



Rapport TP Simulation

Lab #4a – A more realistic population growth

BALLEJOS Lilian & CHARPIN Etienne
ISIMA INP Deuxième année
Spécialité F2
Année Universitaire 2022/2023

Enseignant référant : Mr David HILL

Date du rendu : 21 décembre 2022

Date de la dernière modification : 19 décembre 2022

ISIMA INP

1 rue de la Chebarde - TSA 60125 - CS 60026 - 63178 Aubière CEDEX

Tel : 04 73 40 50 00

Site web : isima.fr

Table des Matières

1	Exercice 1 : Simulation d'une population avec Fibonacci	2
2	Exercice 2 : Une meilleure Simulation	2
2.1	Théorie	2
2.2	Les structures	2
2.2.1	Représentation d'un booléen en C	2
2.2.2	Représentation d'un Lapin	3
2.2.3	Représentation d'un Couple	3
2.2.4	Tableaux de probabilité	3
2.3	InitListRabbit	4
2.4	setLitters	4
2.5	CreateCouple	4
2.6	MAJMaturite	4
2.7	selectionNaturel	5
2.8	SimulEvoRabbit	5
2.9	Quelques exemples de résultat	5
2.10	Les Fontions Outils	6
2.10.1	AfficheListLapin	6
2.10.2	AfficheListCouple	7
2.10.3	AddRabbit	7
2.10.4	deleteRabbit	7
2.10.5	tuer	7

Introduction

Nous allons dans ce TP nous pencher sur l'évolution d'une population de lapin en essayant de simuler celle-ci le mieux possible.

Quelques remarques sur les fichiers .c

La compilation s'effectue via un **Makefile**.

Liste des fichiers fournis pour ce TP :

- mt.c
- mt.h
- lapin.c
- lapin.h
- tp4_simulation.c
- makefile
- tp4_ballejos_charpin.pdf

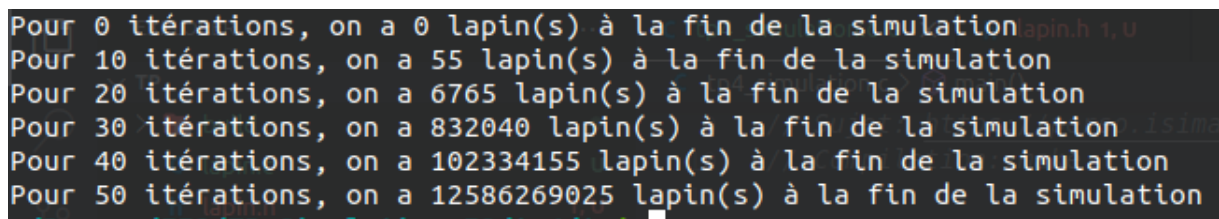
Nous avons réutilisé lors de ce TP les fonctions "MT" de génération de nombre aléatoire de Matsumoto que vous pouvez retrouver [ici](#).

1 Exercice 1 : Simulation d'une population avec Fibonacci

Pour commencer, nous allons effectuer une simulation plutôt simple pour représenter l'évolution d'une population de lapin.

Celle-ci va suivre la suite de Fibonacci. Ainsi, une génération de lapin sera représentée par une itération de la célèbre suite. C'est notre fonction **RabbitFibo** qui effectuera ce comportement.

Nous obtenons des résultats plutôt concluant même si très peu réaliste :



```
Pour 0 itérations, on a 0 lapin(s) à la fin de la simulation
Pour 10 itérations, on a 55 lapin(s) à la fin de la simulation
Pour 20 itérations, on a 6765 lapin(s) à la fin de la simulation
Pour 30 itérations, on a 832040 lapin(s) à la fin de la simulation
Pour 40 itérations, on a 102334155 lapin(s) à la fin de la simulation
Pour 50 itérations, on a 12586269025 lapin(s) à la fin de la simulation
```

FIGURE 1 – Sortie de RabbitFibo() avec différents nombres d'itération

Plusieurs éléments font que cette simulation n'est pas très réaliste :

- Il n'y a pas une gestion de la mort des lapins
- Ce n'est pas une simulation stochastique car nous avons aucun aléatoire : nous suivons seulement une suite mathématique.

