Benjamin Betti

Projet POO:
Modèle Proies-Prédateurs

Dossier de Tests

Historique des révisions

Date	Description et justification de la modification	Auteur	Pages/ chapitre	Edition/ Revision
14/02/2018	**Création**	**Betti**		**0.0**

Table des matières

1. Introduction

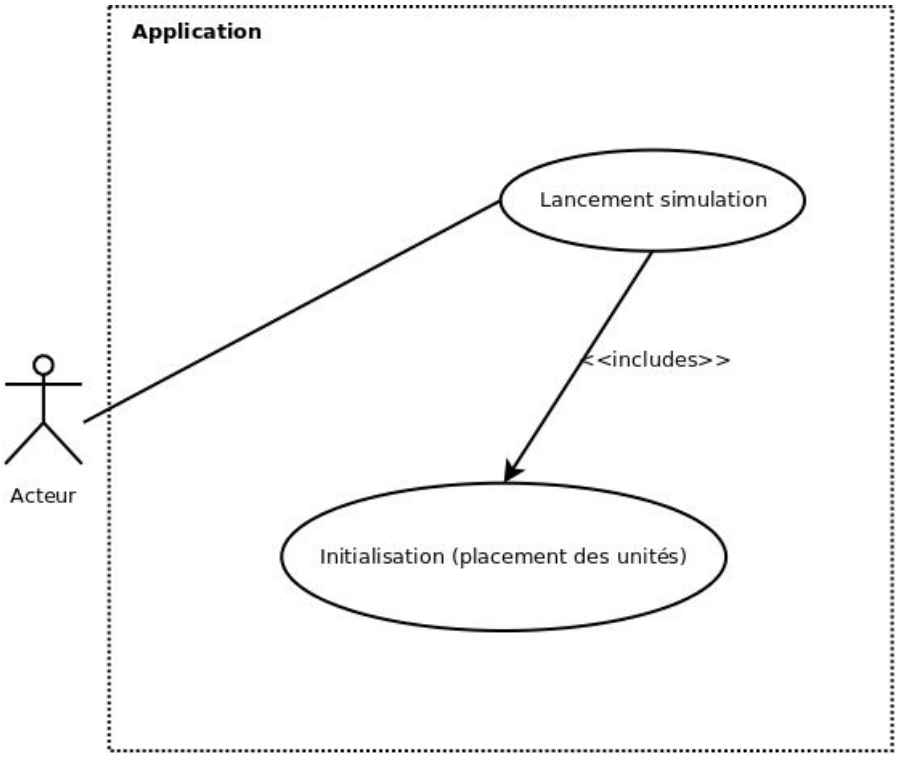
Ce document a pour but de présenter les méthodologies de tests employées lors de la création de l'application, ainsi que des fiches reprenant et résumant les principaux tests mis en oeuvre ainsi que les résultats associés.

1.1 Rapide présentation de l'application

L'application est une simulation simpliste de modèle proies/prédateurs écrite en Java, selon le paradigme de Programmation Orientée Objet. Elle dispose d'une IHM simple, qui permet à la fois l'initialisation, le contrôle, et le suivi du déroulement de la simulation. Les proies sont simplement représentées par des Moutons, et les prédateurs par des Loups.

