en un tour,
- couleur: couleur servant à représenter l'animal dans la fenêtre de l'application.

Ses méthodes:

- cherche: permet à un animal n'ayant repéré ni proie ni prédateur de se déplacer aléatoirement,
- bouge: permet à un animal de se déplacer vers une case donnée,
- vieillit: permet à un animal qui finit son tour de sentir le poids des années passer, en diminuant sa durée de vie,
- meurt: permet à un animal de disparaître lorsqu'il meurt,
- estTue: méthode appelée par un animal qui attaquerait celui-ci, afin de lui signaler qu'il doit mourir,

5.4.2 Classe Mouton

Cette classe implémente un comportement de proie pure, qui ne peut que vieillir ou se faire manger. Elle redéfinit un certain nombre de méthodes de la classe Animal et y ajoute les attributs et méthodes suivants:

Attributs:

- predateurs: liste d'animaux dont il faut avoir peur et qu'il faut fuir pour essayer de rester en vie

Méthode:

- cherchePredateurs: permet à l'unité de vérifier si un prédateur se trouve dans son champ de vision,
- prochePredateur: permet de déterminer quel est le prédateur le plus proche dans le cas où la méthode précédente en a trouvé au moins un à portée de vue,
- aPeurDe: permet de vérifier que l'Animal ciblé représente un danger,
- fuit: déplacement visant à s'éloigner le plus possible du plus proche prédateur repéré, quitte à s'approcher d'un autre.

5.4.3 Classe Loup

Cette classe implémente un comportement de prédateur pur, qui ne peut vieillir ou chasser. Elle redéfinit un certain nombre de méthodes de la classe Animal et y ajoute les attributs et méthodes suivants:

Attributs:

- proies: liste d'animaux qu'il faut chasser pour se nourrir,

Méthode:

- chercheProies: permet à l'unité de vérifier si une proie se trouve dans son champ de vision,
- prochaineProie: permet de déterminer quelle est la proie la plus proche dans le cas où la méthode précédente en a trouvé au moins une à portée de vue,
- peutChasser: permet de vérifier que l'Animal ciblé représente un repas potentiel,
- chasse: déplacement visant à sauter sur la cible afin de l'attaquer, si celle-ci est à portée de déplacement.

5.4.4 Classe Espece

Cette énumération permet simplement de lister les différents types d'unités existants, et de créer à l'aide d'une fabrique les unités du type demandé durant la phase d'initialisation.

5.4.5 Classe GenerateurAnimal

Cette classe permet d'implémenter une Factory chargée de créer les unités du bon type lors de la phase d'initialisation par l'utilisateur.

5.4 Package Erreur

5.4.1 Classe CaseOccupeeException

Cette exception permet de gérer le cas où l'utilisateur essayerait de placer une unité sur une case déjà occupée par une autre unité. L'unité n'est alors pas ajoutée, et tout continue comme si l'utilisateur n'avait pas effectué cette action.

5.4.2 Classe FenetreErreur

Cette fenêtre permet tout simplement de prévenir l'utilisateur qu'il a fait une erreur. La seule erreur implémentée n'étant pas bloquant, la fenêtre n'a réellement qu'un but informatif et peut être fermée sans nuire au déroulement de la simulation.

Lexique

Attribut : Un attribut est une information caractéristique liée à un objet.

Classe : Une classe est le "modèle" d'un objet. Elle fournit les attributs et les méthodes des objets qui seront calqués sur ladite classe.

IHM : Interface Homme Machine, caractérise une interface permettant à la machine d'informer l'utilisateur et à l'utilisateur de communiquer des instructions en retour. Dans le cas présent, il s'agit d'une fenêtre graphique proposant diverses interactions.

Objet : Un objet est une instanciation d'une classe. Il possède donc les attributs et méthodes définies par sa classe et peut évoluer indépendamment d'autres objets de la même classe.

Méthode : Une méthode est un traitement spécifique que peut effectuer un objet.

Package : Un package est un regroupement de classes visant à présenter une forte cohérence interne et à minimiser les dépendances externes à d'autres packages.

Unité : Une unité est un élément évolutif de la simulation, représentant un animal. Elle évolue en fonction des autres unités environnantes.