Benjamin Betti

Projet POO:
Modèle Proies-Prédateurs

Cahier de Conception

Historique des révisions

Date	Description et justification de la modification	Auteur	Pages/ chapitre	Edition/ Revision
10/02/2018	**Création**	**Betti**		**0.0**

Table des matières

1. Introduction

Les spécifications présentées dans ce document visent à décrire comment sera réalisée l'application demandée par le client, à savoir un *automate cellulaire* devant simuler un modèle d'évolution de populations suivant une relation proie-prédateur.

Les fonctions principales de cette application seront donc la création des proies et prédateurs sur un terrain, ainsi que la simulation de l'évolution de leurs populations respectives 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.

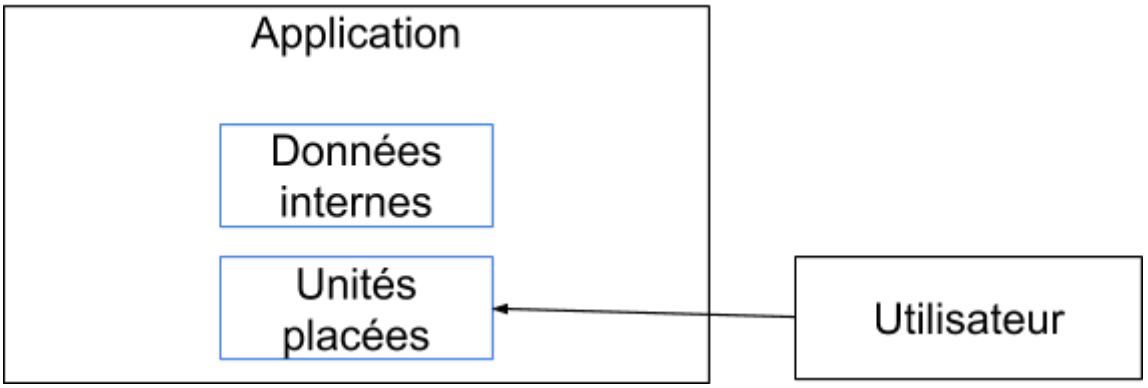
Le terrain est de taille fixe, carré et boucle sur lui-même (ainsi une unité sortant d'un côté se retrouve visuellement de l'autre côté).

2. Exigences opérationnelles

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

Les contraintes de performance sont quasi inexistantes: l'application tournera 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 pour laisser le temps à l'utilisateur d'observer le déroulement.