Nous allons maintenant essayer de faire une simulation plus réaliste qui représentera mieux l'évolution d'une population de lapin.

2 Exercice 2 : Une meilleure Simulation

2.1 Théorie

Nous allons maintenant implémenter une simulation plus réaliste qui prendra en compte :

- Un taux de mortalité dépendant des maladies et prédateurs possibles
- Le vieillissement des lapins
- Des portées possibles pour les lapines plus réaliste dépendant de leur âge et leur état de santé
- La répartition des mâles et femelles dans la population
- Les paramètres de démarrage de la simulation (nombre de lapin au départ et nombre d'itération)

Nous allons expliquer dans ce rapport le comportement des fonctions implémentées une à une et leur utilité dans la simulation.

Mais avant, voyons quelles structures nous avons utilisé pour créer cette simulation

2.2 Les structures

2.2.1 Représentation d'un booléen en C

Nous avons tout d'abord créé un pseudo-type booléen afin de simplifier la syntaxe de notre code pour la suite en créant une *enum* **boolean_t** qui prend pour valeur true ou false.

```
1     typedef enum {
2         false = 0,
3         true = 1
4     } boolean_t;
5
```

FIGURE 2 – Structure permettant de "créer" un type booléen

2.2.2 Représentation d'un Lapin

Puis nous avons créée la structure **lapin_t** qui représente un lapin avec toutes les caractéristiques dont nous avons besoin dont son âge, si c'est un mâle ou une femelle ...

```
1     typedef struct{
2         int     age;
3         boolean_t adult;
4         boolean_t isMale;
5         int     nbLitterThisyear;
6         boolean_t used;
7     }lapin_t;
8
```

FIGURE 3 – Structure permettant de représenter un lapin

2.2.3 Représentation d'un Couple

Ensuite, nous avons créé un type **couple_t** qui représente un couple de lapin et qui sera utile pour gérer la reproduction.

```
1     typedef struct{
2         int     male;
3         int     female;
4     }couple_t;
5
```

FIGURE 4 – Structure permettant de représenter un couple de lapin

2.2.4 Tableaux de probabilité

Enfin, nous avons créé des tableaux contenant les différentes probabilités dont nous aurons besoin pour notre simulation :

```

1      static double proba_porte[5]      = { 0.125 , 0.375 , 0.625 , 0.875 , 1 };
2      static int   proba_nb_child[4]     = { 3 , 4 , 5 , 6 };
3      static double proba_maturite[4]    = { 0.15 , 0.25 , 0.5 , 1 };
4      static double proba_survi[7]       = { 0.35 , 0.6 , 0.5 , 0.4 , 0.3 , 0.2 ,
5      0.1 };

```

FIGURE 5 – Tableau de probabilité pour nos expériences

proba_porte contient la probabilité du nombre de portée qu'il y aura dans une année.

proba_nb_child contient la probabilité du nombre d'enfant que va produire la portée.

proba_maturite contient la probabilité de devenir mature ou non selon l'âge d'un lapin

proba_survi contient la probabilité de survivre à la fin de l'année pour un lapin selon son âge.

2.3 InitListRabbit

C'est la première fonction que nous avons dû implémenter.

Elle ajoute dans notre tableau de lapin en vie, le nombre de lapin passé en paramètre. On peut spécifier si ces lapins seront déjà prêt à se reproduire ou non.

On injecte autant de mâle que de femelle.

2.4 setLitters

Cette fonction parcourt tout notre tableau de lapin en vie et détermine pour chaque femelle le nombre de portée qu'elle aura dans l'année qui varie de 4 à 8 selon une probabilité tirée aléatoirement.

On compare ce tirage aux valeurs stockées dans notre tableau **proba_porte** pour savoir le nombre de portée que pourra effectuer la dite femelle.

2.5 CreateCouple

Cette fonction va créer des couples de lapin qui vont ensuite pouvoir se reproduire entre eux pour créer de nouvelles portées.

Pour cela on liste le nombre total de lapine pouvant se reproduire et ayant encore une possibilité d'avoir au moins une portée durant cette année.

On met en lien ces lapines avec des mâles afin de créer des couples.