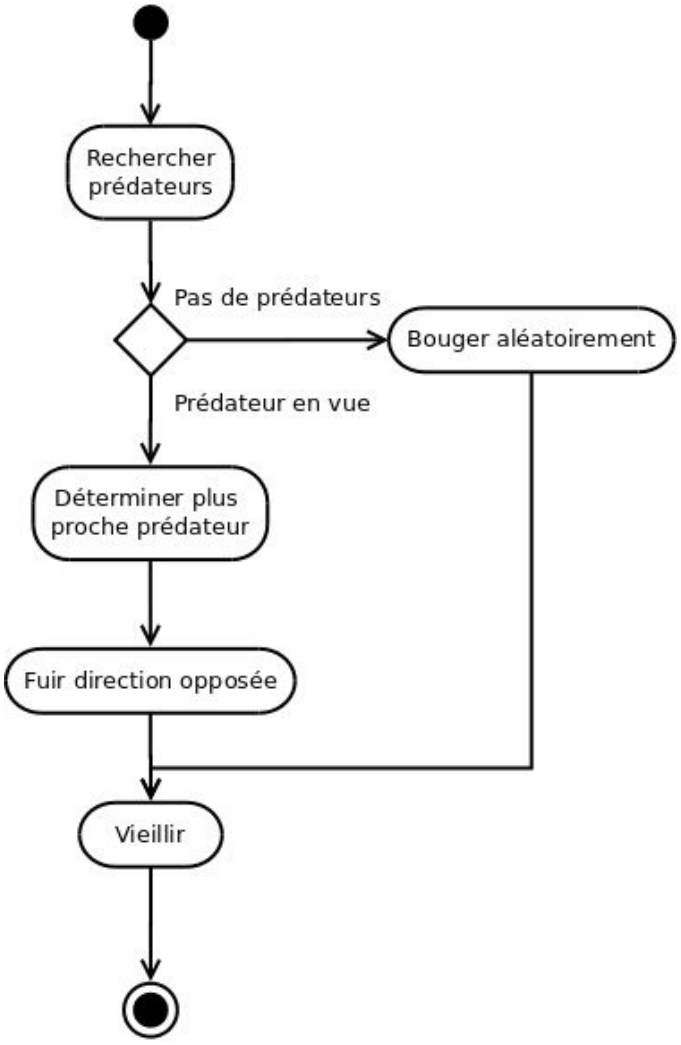
L'initialisation se déroule tout simplement en cliquant sur le monde affiché pour y placer à un endroit donné l'animal sélectionné. Ce monde est représenté par un ensemble de *cases,* ne pouvant être occupées que par une seule unité à la fois. Durant la simulation, ces unités seront amenées à détecter ce qui se trouve à proximité d'elles, à se déplacer, et à mourir. La simulation peut à tout moment être accélérée, ralentie, mise en pause ou quittée. On peut en outre ajouter à tout moment des unités sur la carte.

La seule faute que peut commettre l'utilisateur est de tenter de placer une unité sur une case déjà occupée, auquel cas l'unité n'est pas placée et l'utilisateur voit s'afficher une fenêtre d'erreur n'interrompant pas la simulation.
Les cas d'utilisations sont les suivants:

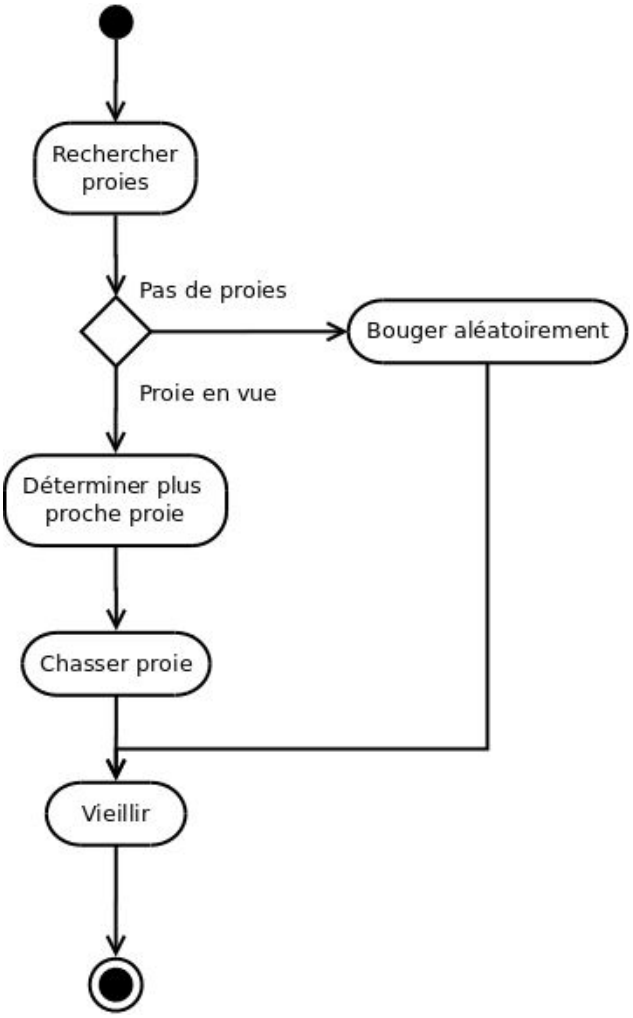


La simulation se déroule au tour par tour, où chaque tour de la simulation demande à chacun des animaux en vie durant ce tour d'effectuer eux-même ce qui correspond à leur définition du tour. Selon leur classe (proie, prédateur, proie & prédateur), la définition du tour peut varier.