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



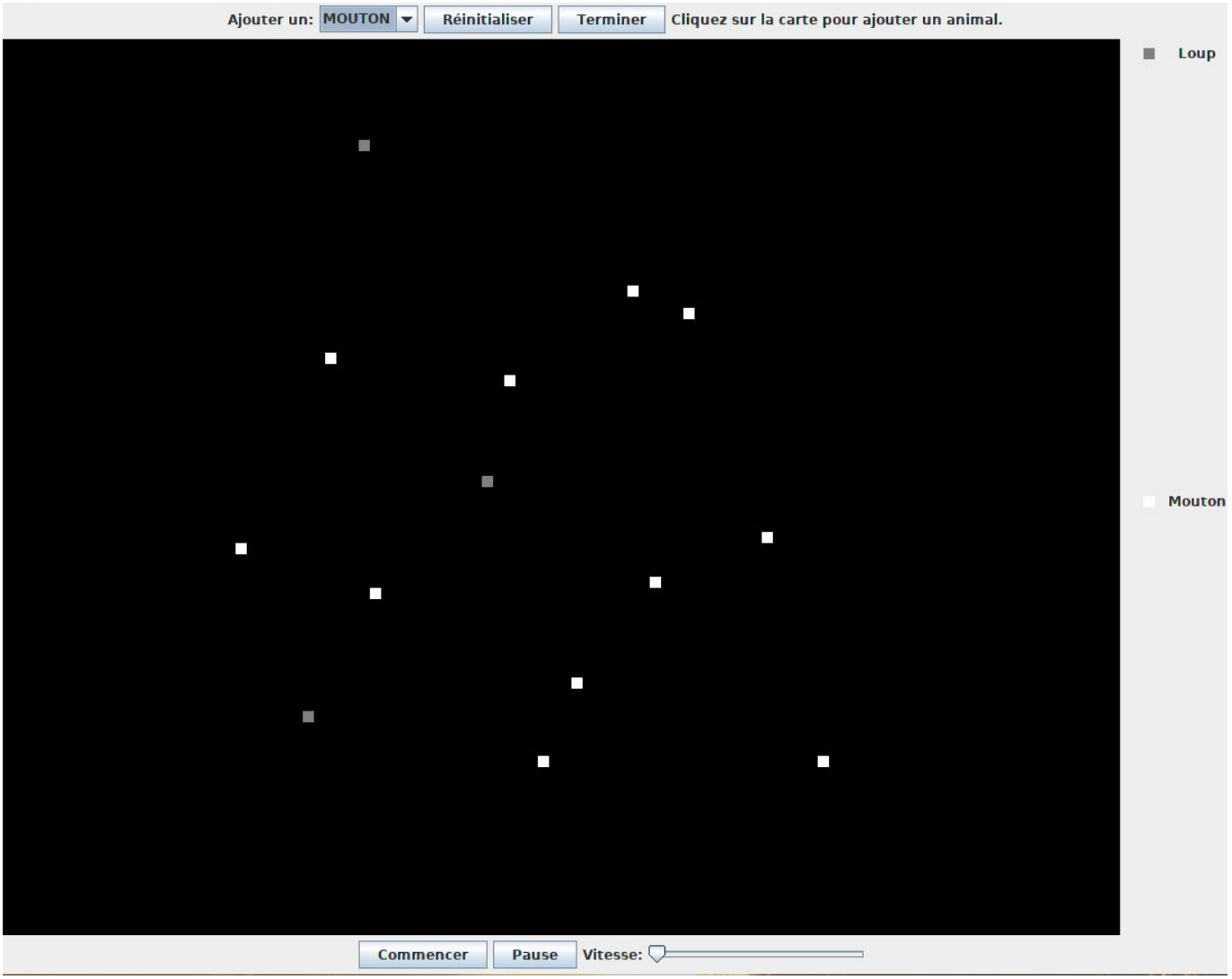
La *simulation* en elle-même sera autonome, mais l'on gardera la possibilité de la mettre en pause ou de la reprendre ou de la quitter à tout moment.

3. Interfaces

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

L'intégralité des donnés générées par la simulation étant transitoire, il n'y aura pas besoin de gérer des connexions ou des fichiers d'entrée/sortie.