Au final on renvoi le nombre de couple créé.

2.6 MAJMaturite

Cette fonction va mettre à jour la maturité des lapins.

On parcourt tout le tableau des lapins en vérifiant à chaque fois si le lapin à mettre à jour n'est bien pas un adulte. De plus on vérifie que celui-ci est âgé entre 5 et 8 ans

Si c'est le cas, on tire aléatoirement un nombre et on compare celui-ci avec la case du tableau de probabilité **proba_maturite** correspondant à l'âge de notre lapin .

Si cette probabilité est inférieure à celle du tableau, il devient mature!

2.7 selectionNaturel

Cette fonction va choisir si oui ou non les lapins doivent mourir.

On parcourt le tableau contenant tout nos lapin et on vérifie si oui ou non on doit les tuer.

Pour cela on tire un nombre aléatoire et on le compare avec la probabilité contenue dans la case du tableau **proba_survi** qui correspond au cas de notre lapin actuel.

Les différents cas sont :

- Le lapin n'est pas mature
- Le lapin est jeune (moins de 10 ans)
- Le lapin est trop vieux (plus de 16 ans), il meurt qui qu'il arrive
- Le lapin à entre 10 et 16 ans (probabilités variables selon l'âge dans cette intervalle)

Sauf dans le cas où le lapin est trop vieux, si la probabilité tirée est inférieure ou égale à celle dans le tableau **proba_survi**, on ne fait rien sinon, on tue le lapin.

2.8 SimulEvoRabbit

Enfin nous y voilà, notre fonction **SimulEvoRabbit** qui va s'occuper de faire la simulation.

Pour cela, on initialise au départ notre liste de lapin avec **InitListRabbit**.

Ensuite on lance la simulation sur le nombre d'itération voulue sachant qu'une itération correspond à un mois dans notre simulation.

Ainsi on teste chaque année de notre simulation (Itération % 12) la mortalité de nos lapins avec **selectionNaturel**

On crée aussi des couples à chaque itérations avec **CreateCouple** et pour chacun d'eux un nombre de portée qu'ils vont avoir. On va donc pour chacune des portées, de chacun des couples, calculer combien d'enfant vont être généré à l'aide du tableau **proba_nb_child**.

Enfin à chaque fin d'itération on ajoute un an de plus à chaque lapin !

2.9 Quelques exemples de résultat

Observons avec les probabilités choisies comment notre simulation évolue au cours du temps sur 10 années.

Tout d'abord voyons avec seulement deux lapins au départ :

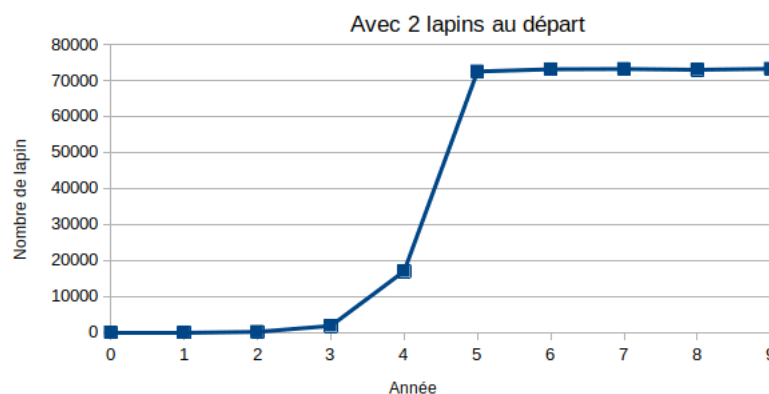


FIGURE 6 – Évolution de la population sur 10 ans avec 2 lapins au départ

On remarque que à la 5ème année on atteint un plateau dit "stable" où le nombre de mort et de naissance se compensent gardant à chaque fois une population d'environ 73 000 lapins.

