Rabbit Simulation Rapport

Jean-Baptiste LARGERON, Karam ELNASORY

June 19, 2025

Sommaire

- **Introduction (p.2)**
- **Premier Essai: Fibonacci (p.2)**
- **Choix de langage (p.3)**
- **choix de codage(p.3)**
- **explication du code (p.4)**
- **Compilation et Exécution du Code (p.6)**
- **Conclusion (p.6)**

Introduction

Nous cherchons via cette simulation à étudier l'évolution des lapins dans le temps selon les caractéristiques suivantes :

1. un lapin devient mature entre 5 et 8 mois
2. il y a approximativement 50% de lapins mâles et 50 % de lapins femelles
3. une femelle donne 3 à 9 portées par an de 3 à 6 petits
4. les adultes ont un taux de survie annuel de 60%
5. les petits ont un taux de survie annuel de 35%
6. à l'age de 10 ans (compris) les adultes perdent 10% de taux de survie par an (à 15 ans ils ont 100%)

Nous cherchons à simuler une 20 aine d'année. Pour cela, nous avons fait des choix et des implémentations, et vous trouverez dans les paragraphes ci-joint le descriptif et les idées de nos programmes et fonctions.

Premier essai: la suite de fibonacci

Dans la première partie de nos travaux, nous avions la consigne d'écrire un code simulant la suite de Fibonacci en langage C. Nous avons alors crée une structure simulant le nombre de lapins adultes/enfants que nous avons. Puis nous les plongeons dans le temps afin de comprendre l'évolution que cela peut avoir. Cette simulation démontre la vitesse exponentielle à laquelle les lapins peuvent se reproduire dans la théorie.

1 Début de la véritable simulation

choix de langage

Nous avons choisi d'utiliser Java afin d'implémenter notre simulation. Le langage objet se prêtait bien à l'idée d'utiliser des objets "lapins" indépendants les uns des autres.
Cependant nous avons rencontré quelques difficultés dû au langage que nous avons choisi. En effet, la Virtual Machine Java consomme beaucoup de mémoire dans son exécution, ainsi notre programme se retrouvait à nous rendre des erreurs de type "out of memory".

Cependant les outils proposés par Java sont aussi un fort avantage, notamment l'implémentation du générateur pseudo-aléatoire Mersenne-Twister, héritant de la class Random, nous permettant de le manipuler de la même manière que la class Random de Java.

De plus, l'outil de la javadoc permet au survol de chaque fonction/méthodes de voir le bloc de commentaire qui lui est associé si vous ouvrez le code avec un IDE ou un editeur de texte avec les modules associés.

2 Choix de Codage

Nous avons d'abord implémenter un code que nous avions réfléchit au préalable. Nous en avons fait un UML, dans l'optique d'avoir une ébauche de ce que l'on souhaitait faire:
Ainsi nous avons commencer le projet.

2.1 Première implémentation (dossier RabbitSim)

Cette implémentation était prévue premièrement puisqu'elle offre des règle de simulation complexe ainsi nous permettant de simuler un système stochastique complexe ou un tout petit changement des paramètres initiaux nous donnera un état final très différente, même avec beaucoup d'optimisation (algorithme de parcours, utilisation de List chaîné, ...) malheureusement telle simulation n'est pas possible avec les ressources dont nous disposons pour satisfaire les critères du projet (simuler 15 ans), mais reste intéressante en tant que système stochastique complexe, la seul manière de la rendre faisable sera de borner le nombre de lapins et de l'espace qu'ils peuvent occupé.

