Compte rendu du TP de simulation

Nathan DESNOS

Mathieu VILLEDIEU DE TORCY

Sommaire :

- Introduction
- Simulation de population de lapins simple (Fibonacci)
  - Fibonacci récursive
  - Fibonacci récursive terminale
  - Fibonacci iterative
  - Comparaison
  - Conclusion
- Simulation de population de lapins réaliste
  - Principe
  - Organisation du code
  - Générateur et lois utilisées
  - Résultats
  - Pistes d'amélioration

Introduction

Le but du TP est de réaliser une simulation de l'evolution d'une population de lapins. Nous avons besoin d'un générateur de nombre aléatoire pour ce TP, et nous allons reprendre Mersenne Twister de Makoto Matsumoto d'un TP précédent.

Simulation de population de lapins simple (Fibonacci)

Un model simple pour simuler l'évolution d'une $u_{n+2} = u_{n+1} + u_n$ avec $u_0 = u_1 = 1$. Avec ce model, u_n représente le nombre de lapins dans la population aux mois n.

Fibonacci récursive

On appliquant directement la définition de la suite de Fibonacci. on définit donc une fonction récursive. La suite de Fibonacci croît rapidement ce qui nous oblige à utiliser des `long` pour stocker les valeurs.

```
/*************************************************/
/* fibo_rec : fct fibonacci f(n)=f(n-1)+f(n-2)    */
/*            calcul de facon recursive           */
/*                                                */
/* entree : un entier -> iteration n              */
/*                                                */
```

```
/* sortie : un long                                 */
/**************************************************/
long fibo_rec(int n)
{
    if (n == 0 || n == 1)
        return 1;
    else
        return fibo_rec(n - 1) + fibo_rec(n - 2);
}
```

Fibonacci récursive terminale

Afin d'améliorer la fonction récursive précédente, on la transforme en fonction récursive *terminale*, ce qui limite le nombre d'appels récursifs.

```
/**************************************************/
/* fibo_rec_terminale : fct fibonacci             */
/*                      fonction rec terminale    */
/*                                                */
/* entree : un entier -> iteration n              */
/*          un long -> u_n                        */
/*          un long -> u_n_1                      */
/*                                                */
/* sortie : un long                               */
/**************************************************/
long fibo_rec_terminale(int n, long u_n, long u_n_1)
{
    if (n == 0 || n == 1)
        return u_n;
    else
        return fibo_rec_terminale(n - 1, u_n + u_n_1, u_n);
}
```

Fibonacci iérative

On peut également transformer la fonction récursive par definition en fonction itérative beaucoup plus performante.

```
/**************************************************/
/* fibo_iter : fct fibonacci f(n)=f(n-1)+f(n-2)   */
/*             calcul de facon iteratif           */
/*                                                */
/* entree : un entier -> iteration n              */
/*                                                */
/* sortie : un long                               */
/**************************************************/
long fibo_iter(int n)
```

```
{
    long u_n = 1;
    long u_n_1 = 0;
    long tmp;

    for (int i = 0; i < n; ++i)
    {
        tmp = u_n;
        u_n = u_n + u_n_1;
        u_n_1 = tmp;
    }
    return u_n;
}
```

Comparaison

Pour differentes valeurs de *n*, nous avons utiliser les différentes fonctions pour le calcul de u_n.

Pour **n = 40** :

```
fibo_rec( 40 ) = 165580141
temps d'execution 1.317778 s pour n = 40

fibo_rec_terminale( 40 ) = 165580141
temps d'execution 0.000002 s pour n = 40

fibo_iter( 40 ) = 165580141
temps d'execution 0.000001 s pour n = 40
```

Pour **n = 50** :

```
fibo_rec( 50 ) = 20365011074
temps d'execution 169.906258 s pour n = 50

fibo_rec_terminale( 50 ) = 20365011074
temps d'execution 0.000149 s pour n = 50

fibo_iter( 50 ) = 20365011074
temps d'execution 0.000002 s pour n = 50
```

	n	fibo_rec	fibo_rec_terminale	fibo_iter
temps d'execution	40	1.317778 s	0.000002 s	0.000001 s
temps d'execution	100	169.906258 s	0.000149 s	0.000002 s

On remarque bien que `fibo_rec` n'est pas du tout efficace.

Conclusion

La suite de Fibonacci donne une représentation très simplifiée de l'évolution de la population de lapins. En effet, il n'y a que des naissances de lapins et la population ne peut donc qu'augmenter. Cette méthode n'est donc pas la plus pratique et réaliste, mais permet de se rendre compte la vitesse de croissance de la population.

