Simulation TP4
Simulation d'une population de lapins

Arquillière Mathieu

18 novembre 2019

Table des matières

Table des figures

1 Introduction

L'objectif de ce TP est de modéliser à l'aide de la programmation objet des lapins afin d'étudier le comportement d'une population. Pour la gestion de l'aléatoire, on utilisera la bibliothèque *random* fournie par la STL du c++. On utilisera le générateur *std : :mt19937* Mersenne Twister et les distributions uniformes et normales.

2 Analyse du sujet

Il s'agit de simuler le comportement d'une population de lapins sans prédateurs. Il y a plusieurs parties à comprendre et à implémenter.

2.1 Naissances

Le sujet nous indique qu'une femelle lapin peut donner 4 à 8 portées par an, avec une probabilité plus forte de faire 4, 5 ou 6 portées. De plus, chaque portée peut donner 3 à 6 bébés lapins avec une 50% de chance d'être un mâle et 50% de chance d'être une femelle pour chaque bébé lapin.

2.2 Majorité

Un lapin devient adulte au bout d'un certain nombre de mois, compris aléatoirement entre 5 et 8 mois. Cette majorité sert à la fois pour les naissances puisqu'une femelle ne peut enfanter qu'à partir de se majorité si il y a au moins 1 lapin mâle également adulte, mais aussi pour la probabilité de mourir.

2.3 Mort

Un lapin a 20% de chance de mourir lors de sa jeunesse (donc sur les 5 à 8 mois avant la majorité). Lorsqu'il atteint l'âge adulte, il a 50% de chance de mourir. On considérera que ce chiffre correspond à l'âge adulte en entier, c'est à dire 11 ans moins le nombre de mois de jeunesse. A partir de 11 ans il rentre dans l'âge "vieux" et cette probabilité de mourir augmente de 10% chaque année (fatalement, il meurt donc forcément à 15 ans).

3 Implémentation

De façon générale, on exécutera les tests en nombre de mois.

3.1 L'objet lapin

L'objet lapin représentera un lapin, quelque soit son genre. Il contient donc un attribut *age* et un attribut *majorité* en nombre de mois. Il contient également les probabilités de mourir jeune et en étant adulte puisque celles-ci sont calculées en fonction de la majorité. Pour ce qui est des méthodes, la classe a besoin de *grow()* qui sera appelée tous les mois pour faire grandir le lapin, elle a besoin d'une méthode permettant de savoir si le lapin doit mourir ce mois (*hasToDie()*) et enfin de 2 méthodes utilitaire, l'une pour savoir si le lapin est adulte (*hasMajority()*) et l'autre pour obtenir l'age (*getAge()*) qui sera surtout utilisée pour faire des statistiques. La classe contient aussi les probabilités de survie afin de faciliter le changement de paramètres pour exécuter des simulations différentes et éviter les *magic numbers*.

A la base, cette classe est destinée à être la classe mère réunissant les classes filles *lapin femelle* et *lapin mâle*. Cependant même si effectivement la classe du lapin femelle ajoute beaucoup de "fonctionnalités", le mâle n'a pas besoin d'une classe spécifique. Ainsi, une instance de la classe *Rabbit* correspond à un lapin mâle, et il faut créer une classe *FemaleRabbit* héritant de la classe *Rabbit* qui elle correspond à un lapin femelle.