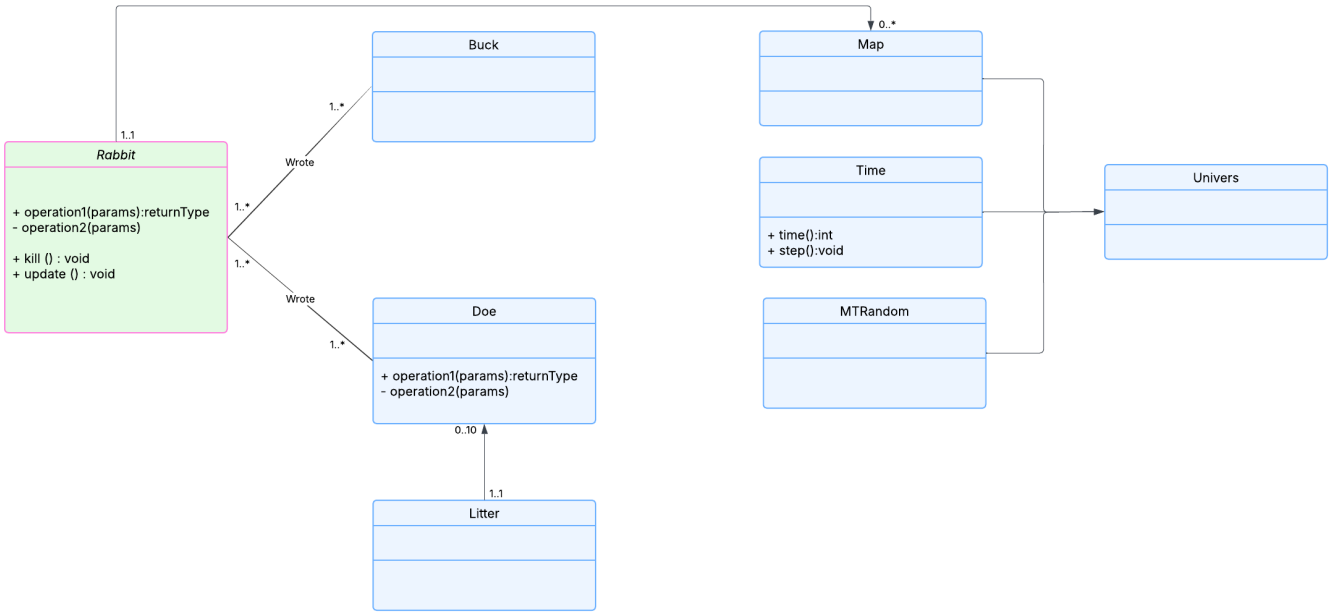

Figure 1: Ebauche UML

2.2 Seconde implémentation (dossier RabbitSimTest)

La seconde implémentation s'est trouvé obligatoire pour nous puisque nous n'avons pas réussi à exploiter la simulation du premier code au delà de 5 ans. Nous avons donc décidez de faire une implémentation différente afin d'optimiser au maximum la mémoire. Nous avons donc réduit l'objet "Rabbit" au maximum et plutôt que de le rendre totalement indépendant, nous l'avons lié à l'objet "generator" permettant de garder toujours la même instance de Mersenne-Twistter tout en gérant les changement d'états de rabbit sans tout stocké dans les cellules de notre liste lié.(objet groupOfRabbits)

Ainsi avec cette implémentation nous avons pu pousser notre simulation jusqu'à 8 ans. Nous avons eu l'idée de transformer les attributs des rabbits en utilisant un seul octet et en le modifiant afin de savoir leur état mais nous manquions de temps.
Nous avons donc fait le choix de limité le nombre maximal de lapin naissant par mois (1 000 000 tout de même).

Nous avons fait le choix dans le but d'optimiser de tuer des lapins de manière proportionnelle tous les mois afin de prendre le moins possible de mémoire.

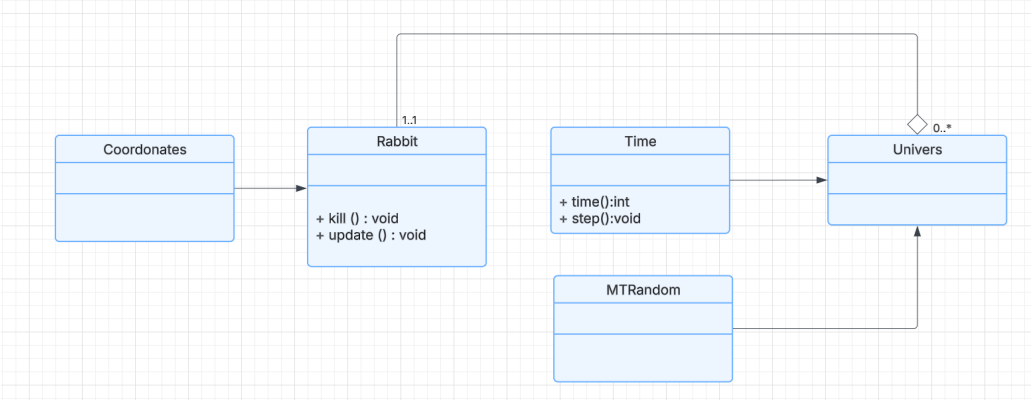

Figure 2: RabbitSim UML

3 Explication du code

3.1 SimRabbit

Aucun dépendance concernant la partie SimRabbit apart LinkedList (package java.util) pour l'implémentation des listes chaîné.