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

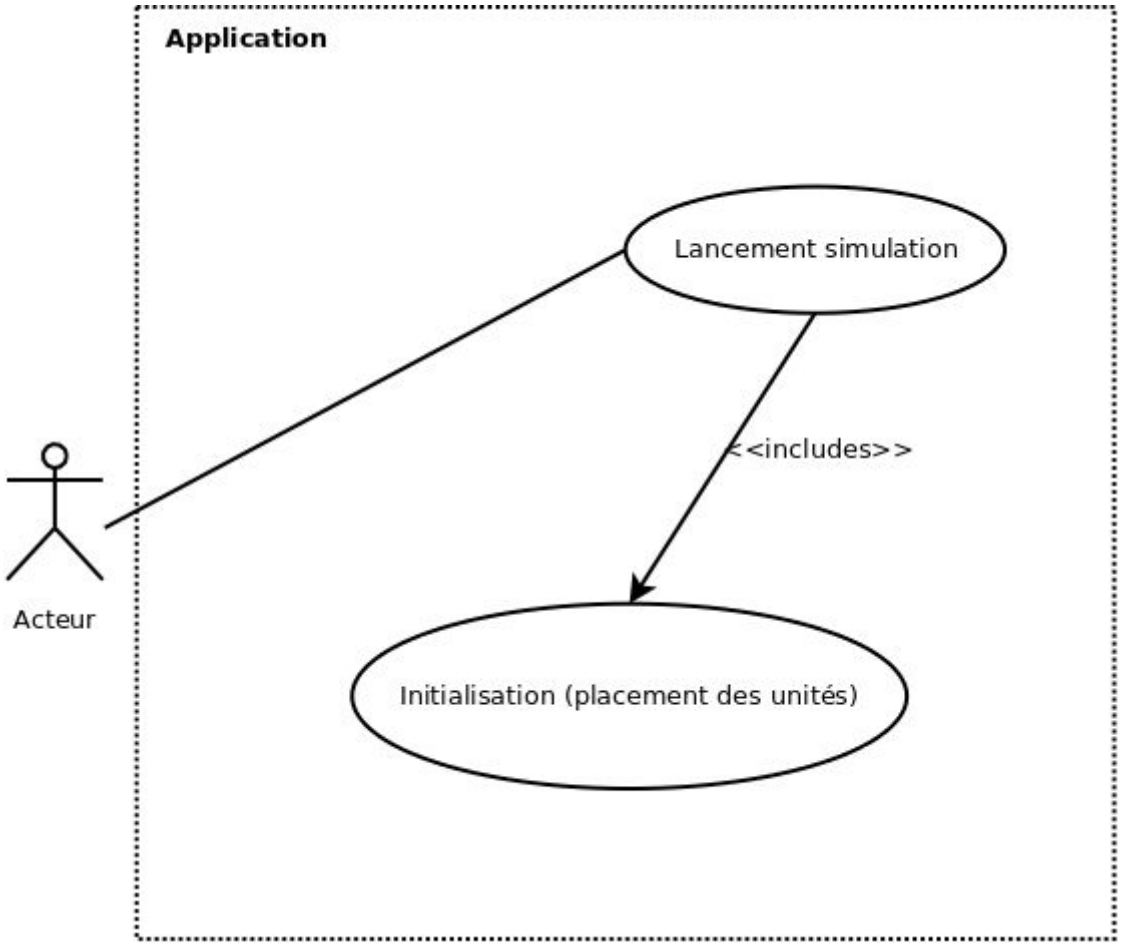
4. Conception

L'application sera réalisée en langage *Java*, pour son programmation orientée objet et sa grande portabilité.

L' interface homme-machine sera 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 et 2 comportements (proies et prédateurs), se déplaçant dans un monde fermé. La simulation se termine dès lors que toutes les unités sont décédées.

Le seul cas d'utilisation est le lancement d'une simulation, précédé par son initialisation.



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, en choisissant dans un menu déroulant l'espèce à

placer, puis en cliquant à l'endroit désiré sur la fenêtre pour placer l'unité. Un clic sur le bouton "Commencer" permettra ensuite de lancer la phase de simulation.

4.2 Éléments d'une simulation

Une simulation fera 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

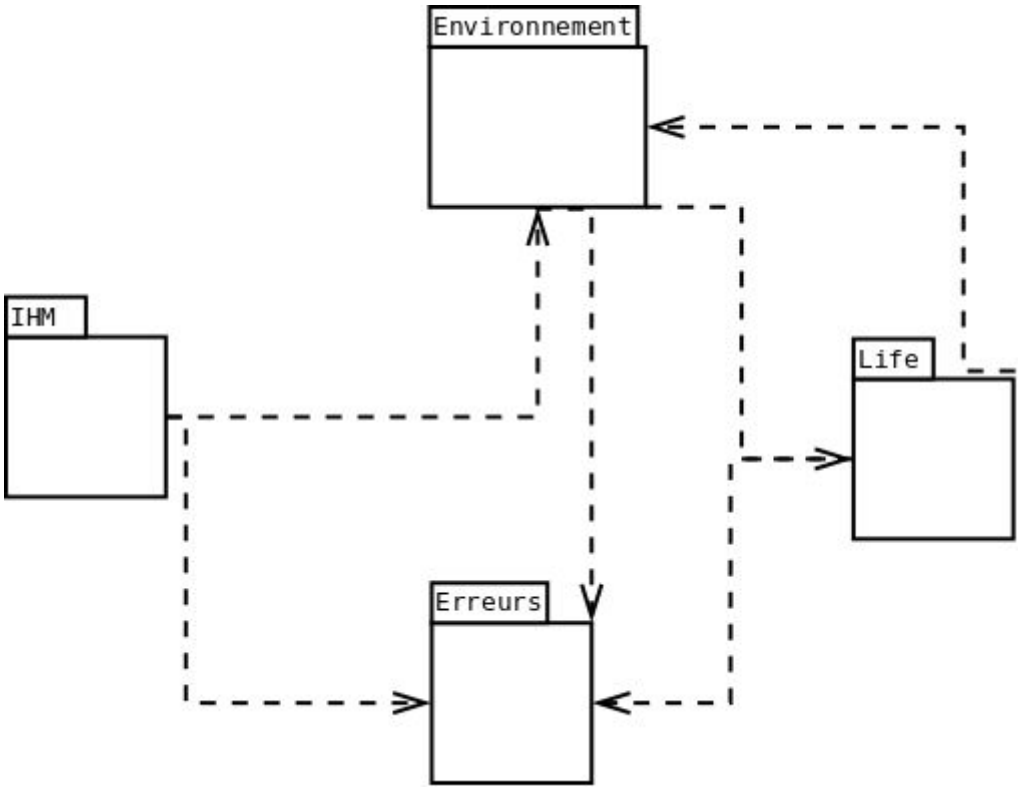
La simulation se déroulera en tour par tour, de manière autonome. Elle s'achèvera au clic sur le bouton "Terminer" ou lorsque toutes les unités seront mortes. Les unités pourront interagir de différentes manières. Un tour d'une unité sera décomposé en la suite d'étapes suivante:

- 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 vers une zone inconnue

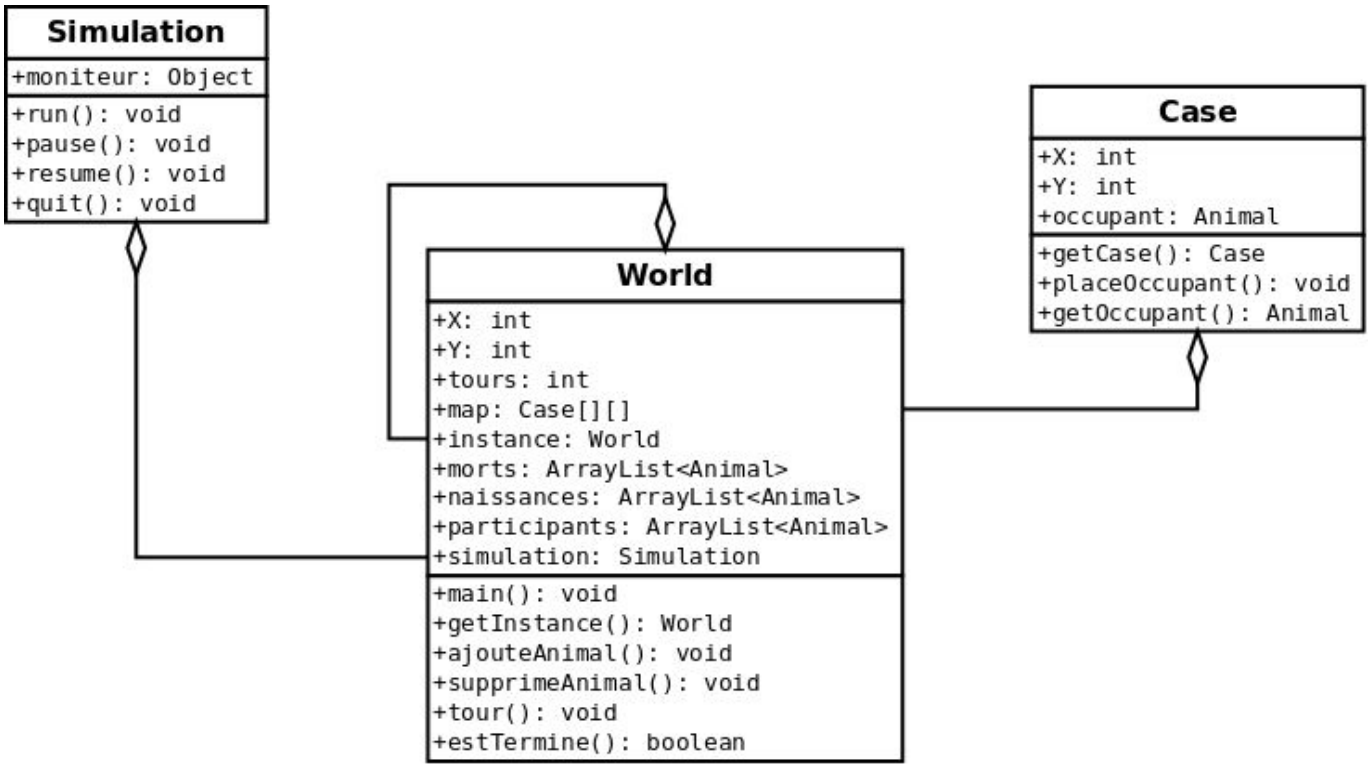
4.4 Découpage logique

Il est prévu de suivre 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.



