Cours de *David Hill*

Année Universitaire 2023-2024

ZZ2 promo 25

TP 4

Une croissance de population plus réaliste

Jouen Martin

Table des matières

Introduction

1 Première simulation avec un langage orienté objet

1.1 Objectifs

Dans cette première partie, la simulation sera simplifiée vers un modèle plus simple, où les lapins sont regroupés par couple, où ils ne meurent pas.

Plus spécifiquement, les paramètres sont les suivants :

- Chaque couple de lapin met bas une fois par mois à un nouveau couple de lapins enfants
- Les couples de lapins enfants deviennent matures au bout du mois suivant.
- Les lapins ne vieillissent et ne meurent pas.

Cette simulation est réalisée dans un langage objet (C#), afin de montrer les avantages et inconvénients de ce paradigme pour ce genre de sujet. Cela permettra également d'orienter le choix du langage et de la structure pour la seconde partie qui se doit d'être beaucoup plus optimisée. De plus, ce paradigme paraît s'imposer à la modélisation de ce problème. En effet, chaque individu est une instance, avec son propre état (enfant ou adulte).

1.2 Implémentation

```
1  class PairLapin
2  {
3      public bool Adult { get; private set; }
4
5      public PairLapin()
6      { Adult = false; }
7
8      public void MoisSuivant(SimulationBasic s)
9      {
10         if (Adult)
11         { s.NouvellePairLapin(); }
12         else
13         { Adult = true; }
14     }
15 }
```

Extrait 1 - Classe de la paire de lapins

Voici la classe représentant un couple de lapins. La simulation étant très basique, la classe l'est aussi.

La classe simulation comptera trois attributs : le nombre de paires de lapins, le nombre de mois écoulés, ainsi que la liste des paires de lapins.

Cette liste utilisera une structure de donnée de type liste chaînée. En effet, sont avantage à pouvoir rajouter des éléments sans avoir à la redimentionner est intéressant pour ce problème. On peut aussi noter que comme le parcours de cette liste se fait élément par élément, et la lecture de chacun est en temps constant. La boucle dans la méthode *Simuler* permet un tel parcours de la liste (code 2)

```
class SimulationBasic
{
    public long NbMois                    { get; private set; }
    public long NbLapins                  { get; private set; }
    public LinkedList<PairLapin> LstLapin { get; private set; }

    public void NouvellePairLapin()
    {
        LstLapin.AddFirst(new PairLapin());
        NbLapins++;
    }

    public void Simuler(int n)
    {
        for (int i = 0; i < n; i++)
        {
            PrintLapin();
            for (LinkedListNode<PairLapin> l = LstLapin.First; l != null; l = l.Next)
            { l.Value.MoisSuivant(this); }
            NbMois++;
        }
    }
}
```

Extrait 2 - Classe de la simulation

Seule la méthode *NouvellePairLapin* est appelée par les instances de lapins. De ce manière on respecte le paradigme objet.

1.3 Résultats

Le mois 39 est atteint au bout de 100 secondes d'exécution, avec un total de 102 334 155 paires de lapins, et en occupant 4 Go de mémoire. On remarque que le temps de calcul double à chaque itération.

On constate facilement que cette solution consomme bien trop de mémoire, et a un temps d'exécution beaucoup trop long. On n'atteint même pas les quatre années de simulation que déjà la simulation est à 50 secondes pour une intervalle.

Ces deux problèmes devront être réglés dans la prochaine partie, pour obtenir une simulation plus efficace.

Cependant, ce modèle possède quelques avantages qui peuvent se révéler bien utile. Premièrement chaque individu possède son instance, il est donc possible de stocker une combinaison de caractéristiques

propre à chacun d'eux. De plus, si l'on connaît le paradigme objet, cette version est très facile à implémenter car très intuitive. Elle peut donc se révéler très efficace sur des petites populations.

2 Simulation plus réaliste

2.1 Choix de la structure de simulation

Afin d'obtenir des résultats réalistes avec une simulation stochastique, il faut être capable de réaliser une simulation rapidement, de l'ordre de quelque secondes. De cette manière, on peut réaliser des milliers de simulations dans un temps respectable.

Pour y parvenir, il faut favoriser les calculs traitant un groupe d'individus similaires plutôt que les traiter un par un.

Lors de la première simulation, nous avons été freinés par un coût en mémoire désastreux, mais une fois ce problème passé, le problème du temps de calcul fait surface. On peut d'ailleurs le constater dès le début : les itération deviennent très rapidement espacées.

