Antony Nifosi

Rapport TP4 Simulation

David Hill

1- Présentation du problème :

Le but de ce TP était de simuler la reproduction d'une population de lapins en ayant une approche un peu plus précise que la suite de Fibonacci qui peut elle aussi estimer la croissance d'une population de lapins.

2- Modélisation :

Il était tout d'abord question de choisir quel modélisation nous allions utiliser pour représenter notre problème.
Tout d'abord pour toutes les simulation de ce TP nous avons utilisé lorsque l'on souhaité générer des nombres aléatoires un générateur Mersenne Twister disponible dans la std.

```
std: :mt19937 generator(8);
std::mt19937 generator2(8);
```

Pour représenter nos lapins on va créer une classe Rabbit ainsi qu'une classe Female qui héritera de la classe Rabbit pour spécifier certaines actions que seules les lapines femelles font comme par exemple donner naissance à des portées.
On aurait pu très bien avoir une sorte de booléen pour décider si oui ou non le lapin est un mâle et tester dans notre code avec un simple if pour faire ou non les actions réservées aux lapines mais le faîte d'avoir une classe ne va pas alourdir le code et va pouvoir de bien distinguer les fonctions liées aux lapins en général et celle liées aux femelles.
Comme on peut le voir la classe Rabbit est doté de plusieurs attributs :

```
class Rabbit
{
    protected:
        bool adult;
        int age;
        float survival;
        int maturity;
        int death_month;
        float proba_death;

    public:
        Rabbit();
        virtual bool update();
        virtual void anniversary();
        bool hasToDie();
};
```

Figure 1: Déclaration de la classe Rabbit notre classe pour les mâles

- Un booléen pour savoir si le lapin a atteint ou non la majorité sexuelle si oui il sera considéré comme adulte.
- Un entier stockant son age en mois.
- Son taux de survie actuelle
- Un entier stockant son âge auquel il est censé devenir adulte.
- Et un réel stockant sa probabilité de mourir chaque mois selon son taux de survie actuel.

En ce qui concerne la classe Female elle n'a que deux attributs supplémentaires :

```
class Female : public Rabbit
{
    private:
        int porte;
        std::vector<int> mois;

    public:
        Female();
        void anniversary();
        bool update();
};
```

Figure 2: Déclaration de la classe Female notre classe pour les lapines

- Un entier désignant combien de portée elle aura dans l'année
- Un vector contenant les mois de l'année pendant les quels elle aura ses portées

La dernière classe que l'on utilisera sera la classe Population qui servira à stocker notre population de lapins dans une std::list ici très utile puisque très performante particulièrement quand on souhaitera retirer les lapins mort de la liste la suppression dans cette structure de données se fait en O(1).
Nous avons également quelques données comme le nombre de lapins, mort ou naissances par mois dans des vectors ainsi que le nombre total de morts.

```
class Population
{
    private:
        std::list<Rabbit*> population;
        std::vector<int> nb_rabbit;
        std::vector<int> nb_birth;
        std::vector<int> nb_death;
        int mort_count;
        std::ofstream file;

    public:
        Population(int femelle, int male);
        void update(int time);
        void print_data();
        void write_data(int pop, int birth, int death);
};
```

Figure 3: Déclaration de la classe Population classe gérant la population de tous les lapins de la population

3 – Fonctionnement :

Le fonctionnement reste assez simple, au lancement du programme on précisé combien de mâles et de femelles souhaite t'on à l'initialisation ainsi que la durée de la simulation :

```
Population::Population(int femelle, int male) : mort_count(0)
{
    /* Creation du fichier de donnees pour l'histogramme */
    file.open("result.txt");

    /* Initialisation de la population avec des males et des femmelles */
    for (int j = 0; j < femelle; j++)
    {
        population.push_back(new Female());
    }
    for (int j = 0; j < male; j++)
    {
        population.push_back(new Rabbit());
    }
    write_data(femelle + male, 0 , 0);
}
```

Figure 4: Constructeur de la classe Population + initialisation de la population

Une fois la population on appelle la fonction update() de la classe Population qui va permettre de lancer la simulation de notre population de lapins.