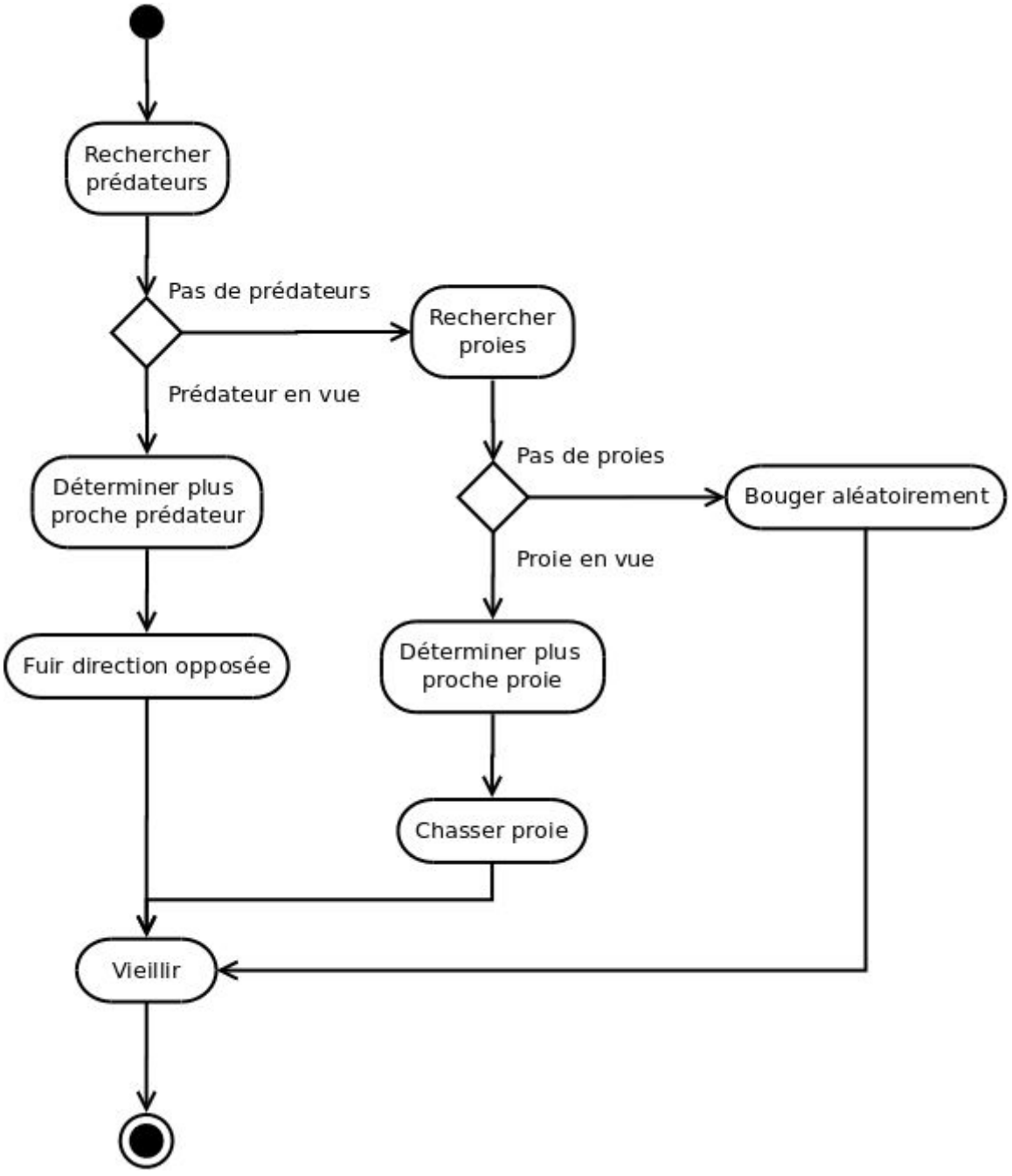
Cas d'une proie pure:



Cas d'un prédateur pur:



Cas d'un animal à la fois proie et prédateur:



1.1.1 Exigences opérationnelles

L'utilisateur se verra 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).