Pour cela il faut donc réfléchir à une nouvelle structure de donnée qui permettrai de factoriser les lapins par leur caractéristiques semblables. Celles-ci sont leur catégorie (femelle, mâle, enfant), leur âge, et leur probabilité de survie. La probabilité de survie dépendant exclusivement de l'âge, celle-ci peut être calculée au moment opportun, il n'est donc pas nécessaire de la stocker.

L'idée la plus évidente pour factoriser ces données et que chaque paire âge/catégorie stocke le nombre de lapins auxquels cette paire de paramètres corresponds. Cette méthode est envisageable car : - Il y a un faible nombre de paramètres (2 : âge et catégorie) - Chaque paramètre a un nombre finit de valeur possible (seulement 3 pour la catégorie, et chaque lapin ne peut excéder les 15 ans d'existence).

Cette implémentation permet en effet de réduire grandement le coût en mémoire exponentiel, à seulement quelques tableaux de taille fixe, mais en plus de n'avoir que des entier à lire plutôt que des listes entière d'individus.

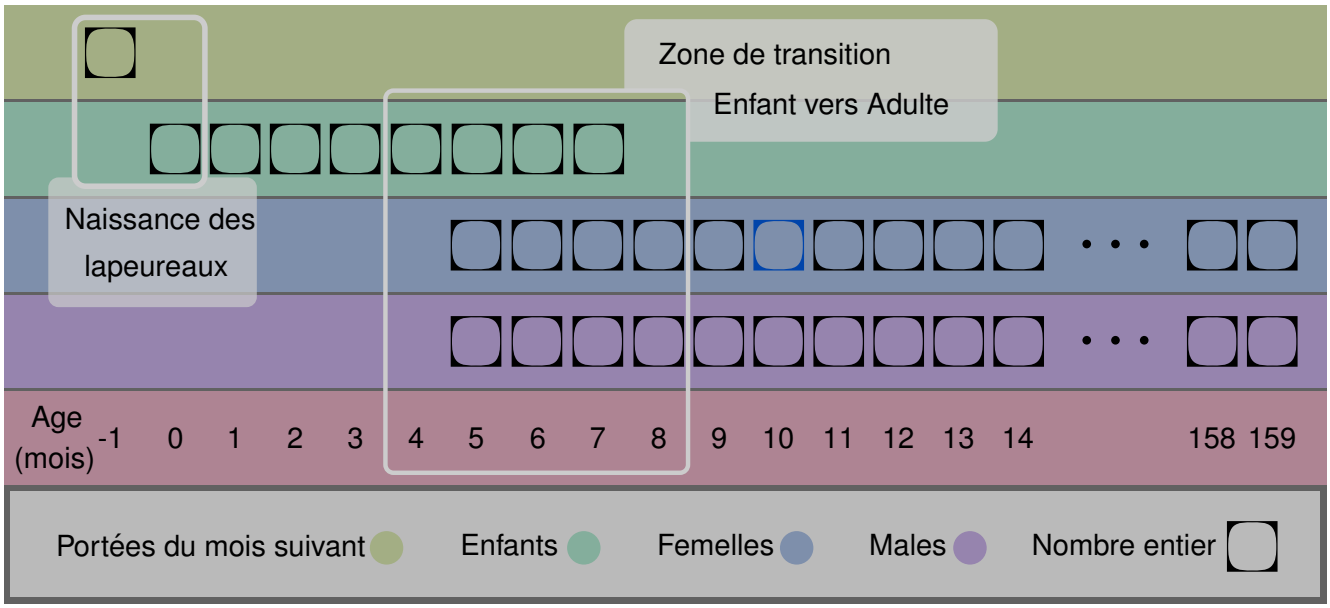

figure 1 - Agencement chronologique de la structure de donnée de la simulation.

Sur ce schéma, la structure de donnée de la simulation est représentée. Chaque case correspond à un **entier**, servant de compteur pour la combinaison de paramètres **âge** (en mois) et **catégorie** (portée, enfant, femelle et mâle). Ainsi la case en bleu correspond au nombre de femelles âgées de 10 mois.

Pour simplifier la structure, il a été choisis de réunir les enfants en une seule catégorie, et de déterminer leur sexe à leur passage à l'âge adulte, en les rangeant dans l'un des deux tableau correspondants.

On note deux zones particulière dans cette structure :

La première est la naissance des lapereaux. Depuis la case "naissances du mois suivant", un nombre de lapereaux (entre 3 et 6) est choisis pour chaque portée, qui est ajouté au premier compteur dans la catégorie "enfant". Cette case compte désormais le nombre d'enfants lapin âgés de 0 mois (ils viennent de naître).