La routine de la simulation se déroule de la manière suivante :

```
for (int j = 0; j < time; j++)
{
    for (auto i = population.begin(); i != population.end(); i++)
    {
        /* Si le lapin doit mourrir */
        if ((*i)->hasToDie())
        {
            delete *i;
            *i = nullptr;
            death_tmp++;
            mort_count++;
        }
```

Figure 5: Vérification de la survie des lapins au mois j

Pour tous les lapins que l'on stocke dans notre std::list on regarde si il doivent mourir ou non grâce à la fonction hasToDie() :

```
bool Rabbit::hasToDie()
{
    bool die = false;
    float mort = survivalite(generator);

    if (mort > proba_death)
    {
        die = true;
    }
    return die;
}
```

Figure 6: Fonction retournant si oui ou non le lapin doit mourir au mois courant

La fonction va générer un nombre aléatoire entre 0 et 100 et va regarder si il est plus grand ou non que la probabilité que le lapin survive dans le mois courant. Si c'est le cas la fonction va renvoyer true sinon elle renverra false. Une fois revenue dans la fonction upodate() de Population si hasToDie() a renvoyé vrai on delete le lapin courant pointé par l'itérateur et on met son pointeur a nullptr et on incrémente les compteurs pour les statistiques.

Si le lapin ne doit pas mourir alors on va mettre à jour son age :

```
else
{
    if ((*i)->update())
    {
        int nb_baby = naissance(generator2);
        for (int k = 0; k < nb_baby; k++)
        {
            birth_tmp++;
            if (sexe(generator2))
            {
                population.push_back(new Female());
            }
            else
            {
                population.push_back(new Rabbit());
            }
        }
    }
}
```

Figure 7: Zoom sur la fonction update() de la classe Rabbit

La fonction update() de la classe Rabbit se présente comme cela :

```
bool Rabbit::update()
{
    age++;
    /* Update maturity */
    if (!adult && age >= maturity)
    {
        adult = true;
        survival = 50;
        proba_death = (float)pow((float)survival / 100, (float)1 / (132 - maturity)) * 100;
    }

    /* Update anniversaire (porté + survavibilité) */
    else if (age % 12 == 0)
    {
        anniversary();
        if (age >= 11 * 12)
        {
            survival -= 10;
            proba_death = (float)pow((float)survival / 100, (float)1 / (132 - maturity)) * 100;
        }

    }
    return 0;
}
```

Figure 8: Définition de la fonction update() de la classe Rabbit

Un incrémente donc son age et vérifie qu'il atteint la maturité pour lui mettre à jour son booléen et ajuster son taux de survie ainsi que sa nouvelle probabilité de survie.
Dans ce programme les lapins jeunes n'ayant pas atteint la maturité ont 20 % de chance de de survivre durant toutes leur jeunesse et après ont 50 % de survie jusqu'à atteindre 11 ans une fois la maturité atteinte. Cette chance de survie diminue 10% d'année en année à partir de 11 ans pour atteindre 0% à 15 ans ce qui signifie que tous lapins meurent systématiquement à 15 ans au plus tard.

Si un lapin passe une nouvelle année et qu'il est adulte on va appeler la fonction anniversary() qui n'a de réelle impact que pour les femelles (cette fonction est vide pour la classe Rabbit)

```
void Female::anniversary()
{

    int i = 0;
    int mois_tmp;

    porte = distribution(generator);
    mois.clear();

    while (i < porte)
    {
        mois_tmp = repartition(generator);

        std::vector<int>::iterator p;
        p = std::find(mois.begin(), mois.end(), mois_tmp);

        if (p == mois.end())
        {
            mois.push_back(mois_tmp);
            i++;
        }
    }
}
```

Figure 9: Définition de la fonction anniversary() de la classe Female

La fonction anniversary() pour une lapine va simplement générer le nombre de portées qu'elle aura dans l'année ainsi que les moins durant lesquels elle donnera naissance.

La variable porte sera donc un nombre aléatoire entre 4 et 8 mais qui sera générer par une loi uniforme autour de 6 d'écart type pour essayer d'avoir plus de valeur autour de 5, 6 et 7 comme demandé dans l'énoncé. Certes ce n'est pas la manière optimum en effet avec cette méthode nous avons plus de chance d'avoir le chiffre 6 que 5 et 7 mais c'était une manière assez correcte pour approcher le plus l'énoncé.

Ensuite on stocke dans un vecteur mois les mois pendant lesquels la lapine fera ses portés en vérifiant bien sur de ne pas ajouter le même mois deux fois dans le vecteur.