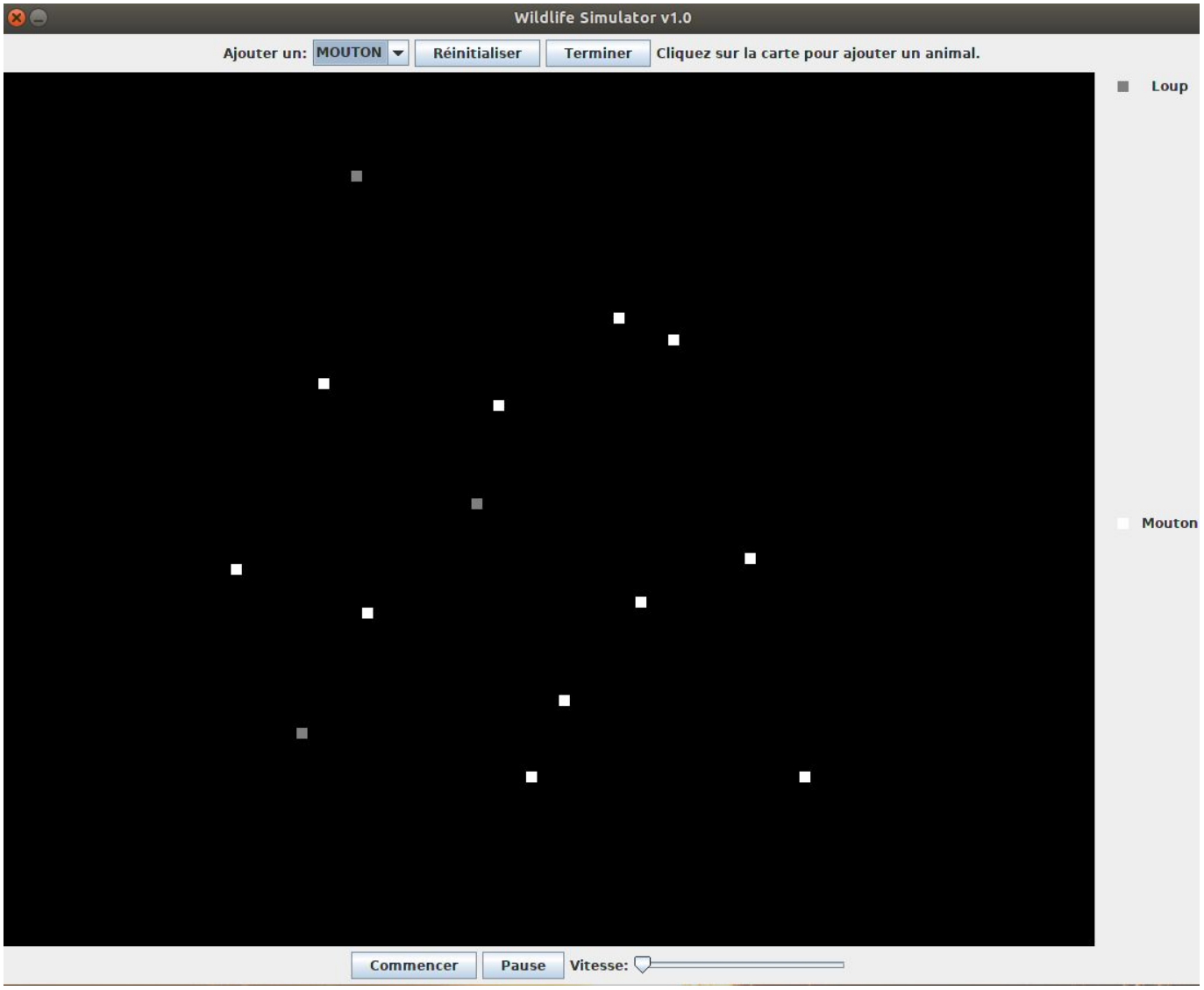
Il n'y a pas de contraintes de sécurité, ni d'intégrité.

1.1.2 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.

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

2. Organisation des tests

Les tests décrits ultérieurement seront réalisés le plus tôt possible, durant le développement, afin d'éviter de développer en partant sur de mauvaises bases, et afin de faciliter la détection et la correction rapide des erreurs.

Au vu de la faible complexité de l'application, les tests ne sont pas automatisés, et les vérifications sont la plupart du temps faites soit à l'aide du mode débug de l'IDE, soit par des tests de valeurs et des impressions dans la sortie standard permettant de vérifier l'état de l'application à des moments choisis.