Dans la seconde zone en évidence, les lapereaux peuvent passer à l'âge adulte dans l'intervalle d'âge de 5 à 8 mois. Le moment où le lapin passe à l'âge adulte est déterminé au hasard, de manière à ce que ce passage se fasse uniformément. A l'âge de 8 mois, tous les lapins sont adultes. A ce moment, leur sexe leur est attribué.

2.2 Choix de la méthode de simulation

Après avoir initialisé la simulation, il faut faire la faire avancer dans le temps. La première question à se poser pour cette étape est le pas de simulation. Les deux meilleures options candidates sont de mettre a jour mois par mois, ou année après année. La première permet une simulation plus précise, tandis que la seconde est plus simple à implémenter.

La mise à jour mois par mois a été choisie car elle permet de mieux simuler les enfant qui deviennent adultes jusqu'à 8 mois d'existence, donc moins précis si l'on simplifie à une année. D'autant plus que leur probabilité de survie est diminuée lors de cette période, il parait donc assez important de la garder la plus entière possible.

D'autres aspect rendent la simulation année par année moins uniforme. Par exemple, aucuns enfant ne peut être enfant à cheval sur deux années, ce qui ne correspond objectivement pas à une simulation réaliste. On pourrait éventuellement trouver des solution à ces imprécisions, mais cela nuirait au caractère facile à implémenter de cette méthode.

Cependant, le fait de coupler le pas de simulation par mois à la structure de donnée des tableaux d'âge empêche de stocker facilement de l'information pour chaque lapin, ce qui amène à d'autres problèmes. Prenons l'exemple du nombre de portée par femelles. Celui-ci est compris dans l'intervalle [4 ;8]. Ces bornes posent véritablement problème dans cette implémentation. Comment s'assurer que chaque femelles ai eu au moins 4 portée par année ? Cela est compliqué est génère d'autres problèmes. La méthode choisis à donc été poser une probabilité fixe de portées par mois, correspondant à l'espérance du nombre de portée de l'énoncé. Cependant, afin de rajouter un peu de réalisme dans cette simulation, il a été choisis d'attribuer un taux de reproduction spécifique à chaque, mois. Ainsi les mois hivernaux ont une probabilité très faible, et le printemps et l'été assurent quasiment les quatre portées minimales. De cette manière, on peut également réduire l'écart type du nombre de portée, qui peut mener à dépasser de l'intervalle. Voici un tableau montrant la probabilité par mois qui a été choisis pour cette simulation.

Mois	Proba (%)	Mois	Proba (%)
Janvier	3	Juillet	82
Février	2	Août	75
Mars	37	Septembre	48
Avril	69	Octobre	32
Mai	92	Novembre	9
Juin	89	Décembre	3

Figure 2 - Probabilité de reproduction des lapins à chaque mois de l'année

Voici donc la méthode qui a été choisie pour cette simulation, mais voici une autre implémentation possible pour remédier à cet intervalle car elle mène à des problèmes intéressant.

On pouvait choisir que chaque femelle, lors du passage à la nouvelle année, prévois un nombre aléatoire de portée dans un nouveau tableau, qui indique le nombre de portées dans les 12 prochains mois. Cette méthode résout parfaitement le problème des bornes de l'intervalle, mais apporte deux nouveaux problèmes.

Premièrement, jusqu'à preuve du contraire, les rongeurs ne fêtent pas le nouvel an, ou plutôt cette transition n'as pas lieu d'être dans cette simulation. En effet, imaginons le cas d'une femelle qui passerait à l'âge adulte en février, celle-ci devrait donc attendre onze mois jusqu'à la prochaine année

pour commencer à procréer. Cette situation n'as rien de naturel, surtout quand le printemps arrive. Deuxièmement, une fois que toutes les femelles ont ajoutés leur prévisions de portée pour l'année, celles-ci ont une chance de ne pas finir l'année. Cela implique de devoir réviser les prévisions, au risque de voir des portée fantômes apparaître. On ne connais pas le nombre de portées que la femelle fraîchement décédée avait prévu cette année. Il faut donc supprimer des naissances au hasard, ce qui amène le risque très probable d'en supprimer le mauvais nombre.

Ce deuxième problème apporte globalement la même imprécision que d'avoir un taux fixe de portée par mois. A savoir, tenter de répliquer un premier tirage aléatoire par un seconds avec les mêmes paramètres (écart type et moyenne). A choisir, la première méthode du taux fixe est préférable car elle n'apporte qu'un seul problème mineur (mineur dans le sens où il peut être réduit en choisissant les bonnes valeurs de paramètres).