FIGURE 1 – Déclaration de la classe *Rabbit*

```cpp
/**
 * @class Rabbit
 * @brief Classe représentant un lapin (mâle ou femelle)
 *
 * Si l'objet est créé en tant que Rabbit, c'est qu'il représente un lapin mâle, sinon la classe utilisée sera FemaleRabbit.
 * Cependant la classe Rabbit synthétise les deux genres. Il n'y a juste pas de classe MaleRabbit puisque celle-ci n'implémenterait rien de plus que la classe Rabbit.
 */
class Rabbit
{
public:
    Rabbit();
    virtual ~Rabbit();

    virtual void    grow();
    bool            hasMajority() const;
    bool            hasToDie() const;
    unsigned int    getAge() const;
    virtual bool    isMale() const;

    static constexpr double survival_proba_young = 0.2; //!< probabilité de survivre à la jeunesse (les 5 à 8 premiers mois)
    static constexpr double survival_proba_adult = 0.5; //!< probabilité de survivre pour un lapin adulte (11 ans moins les mois de jeunesse)

protected:
    unsigned int    _age; //!< L'age du lapin en mois
    unsigned int    _majority; //!< Nombre de mois avant que le lapin n'atteigne la majorité et passe à l'age adulte et puisse enfanter (si c'est une femelle)
    double          _proba_to_die_young; //!< Probabilité de mourir (par mois) avant de devenir adulte, calculée avec la probabilité de survivre et le nombre de mois de jeunesse
    double          _proba_to_die_adult; //!< Probabilité de mourir (par mois) en étant adulte, calculée avec la probabilité de survivre et le nombre de mois de jeunesse
};
```

La plupart des implémentations des méthodes de cette classe sont assez simples (*getAge()* renvoie l'age, *grow()* augmente l'age de 1). Le constructeur lui est un peu plus spécial. En effet c'est dans celui-ci qu'on décide de la majorité et donc les probilités de mort. Puisque notre unité de temps est le mois, on redéfinit les probabilités du sujet pour obtenir une probabilité par mois.

$$(P_{jeune})^{majorite} = 0.2 \Leftrightarrow P_{jeune} = 0.2^{majorite}$$

$$(P_{adulte})^{11ans-majorite} = 0.5 \Leftrightarrow P_{adulte} = 0.2^{132-majorite}$$

FIGURE 2 – Constructeur de la classe *Rabbit*

```cpp
/**
 * @brief Construit un nouvel objet "Rabbit"
 *
 * Définit l'age de la majorité, la probabilité de mourir jeune et la probabilité de mourir en étant adulte
 */
Rabbit::Rabbit() :
    _age(0),
    _majority(dis_int_5_8(generator)),
    _proba_to_die_young(std::pow(survival_proba_young, 1.0 / _majority)),
    _proba_to_die_adult(std::pow(survival_proba_adult, 1.0 / (132 - _majority)))
{
}
```

3.2 L'objet Lapin femelle

L'objet *FemaleRabbit* est une spécification de la classe *Rabbit*. En effet, elle rajoute la possibilité de donner naissance. Pour ça, elle a besoin d'une référence sur la liste contenant la population (pour y rajouter ses enfants) qu'on lui passe en paramètre lorsqu'on créer une instance.

FIGURE 3 – Déclaration de la classe *FemaleRabbit*

```
/**
 * @class FemaleRabbit
 * @brief Classe représentant un lapin femelle
 *
 */
class FemaleRabbit : public Rabbit
{
public:
    FemaleRabbit(std::list<Rabbit*> & pop);
    ~FemaleRabbit();

    virtual void grow();
    virtual bool isMale() const;

private:
    bool                    _monthsToBirth[12]; //!< représente les mois où la femelle doit enfanter, si monthToBirth[i] vrai -> la lapine doit enfanter au mois i
    std::list<Rabbit*> &    _whereToGiveChildren; //!< représente la population où la femelle doit y "ajouter" ses enfants
};
```

Elle redéfinit également la méthode *grow()*. Celle-ci permet ainsi de grandir (en appelant la méthode de *Rabbit*) mais aussi d'enfanter. Afin de simplifier le fait de donner des portées 4 à 8 fois par an avec plus de probabilités pour 5, 6 ou 7 mois, on décide 1 fois par an le nombre de portées que la femelle aura dans l'année avec une distribution normale.