2.1 Environnement

Il convient tout d'abord de définir le matériel sur lequel l'application sera développée et donc testée:

Machine:	Ordinateur portable Gigabyte P15F-V5 15,6"	Ordinateur portable MSI CX-61 FR 15,6"
Processeur:	Intel Core i7-6700HQ - 4 Cores/8 Threads - 2,6 Ghz	Intel Core i3-4100M - 2 Cores/2 Threads - 2,5 Ghz
RAM:	8 Go DDR3	4 Go DDR3
Disque Dur:	128 Go SSD + 1To HDD	1 To HDD
Circuit Graphique:	NVIDIA Geforce GTX 950M	NVIDIA Geforce GTX 940M
Écran:	1920x1080, 60Hz, 15,6"	1366x768, 60Hz, 15,6"
OS:	Ubuntu 17.10 64 bits Gnome DE	Ubuntu 17.10 64 bits Gnome DE
Java:	OpenJDK 9 64 bits	OpenJDK 9 64 bits
Ide:	Eclipse 4.7.2	Eclipse 4.7.2

Aucune donnée autre que l'exécutable et ses dépendances n'est stockée de manière durable. De même, aucune communication réseau n'a lieu.

3. Fiches de Tests

3.1 Tests d'ajout d'unités

Nom de l'application:	Wildlife Simulator
Nature du test:	Nominal
Objectif du test:	Vérifier le fonctionnement des méthodes d'ajout d'unités à la simulation ou à un emplacement de la carte en phase d'initialisation et de simulation.
Contexte:	IHM fonctionnelle, application lancée, simulation non encore déclenchée.
Principe du test:	On vérifie dans un premier temps par un système de logging que les EventListener au clic sur la carte sont bien déclenchés, puis on liste dans un second temps la totalité des unités placées sur la carte.

Action:	**Résultat:**
Choix d'une unité via le menu déroulant, affichage de la classe sélectionnée au clic sur la carte.	Fonctionnel. Les EventListeners sont bel et bien déclenchés au clic, et la variable contenant l'espèce sélectionnée possède bien la bonne valeur à chaque fois.
Ajout d'une ou plusieurs unités sur des cases distinctes, récupération de la liste des unités présentes.	Fonctionnel. Toutes les unités placées sont bien enregistrées, et leurs attributs sont corrects. En outre, on constate bel et bien l'apparition immédiate du carré de couleur symbolisant l'animal ajouté au bon endroit sur la carte.
Ajout de plusieurs unités, de la même classe puis de classes différentes, sur la même case.	Fonctionnel. Le message d'erreur s'affiche correctement, l'unité déjà présente sur la case y reste, et la nouvelle n'est tout simplement pas ajoutée à la simulation. L'affichage ne change pas.
Sous fausse simulation (i.e. tours factices consistant à afficher un texte à chaque tour et à déplacer et faire mourir les animaux de manière aléatoire), ajout d'animaux sur des cases occupées par un animal vivant ou mort, et sur des cases libres.	Fonctionnel. Les animaux sont bien mis en attente puis placés dès le début du tour d'après. Placer un animal sur un cadavre fonctionne, car seuls les animaux vivants sont considérés comme des occupants. L'affichage se met à jour correctement. Placer un animal sur une case occupée provoque la même réaction qu'au tour précédent.
Sous fausse simulation, on fait se déplacer et mourir les unités de façon à essayer d'en placer deux sur la même case.	Fonctionnel. Les animaux tentant de se déplacer sur une case occupée n'y arrivent pas et restent sur place.

3.2 Tests de mort d'unités

Nom de l'application:	Wildlife Simulator
Nature du test:	Nominal
Objectif du test:	Vérifier le fonctionnement des méthodes de vieillissement et suppression des unités.
Contexte:	IHM fonctionnelle, application lancée, simulation en cours.
Principe du test:	On vérifie dans un premier temps par un système de logging que le vieillissement et la mise en attente de suppression des unités fonctionne correctement. On affiche ensuite à chaque tour les unités en vie.

Action:	**Résultat:**
Placement d'une unité seule sur la carte, vouée à mourir de vieillesse ou de faim. Placement de plusieurs unités de la même classe dans le même but.	Fonctionnel. L'unité voit bien son espérance de vie diminuer à chaque tour, elle est bien mise en attente lorsqu'elle meurt, ce qui l'empêche de jouer son tour, et disparaît au début du tour suivant.
Ajout d'une proie et d'un prédateur dont le tour consiste uniquement en l'attaque de cette proie, dans des ordres différents afin de tester des tours différents.	Fonctionnel. Si l'unité est en vie quand arrive son tour de jouer, elle joue son tour, si elle est tuée elle est mise en attente, ne joue pas son tour, et est supprimée de la simulation au début du tour suivant. On constate par ailleurs que le prédateur récupère au passage la bonne quantité de nourriture et/ou de vie.