On peut noter en plus de ça que ces problèmes sont réellement graves dans des cas de très faible population (une seule femelle restante ne pouvant pas donner bas car la tradition des lapin veut qu'elle prévoit ses portées au nouvel an). Comme la population est amenée à croître très fortement, ceux-ci auront peu d'impact, mais on peut facilement imaginer qu'un tel scénario est accessible en début de simulation quand la population est encore faible, mettant en péril la survie de l'espèce.

2.3 Mise à jour de la simulation

Finalement la plus grosse partie de cette simulation réside dans la mise à jour de la simulation d'un moment au suivant. Il faut tenir compte d'actualiser les individus dans le bon ordre.

Voici l'ordre choisis pour cette implémentation.

- Les lapins adultes vieillissent (du plus ancien au plus jeune)
- Les enfants pouvant passer adulte vieillissent, et déterminent leur sexe.
- Les enfants plus jeunes vieillissent
- Naissances des lapereaux du mois précédent
- Finalement les lapins adultes procréent.

De cette manière, on fait vieillir tous les lapins du plus ancien au plus jeune (plus facile dans ce sens avec la structure du tableau). Puis une fois que l'on a décidé quel lapins ont survécus au mois, on peut décider du nombre de portée. Cet ordre est justifiable par le temps de gestation des femelles arrondis à un mois. Dans la réalité, si la femelle tombe enceinte au début du mois, et qu'elle meurent avant la fin, sa portée ne verra pas le jour malheureusement.

En terme de reproduction, seul le nombre de femelles détermine le nombre de portées pour le mois prochain. On estime qu'il y aura toujours assez de mâles par rapport aux femelles. Le cas où les femelles sont réellement en surnombre est trop rare pour valoir la peine d'être calculé.

Voici quelques parties importantes de cette mise à jour, le concept de base étant similaire selon la tranche d'âge, seul le plus simple sera expliqué.

Voici le cas du vieillissement des enfants en bas âge :

```
// Les autres lapin enfants vivants veillissent
for (int age = MIN_MATURITE_SEXUELLE - 1; age > 0; age--)
{
    s->nb_enfant          -= s->lapin_enfant[age-1];

    entier enfant = 0;
    for (int j = 0; j < s->lapin_enfant[age-1]; j++)
    {
        enfant += veillir_lapin(age);
    }

    s->nb_enfant += enfant;
}
```

Extrait 3 - Vieillissement des lapins enfants : précis

Pour chaque enfant de chaque âge, on effectue un tirage déterminant sa survie (à ne jamais appliquer en situation réelle), et on met à jour le compteur global de sa catégorie. Cette méthode est très précise, mais aussi très lente, car elle dépend du nombre de lapins. Avec cette méthode, la simulation commence à fortement ralentir dès les 120 mois de simulation, soit 10, et donc la moitié des 20 ans attendus. Pour rendre ces calculs plus rapides, dès qu'une tranche d'âge comporte trop d'individus, on bascule sur la génération d'un nombre aléatoire répartit selon une gaussienne (simulée par la méthode de Box et Muller), qui permet un calcul en temps constant. Voici le même code mais adapté avec cette nouvelle méthode.

```
// Les autres lapin enfants vivants veillissent
for (int age = MIN_MATURITE_SEXUELLE - 1; age > 0; age--)
{
    s->nb_enfant -= s->lapin_enfant[age-1];

    entier enfant = 0;
    if (s->lapin_enfant[age-1] < MAX_SIMU_COMPLETE)
    {
        for (int j = 0; j < s->lapin_enfant[age-1]; j++)
        {
            enfant += veillir_lapin(age);
        }
    }
    else
    {
        double random = gen_rand_gaussienne() / ECART_TYPE;
        enfant = (entier)((random + PROBA_SURVIE_ENFANT) * s->lapin_enfant[age-1]);
    }
    s->lapin_enfant[age] = enfant;
    s->nb_enfant        += enfant;
}
```

Extrait 4 - Vieillissement des lapins enfants : optimisé

La valeur de la constante *ECART_TYPE* (1/6) n'a aucun fondement statistique, mais est seulement présente afin de resserrer les valeurs générées par la Gaussienne. Toutes les valeurs aléatoires sont obtenue par le PRNG Mersenne-Twister.

Le même principe s'applique ensuite sur les autres étapes de la simulation, avec parfois un double degré d'aléatoire comme lors de la survie des lapin de 4 à 8 mois suivit de leur possible passage à l'âge adulte.