Simulation de population de lapins réaliste

Organisation du code

Le code est constitué :

- tp4.c
- tp4.h
- rabbit_simu.c
- rabbit_simu.h
- lapin.c
- lapin.h
- makefile

Le fichier *tp4.c* contient les fonctions de génération de nombre aléatoire Mersenne Twister de Makoto Matsumoto, des fonctions générant des lois aléatoires, et les fonctions fibonacci de la partie 1.

Le fichier *lapin.c* contient les fonctions qui concerne les lapins directements : création/mort de lapin etc...

Le fichier *rabbit_simu.c* contient les fonctions gérant la simulation, boucles principales des évènements dans un mois.

Pour chaque lapin, on utilise la structure suivante, pour définir ses caractéristiques :

```
typedef struct lapin
{
    int age; // Age en mois
    enum Sexe sexe;
    int ageMaturite;
    enum Maturite maturite;

    // Pour les femelles
    int nbPortees;
    int moisPortees[12];

} lapin_t;
```

On utilise cette structure pour garder à jour les informations sur la population :

```c
typedef struct infoPop
{
    int nbMale;
    int nbMaleAdulte;
    int nbMaleBebe;

    int nbFemelle;
    int nbFemelleAdulte;
    int nbFemelleBebe;

    int nbAdulte;
    int nbBebe;
    int nbTotal;

} infoPop_t;
```

Les lapins de la population sont stockés dans deux tableaux pour la distinction des males et femelles :

```c
typedef struct population
{
    lapin_t *male[POPULATION_MAX];
    lapin_t *femelle[POPULATION_MAX];
} population_t;
```

Principe

Pour chaque mois simulés, on va effectuer les opérations suivantes sur chaque lapin :

- Anniversaire des lapins
  - Incrémente l'age du lapin et vérifie s'il passe adulte
- Definie les mois de portees
  - Tous les premiers mois d'une année, définit le nombres de portée d'une femelle pour l'année
- Reproduction
  - Pour un male et une femelle adultes crée la portée avec le bébés lapins
- Mort des lapins
  - Vérifie si le lapin meurt ce mois-ci

Afin de débuter la population de lapin, il faut les proteger durant la première année en ne regardant pas s'ils doivent mourrir. Les bébés ayant une faible chance de survie et le premier couple étant des bébés, il faut attendre un certain temps avant l'age de reproduction et la première portée.

Générateur et lois utilisées

Le générateur aléatoire utilisé est celui des TP précédent (Mersenne Twister de Makoto Matsumoto).

Le nombre de portée qu'une lapine peut avoir au cours d'une année suit une **loi normale N(6,1)** générée par la fonction suivante :

```
int loiNormale(int mu, int sigma)
{
    int x = box_muller(); // x -> N(0,1)
    x *= sigma;
    x += mu;
    return x;
}
```

Le nombre de bébé lapins lors d'une portée suit une **loi uniforme** entre 3 et 6.

La mort du lapin chaque mois est calculée suivant son âge, avec les probabilités suivantes :

```
// age année -> chance survie :
// 10 ans -> 0.5 (an) / 0.94387 (mois)
// 11 ans -> 0.4 (an) / 0.92648 (mois)
// 12 ans -> 0.3 (an) / 0.90454 (mois)
// 13 ans -> 0.2 (an) / 0.87449 (mois)
// 14 ans -> 0.1 (an) / 0.82540 (mois)
// 15 ans -> 0.0001 (an) / 0.46416 (mois)
//
// Age < 10 ans :
// Adulte -> 0.6  (an) / 0.95832 (mois)
// Bebe   -> 0.35 (an) / 0.91623 (mois)
```

Résultats

En lançant la simulation pour 9 ans, (9*12 mois) :
Au premier mois, population initiale

```
 Affichage etat de la population

 nbMale : 1
 nbMaleAdulte : 0
 nbMaleBebe : 1

 nbFemelle : 1
 nbFemelleAdulte : 0
 nbFemelleBebe : 1

 nbAdulte : 0
 nbBebe : 2
 nbTotal : 2
```

A la fin de la simulation

```
Affichage etat de la population

nbMale : 21829277
nbMaleAdulte : 7471098
nbMaleBebe : 14358179

nbFemelle : 21835320
nbFemelleAdulte : 7470551
nbFemelleBebe : 14364769

nbAdulte : 14941649
nbBebe : 28722948
nbTotal : 43664597
```

Pistes d'amélioration

Nos tableaux contenant les lapins males et femelles ont une taille de 100 000 000. Or au dela de 9 ans de simulation, la population devient supérieure.
Nous nous limitons donc à 9 ans de simulation.

Pour pouvoir simuler sur plus longtemps, il faudrait réequilibrer le ratio naissance/mort :

- rajouter un système de prédateur afin de réguler la population
- augmenter le taux de mortalité
- diminuer le taux de natalité
- prendre en compte l'age de la lapine dans les calculs du nombre de portées (fréquence et quantité diminués avec l'age)