3.3 Tests de détection

Nom de l'application:	Wildlife Simulator
Nature du test:	Nominal
Objectif du test:	Vérifier le fonctionnement des méthodes de détection des unités à portée de vue et des actions à prendre.
Contexte:	IHM fonctionnelle, application lancée, simulation en cours.
Principe du test:	On vérifie par un système de logging que la détection examine les bonnes

	cases et détecte toutes les unités en vie qui y sont présentes. Ensuite, on regarde quelle méthode elles appellent dans la suite du tour.

Action:	**Résultat:**
Placement d'une unité seule, puis de plusieurs unités de même classe et de classes différentes en dehors de leur champ de visions respectifs.	Fonctionnel. Les unités sont toutes présentes mais ne se détectent pas les unes les autres.
Ajout de plusieurs unités de même classe à portée de vue les unes des autres.	Fonctionnel. Les unités ne détectent que leurs semblables situés dans leur champ de vision, et reconnaissent bien que ce sont leurs semblables.
Ajout d'une proie et d'un prédateur en modifiant les valeurs de leur champ de vision afin que la détection ne soit pas réciproque.	Fonctionnel. Une unité ayant un plus grand champ de vision détectera bien d'autres unités sans se faire voir si elle est elle-même en dehors du champ de vision de sa cible.
Ajout de plusieurs proies et prédateurs afin de vérifier que les détection du plus proche ennemi fonctionnent correctement.	Fonctionnel pour les Loups. L'oubli de la vérification d'une condition dans la méthode "prochePrédateur" des Moutons fait que si un prédateur est dans leur champ visuel, alors les moutons détectent le plus proche prédateur comme étant…. eux-mêmes. De fait ils ne peuvent s'enfuir à cause d'une exception arithmétique et restent sur place, terrifiés par ce qui pourrait être assimilé à leur ombre. Ajout de la condition nécessaire au bon fonctionnement de cette méthode dans la classe Mouton.
Ajout de plusieurs proies et prédateurs afin de vérifier que les bonnes actions sont effectués après détection..	Fonctionnel. Les Moutons s'enfuient bien le plus vite possible dans la direction opposée au plus proche prédateur, et les Loups sautent dès que possible sur la proie la plus proche.

3.4 Tests de contrôle de la simulation

Nom de l'application:	Wildlife Simulator
Nature du test:	Nominal
Objectif du test:	Vérifier le fonctionnement des méthodes de contrôle d'exécution du thread jouant la simulation.

Contexte:	IHM fonctionnelle, application lancée, simulation en cours.
Principe du test:	On essaye en phase d'initialisation, de simulation ou de pause de pauser, reprendre, ou terminer la simulation, le tout avec des tours factices consistant à faire aller les unités toujours tout droit (rappel: la carte est "fermée" et sortir d'un côté revient à réapparaître de l'autre).

Action:	**Résultat:**
Appui sur le bouton "Terminer" en initialisation, en pause, en simulation.	Fonctionnel. L'application se termine correctement, ainsi que le Thread si il avait déjà débuté, comme on a pu le constater en observant la liste des processus en cours.
Appui sur le bouton "Commencer/Reprendre" de manière répétée lorsque la simulation est en pause, terminée ou en cours.	Fonctionnel. Si la simulation n'est pas lancée, elle se lance. Si elle est en pause, elle reprend. Si elle est en cours ou terminée, rien ne change.
Appui sur le bouton "Pause' de manière répétée avant le début de la simulation, lorsqu'elle est en cours, lorsqu'elle est terminée ou en pause.	Fonctionnel. Si la simulation est en cours, elle s'arrête à la fin du tour actuel. Sinon, rien ne se passe.
Modification du réglage de vitesse avant et après le lancement et la fin de la simulation, quand elle est en cours et quand elle est en pause.	Fonctionnel. La vitesse change dès le tour suivant ou la reprise/le lancement de la simulation. Si la simulation est terminée, rien ne se passe. Une vérification par logging montre que la valeur est correctement lue.
Lancement d'une simulation ne comportant qu'un seul prédateur ou une seule proie ou les deux mais incapables de se déplacer et d'attaquer afin de vérifier que la simulation termine correctement.	Fonctionnel. Un système de logging permet de s'assurer que les méthodes visant à terminer le thread et l'application sont bien appelées, et l'observation des processus actifs vérifie leur bon fonctionnement.