3 Résultats

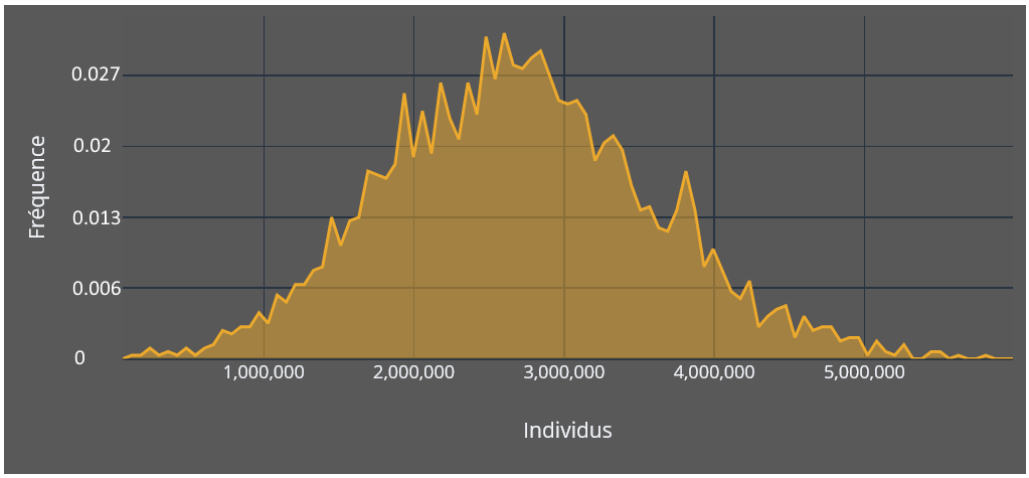

Figure 3 - Répartition stochastique de la taille de la population après 80 mois. Méthode la plus précise.

Résultats pour 3000 simulation de 80 mois en n'utilisant que la méthode exacte (tirage aléatoire pour chaque individu). Malheureusement, cette méthode est bien trop lente pour obtenir des résultats de 240 comme pour la méthode rapide (plus de 10 minutes pour une seule simulation). Cependant, cela est suffisant pour observer une gaussienne apparaître

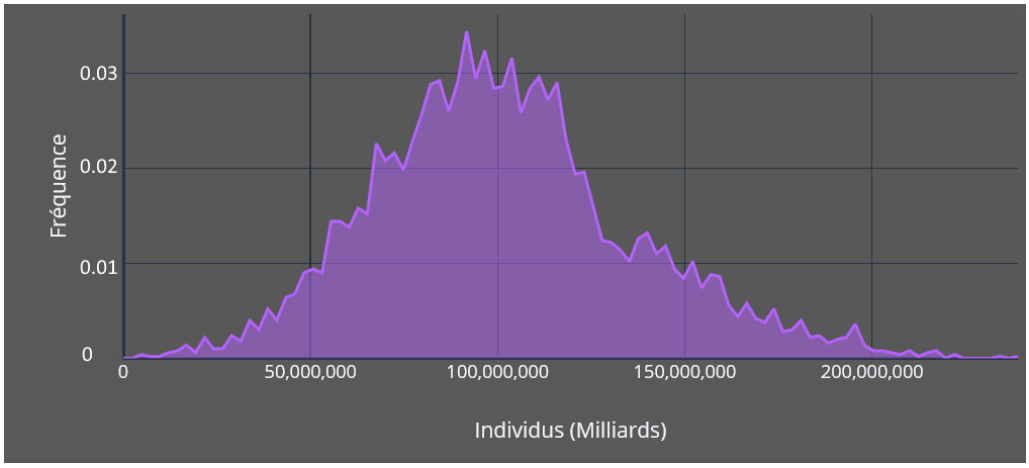

Figure 4 - Répartition stochastique de la taille de la population après 240 mois. Méthode la plus rapide.

Résultats pour 5000 simulation de 240 mois (20 ans) en mixant la méthode exacte pour les faibles population, avec la méthode rapide (répartition selon une gaussienne). La gaussienne obtenue est légèrement plus étroite que celle obtenue naturellement, il faudrait diminuer ”l'écart-type” utilisé dans la fonction *gen_rand_gaussienne* pour obtenir un résultat plus similaire.

Conclusion

Nous avons pu étudier un cas concret d'une simulation stochastique. De nombreux raccourcis sont possibles pour optimiser la vitesse d'exécution, mais il faut veiller à reproduire le comportement original le plus fidèlement possible. Il y a souvent des compromis à faire en terme de choix d'implémentation, et il faut prendre en compte à quel ordre de grandeur ces compris impactent le résultat final.

Nous avons utilisé une structure de donnée en tableau, car les combinaisons de paramètres pour les individus étaient convenables. Cependant, un langage objet peut être préférable dans un scénario où chaque individu a un état plus unique. Cependant, les optimisations seront moindres, et il faudra ajuster l'échelle de la simulation.