Enfin pour finir la dernière partie de la fonction update() de la classe Population consiste à générer les portées si le mois courant correspond à un mois de la portée de la lapine.

```
int nb_baby = naissance(generator2);
for (int k = 0; k < nb_baby; k++)
{
    birth_tmp++;
    if (sexe(generator2))
    {
        population.push_back(new Female());
    }
    else
    {
        population.push_back(new Rabbit());
    }
}
```

Figure 10: Génération d'une portée d'une lapine

Ici on génère le nombre de bébés de la portée (un nombre aléatoire entre 3 et 6) et on génère également un chiffre entre 0 et 1 pour savoir si c'est un mâle ou une femelle. Une fois fois le choix du sexe fait on ajoute le bébé à la std::list de Population.

Une fois tous les lapins de la liste traités on retire tous les lapins morts de la liste et on sauvegarde les statistiques :

```
population.remove(nullptr);

/* On recupere les donnees */
nb_rabbit.push_back(population.size());
nb_birth.push_back(birth_tmp);
nb_death.push_back(death_tmp);
write_data(population.size(), birth_tmp, death_tmp);
birth_tmp = 0;
death_tmp = 0;
```

Figure 11: Suppréssion des lapins morts de la liste + sauvegarde des statistiques du mois courant

On fait ceci le nombre de mois passé en paramètre de update().

4 – Résultats :

Avant de parler des résultats il est utile de préciser que notre simulation une lapine peut donner naissance même si il n'y a pas de mâle chose qui n'arrive presque jamais dans les simulations faîtes sauf si l'on démarre avec un très petit nombre de mâle au mois 0.

Les graphiques sont obtenus grâce aux données que le programme sauvegarde dans result.txt :

```
void Population::write_data(int pop, int birth, int death)
{
    file << pop << ";" << birth << ";" << death << "\n";
}
```

Figure 12: Fonction permettant la sauvegarde des statistiques dans un fichier .txt

On stocke chaque mois la population actuelle, le nombre de naissance et le nombre de mort.