4.4.1 Package Environnement



4.4.1.1 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,
- 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,
- participants, liste de type Animal: liste les unités présentes et vivantes, qui doivent donc jouer le prochain tour,

Ses méthodes:

- main: point d'entrée du programme,

- un constructeur permettant de créer un monde de taille définie,
- ajouteAnimal: méthode permettant d'insérer un Animal sur la carte,
- supprimeAnimal: méthode supprimant un animal mort de sa case
- 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,

4.4.1.2 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:

- getCase: méthode pour obtenir une case précise par ses coordonnées,
- getOccupant: méthode permettant d'accéder à l'occupant de la case,
- placeOccupant: méthode permettant d'ajouter un occupant à la case.

4.4.1.3 Classe Simulation

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

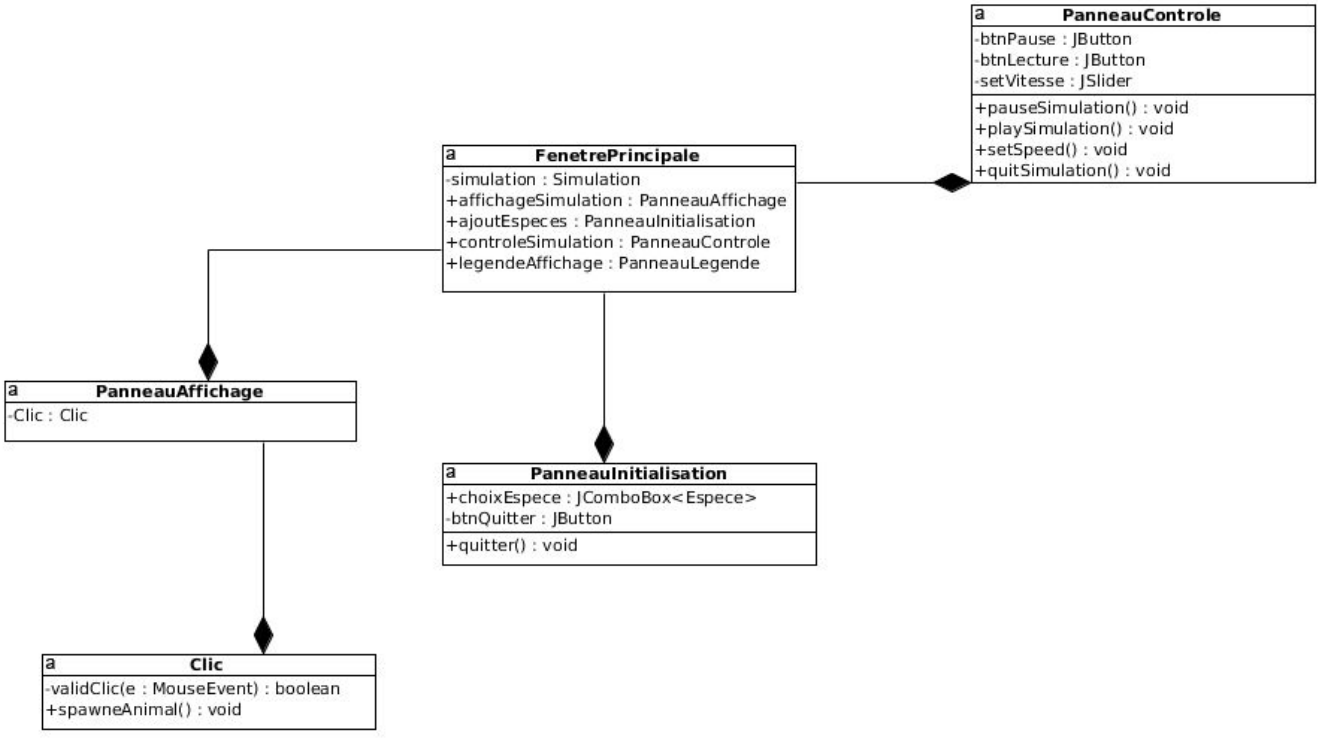
Attributs:

- moniteur: objet servant de moniteur pour pauser/reprendre la simulation depuis les contrôles de la fenêtre de l'application,

Méthodes:

- run: méthode permettant de lancer les actions à effectuer durant la simulation, point d'entrée du thread,
- resume: 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,

4.4.2 Package IHM



4.4.2.1 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.

4.4.2.2 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,
- btnQuitter: un boutton permettant de quitter l'application à tout moment,

4.4.2.3 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 durant la phase d'initialisation.