Classe	**Description**
Cezeau	Est le mode dans le quelle déroule la simulation
Coordonates	La classe qui gère les coordonnée sur un grille (virtuelle ou réelle
MTRandom	Générateur pseudo-aléatoire uniforme
Rabbit	La classe qui gère le comportement des lapins
Time	La classe responsable pour l'implémentation du temps

Table 1: Tableau des classes

Nous allons continuer de travailler sur cette implémentation, vous pourrez suivre l'évolution du projet dans le temps via la dépositoire Git (liens dans la conclusion).

3.2 SimRabbitTest

Dans cette partie, l'objet Rabbit est totalement dépendant de l'objet generator.

Le generator génère les objets Random et donc génère l'aléatoire, puis c'est lui qui détermine si l'objet Rabbit change d'état.

L'objet Rabbit quand à lui a dans ses méthodes la nécessité de prendre en paramètre un objet Random et il ne fait que demander s'il change son état. De ce fait, on garantie que la simulation soit reproductible (par l'initialisation de MT) mais aussi nous garantissons que le generateur de

nombre ne se réalise pas entre chaque simulation.

L'objet Time permet de plonger la simulation dans le temps. Celui-ci étant indépendant des objets qui l'entoure nous permet de gérer le temps indépendamment des états de generator et de nos lapins.

L'objet groupe de Rabbit est l'objet sur lequel nous avons le plus travailler dans l'optique de d'optimiser la mémoire. Nous avons fait le choix d'implémenter deux listes distinctes pour différencié les sexes des lapins. Ainsi, cela prend un octet (taille d'un booléen) de moins à stocker dans la mémoire pour chaque lapins. De plus, de par cette implémentation il est beaucoup plus simple de gérer rabbit.reproducion() puisque nous avons à connaitre le nombre de lapins d'un seul sexe car nous avons fait le choix de considérer qu'un lapin ne peux se reproduire qu'avec un autre lapin par mois.(sinon nous aurions encore nos problème de mémoire).

Concernant les chances d'avoir un certain nombre de portée par an ainsi que du nombre de petits par portée, nous avons pris la décision de "forcer" la gaussienne, afin que cela corresponde au exigence des consignes.

4 Compilation et Execution du code

Tout d'abord, assurer vous d'avoir une version de Java supérieur a Java8, car nous utilisons des fonctions lamda implémenter à partir de Java 8, si nécessaire vous pouvez télécharger les dernieres versions sur ce lien : https://www.oracle.com/java/technologies/downloads/

Pour compiler le code, il vous suffira d'ouvrir une invite de commande dans le dossier Rabbit-Simulation et de taper la commande:
$javac -d build/ src/SimRabbitTest.*java ou $javac -d build/ src/SimRabbit.*java (en fonction du code que vous voulez compiler).

Pour exécuter le code, il suffira de taper la commande:
$ java -cp build SimRabbitTest.Principal ou $ java -cp build SimRabbit.Cezeaux

5 Modularité du code

Vous pouvez modifier les attribut static final SIMULATION_COUNT et TIME_OF_A_SIM afin de changer le temps de simulation et le nombre de simulation.
ATTENTION : si vous changer le nombre de simulation il faudra mettre a jour le nombre de error_margin
(voir ce tableau : https://www.supagro.fr/cnam-lr/statnet/tables.htm#student)

Nous vous conseillons de faire attention avant lancement, le paramètre par défaut étant à 20 ans, la simulation peut durée plusieurs longues minutes. (Environ 45 min sur un processeur AMD Ryzen 7 5800x et 32GB de RAM).

6 Conclusion

Pour conclure sur nos travaux, la simulation dans son ensemble fonctionne, les implémentations de l'écart type et de l'intervalle de confiance ainsi que la moyenne permet d'étudier l'évolution des lapins au fil du temps et des expériences. Il est cependant à noter que nous avons malheureusement été limité par le matériel et nous aurions bien apprécié avoir des machines plus puissante afin de prolongé la simulation dans un temps plus grand et sans limitations dans l'algorithme.

En vu des grande ambition pour cette projet et notre intérêt pour la simulation nous allons continuer à porté des amélioration et s'approcher de ce qui était prévu, vous pouvez suivre l'évolution via la repositoire Git : https://github.com/KaramNS/Rabbit-Simulation.git