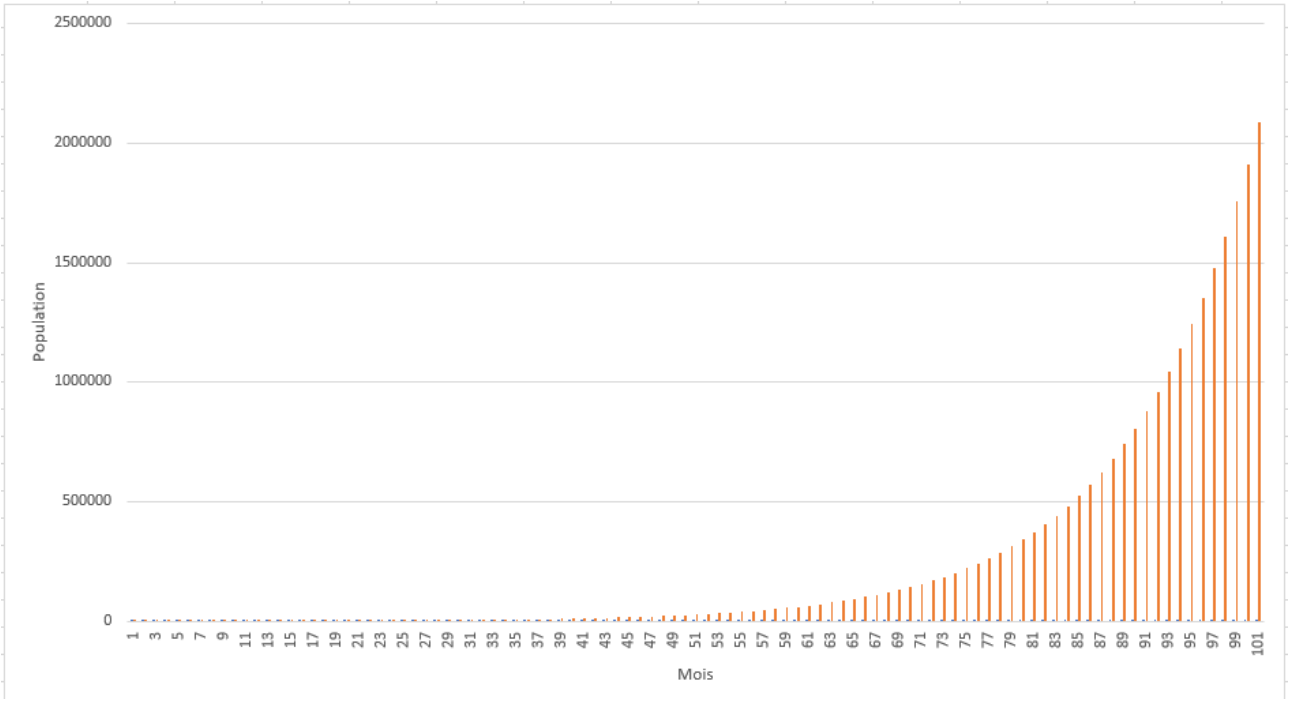

Figure 13: Simulation d'une population de lapins sur 100 mois en partant avec 1000 mâles et 1000 femelles

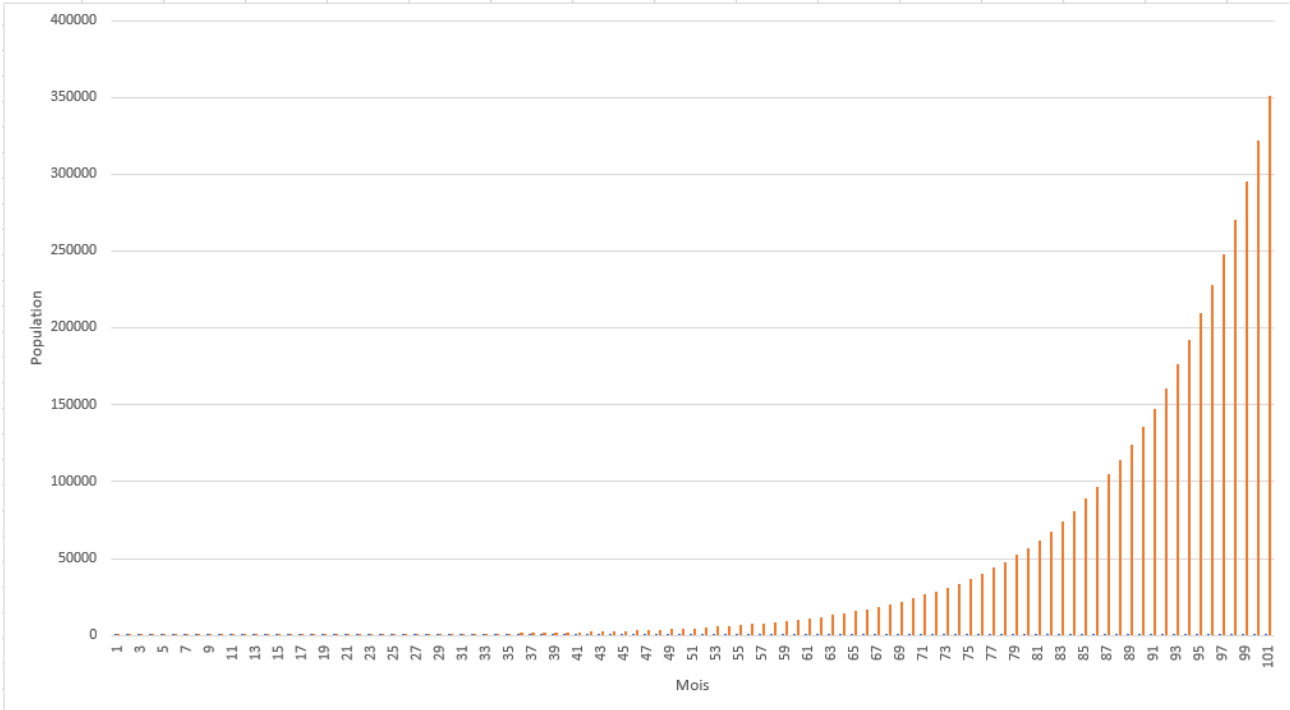

Figure 14: Simulation d'une population de lapins sur 100 mois en partant avec 1000 mâles et 100 femelles

Comme on peut le voir sur les histogrammes la croissance suit une sorte d'exponentielle.