4.4.2.4 Classe PanneauContrôle

Ce panneau permet de contrôler le déroulement de la simulation.

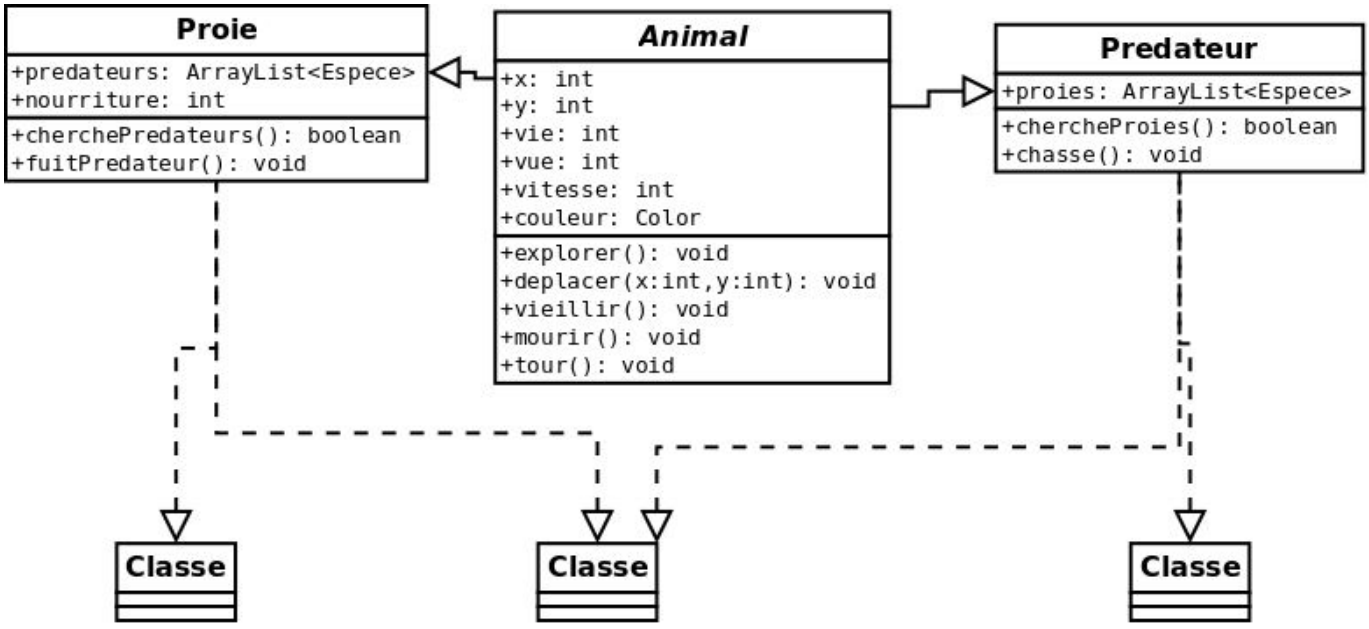
Ses attributs:

- 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.

Ses méthodes:

- quitSimulation: met fin à la simulation à la demande de l'utilisateur,
- playSimulation et pauseSimulation permettent de contrôler le déroulement de la simulation,

4.4.3 Package Life



4.4.3.1 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,
- 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:

- explorer: permet à un animal n'ayant repéré ni proie ni prédateur de se déplacer aléatoirement,
- deplacer: 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,
- tour: représente l'enchaînement des actions qui survient durant un tour.

4.4.3.2 Interface Proie

Cette interface présente un comportement de proie pure, qui ne peut que fuir 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,
- fuitPredateur: déplacement visant à s'éloigner le plus possible du plus proche prédateur repéré.

4.4.3.3 Interface Prédateur

Cette interface présente un comportement de prédateur pur, qui ne peut que 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,
- chasse: déplacement visant à sauter sur la cible afin de l'attaquer, si celle-ci est à portée de déplacement.

En plus de ces bases existeront plusieurs classes héritant de Animal et implémentant Proie, Prédateur, ou les deux, selon un pattern design de type Template Method.

4.4.4 Package Erreurs

4.4.4.1 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.

4.4.4.2 Classe FenetreErreur

Cette fenêtre permet tout simplement de prévenir l'utilisateur qu'il a fait une erreur. La seule erreur implémentée n'étant pas bloquante, 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.