Voyons comment réagit notre population maintenant avec 10 lapins au départ

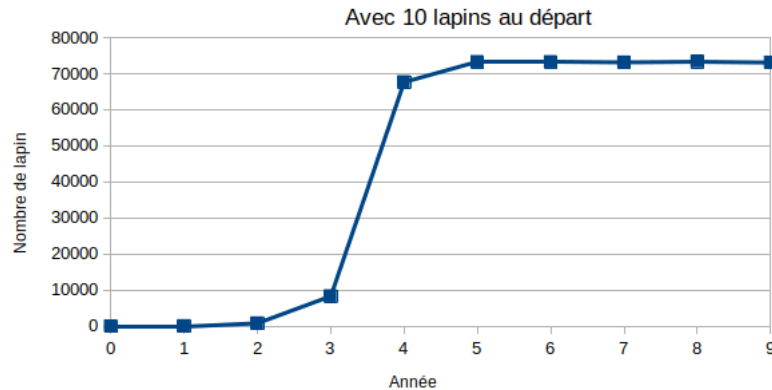


FIGURE 7 – Évolution de la population sur 10 ans avec 10 lapins au départ

On remarque que encore une fois notre population atteint un plateau à environ 73 000 lapins mais cette fois-ci on converge vers ce plateau plus vite (à la 4ème année soit 1 ans de moins qu'avec 2 lapins)

Testons de voir si on obtient le même plateau avec 100 lapins au départ et si l'on peut converger plus vite dessus !

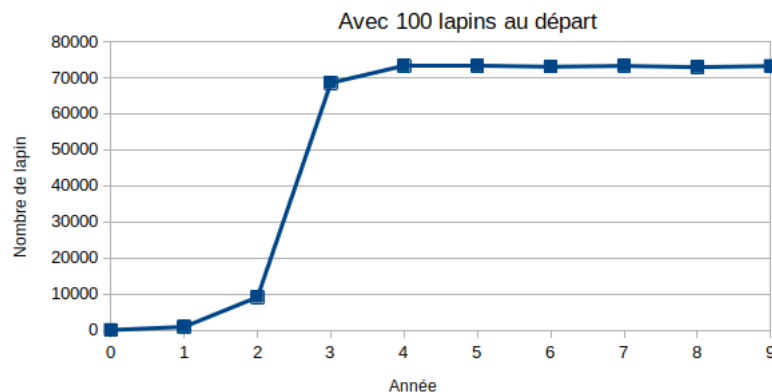


FIGURE 8 – Évolution de la population sur 10 ans avec 100 lapins au départ

On obtient bien le même plateau qui est cette fois-ci atteint en seulement 3 ans !

2.10 Les Fontions Outils

Avant de finir le rapport nous souhaitons tout de même vous expliquer l'utilité et le comportement des fonctions "outils" qui nous ont été nécessaires.

2.10.1 AfficheListLapin

Cette fonction nous permet d'afficher 1 à 1 tous les lapins contenus dans notre tableau de lapin en vie avec leurs caractéristiques.

2.10.2 AfficheListCouple

Cette fonction nous donne le nombre de couple de lapin que nous avons à l'instant t où nous l'appelons.

Elle affiche ensuite 1 à 1 les couples de lapin.

2.10.3 AddRabbit

Cette fonction vérifie que la capacité maximale de stockage de notre tableau de lapin n'est pas atteinte et si c'est le cas, elle ajoute un nouveau lapin né dans le tableau.

2.10.4 deleteRabbit

Cette fonction enlève un lapin à l'indice voulu de notre liste des lapins en vie.

Pour cela on effectue un décalage vers la gauche sur notre tableau afin d'écraser la case du lapin actuel à supprimer et de garder un tableau propre avec seulement les lapins en vie dedans.

2.10.5 tuer

Cette fonction fait appel à la fonction **deleteRabbit** afin d'enlever le lapin actuel que l'on veut tuer de la liste des lapins en vie. Elle met ensuite à jour le compteur du nombre de lapin tué et du nombre de lapin encore en vie.

Table des figures

1	Sortie de RabbitFibo() avec différents nombres d'itération	2
2	Structure permettant de "créer" un type booléen	3
3	Structure permettant de représenter un lapin	3
4	Structure permettant de représenter un couple de lapin	3
5	Tableau de probabilité pour nos expériences	4
6	Évolution de la population sur 10 ans avec 2 lapins au départ	5
7	Évolution de la population sur 10 ans avec 10 lapins au départ	6
8	Évolution de la population sur 10 ans avec 100 lapins au départ	6