On peut aussi observer la sortie console du programme avec d'autre paramètres comme le nombre de naissances, de mort et la population actuelle de chaque mois :

```
Simulateur de population de lapins :
Rentrer le nombre femelle, de male et la duree de la simulation (en mois)
svp : 1000 1000 20
Update de la population :
Nombre de lapins au mois 0 : 1482 --> Naissances : 0 --> Morts : 518
Nombre de lapins au mois 1 : 1116 --> Naissances : 0 --> Morts : 366
Nombre de lapins au mois 2 : 851 --> Naissances : 0 --> Morts : 265
Nombre de lapins au mois 3 : 651 --> Naissances : 8 --> Morts : 208
Nombre de lapins au mois 4 : 532 --> Naissances : 60 --> Morts : 179
Nombre de lapins au mois 5 : 555 --> Naissances : 177 --> Morts : 154
Nombre de lapins au mois 6 : 647 --> Naissances : 246 --> Morts : 154
Nombre de lapins au mois 7 : 791 --> Naissances : 317 --> Morts : 173
Nombre de lapins au mois 8 : 909 --> Naissances : 317 --> Morts : 199
Nombre de lapins au mois 9 : 993 --> Naissances : 311 --> Morts : 227
Nombre de lapins au mois 10 : 1105 --> Naissances : 344 --> Morts : 232
Nombre de lapins au mois 11 : 1169 --> Naissances : 354 --> Morts : 290
Nombre de lapins au mois 12 : 1043 --> Naissances : 91 --> Morts : 217
Nombre de lapins au mois 13 : 1224 --> Naissances : 475 --> Morts : 294
Nombre de lapins au mois 14 : 1432 --> Naissances : 556 --> Morts : 348
Nombre de lapins au mois 15 : 1708 --> Naissances : 643 --> Morts : 367
Nombre de lapins au mois 16 : 1885 --> Naissances : 614 --> Morts : 437
Nombre de lapins au mois 17 : 2063 --> Naissances : 708 --> Morts : 530
Nombre de lapins au mois 18 : 2219 --> Naissances : 695 --> Morts : 539
Nombre de lapins au mois 19 : 2444 --> Naissances : 792 --> Morts : 567
Taille finale de la population : 2444 / Nombre de morts totales : 6264
```

De plus il possible de demander au programme de réitérer la même simulation plusieurs fois pour avoir une moyenne de la population ainsi qu'un intervalle de confiance à 95 % (le code permettant de générer cet intervalle est presque le même que celui du TP3 c'est pourquoi le code n' est pas présent dans le rapport) :

```
Simulateur de population de lapins :
Souhaitez vous faire plusieurs simulations et obtenir une moyenne (YES)
ou simplement une seule detaillee (NO) ? Y / N
Y
Rentrer le nombre de simulations a faire, le nombre de femelle, de male
et la duree de la simulation (en mois) svp : 20 200 200 50
Population de la simulation 1 : 5202
Population de la simulation 2 : 5097
Population de la simulation 3 : 4660
Population de la simulation 4 : 5480
Population de la simulation 5 : 5520
Population de la simulation 6 : 5664
Population de la simulation 7 : 5756
Population de la simulation 8 : 5128
Population de la simulation 9 : 6052
Population de la simulation 10 : 4889
Population de la simulation 11 : 4647
Population de la simulation 12 : 3230
Population de la simulation 13 : 5018
Population de la simulation 14 : 5917
Population de la simulation 15 : 5060
Population de la simulation 16 : 6783
Population de la simulation 17 : 4297
Population de la simulation 18 : 5110
Population de la simulation 19 : 6009
Population de la simulation 20 : 6922
Moyenne des simulation --> Population moyenne : 5322.05 / Ecart type :
834.771
Interval de confiance à 95% : [4932.68 ; 5711.42]
```

Comme on peut le voir l'écart type est important est l'intervalle est grand ce qui est normal puisque les simulations sont aléatoires et dépendent énormément du nombre de femelles qui vont survivre spécialement lorsqu'elles sont encore jeunes et qu'elles n'ont que très peu de chance de survie.

5 – Annexes :

Le fichier result.txt contient la population le nombre de naissance et nombre de mort séparé par un ; on peut importer ses données sous un tableau pour en faire un graphique comme ceux précédemment montré.

Pour compiler le programme lancer la commande :

```
make
```

Pour lancer le programme lancer la commande :

```
./tp4
```

Une documentation est disponible dans html/index.html, dans la page qui s'ouvre aller dans le menu Files → Files List et naviguer à travers toutes les classes / fonctions du programme.