FIGURE 4 – Exemple d'utilisation de distribution normale en C++ (voir cppreference)

```
#include <iostream>
#include <iomanip>
#include <string>
#include <map>
#include <random>
#include <cmath>
int main()
{
    std::random_device rd{};
    std::mt19937 gen{rd()};

    // values near the mean are the most likely
    // standard deviation affects the dispersion of generated values from the mean
    std::normal_distribution<> d{6,1};

    std::map<int, int> hist{};
    for(int n=0; n<10000; ++n) {
        ++hist[std::round(d(gen))];
    }
    for(auto p : hist) {
        std::cout << std::setw(2)
                  << p.first << ' ' << std::string(p.second/200, '*') << '\n';
    }
}
```

Possible output:

```
 2
 3
 4 ***
 5 ************
 6 ********************
 7 ***********
 8 **
 9
10
```

Une fois le nombre de portées décidé, on décide aléatoirement avec une distribution uniforme entre 0 et 11 les mois auquels la femelle donne une portée. On stocke ces informations avec un tableau de booléens correspondant au mois. Ainsi chaque mois (donc dans cette même fonction *grow()*), on teste si la femelle doit enfanter (*true* dans le tableau) et dans ce cas, on décide aléatoirement suivant une loi uniforme entre 3 et 6 le nombre de bébés de la portée.

FIGURE 5 – Méthode *grow()* de *FemaleRabbit*

```
/**
 * @brief Méthode faisant vieillir le lapin de 1 mois
 *
 * Appelle la méthode grow de Rabbit afin de grandir en tant que lapin
 * Si c'est l'anniversaire de la lapine, les mois auquels elle enfante dans l'année sont choisis
 * Si c'est un mois où elle doit enfanter, le nombre de bébés lapins dans la portée est décidé et elle rajoute ce nombre de nouveaux lapins à la population
 */
void FemaleRabbit::grow()
{
    Rabbit::grow();

    // Happy birthday!
    if(_age % 12 == 0)
    {
        // On décide combien de portées la lapine aura cette année
        int nbOfLitters = dis_normal_6_1(generator);

        std::memset(_monthsToBirth, false, 12);

        // On regarde si il y a au moins 1 male dans la population qui a la majorité et si la lapine a la majorité
        if (std::any_of(_whereToGiveChildren.begin(), _whereToGiveChildren.end(), [](Rabbit* r) { return r->isMale() && r->hasMajority(); }) && hasMajority())
        {
            // On décide à l'avance les mois auquel la lapine enfante
            for (int i = 0; i < nbOfLitters; i++)
            {
                int month = dis_int_0_11(generator);
                _monthsToBirth[month] = true;
            }
        }
    }

    // Si ce mois la lapine doit enfanter
    if(_monthsToBirth[_age % 12])
    {
        // On décide le nombre de bébés de la portée
        int nbOfChildren = dis_int_3_6(generator);
        for (int i = 0; i < nbOfChildren; i++)
        {
            // Nouvelle naissance -> 50% de chance d'avoir un male / 50% de chance d'avoir une femelle
            double random = dis_real_0_1(generator);
            _whereToGiveChildren.push_back(random < 0.5 ? new Rabbit() : new FemaleRabbit(_whereToGiveChildren));
        }

    }
}
```

3.3 L'objet Population

Maintenant, afin de gérer un ensemble de lapins et de les faire interargir, on créé un objet *Population*. On utilisera une liste (*std : :list*) de la STL afin de stocker des pointeurs sur des lapins. Une liste a l'avantage d'avoir l'insertion / suppression en *O(1)*. Cette classe démarrera avec 1 lapin mâle et 1 lapin femelle ou avec un nombre de lapins de chaque sexe donné. Ensuite elle se chargera de "faire passer le temps" aux lapins et de récuperer un certain nombre d'informations comme le nombre de morts ou de naissances.

FIGURE 6 – Déclaration de la classe *Population*

```cpp
/**
 * @class Population
 * @brief Représente une population de lapins
 *
 */
class Population
{
public:
    Population();
    Population(unsigned int numberOfMale, unsigned int numberOfFemale);
    ~Population();

    void passTime(unsigned int nbOfMonths);
    unsigned int getMonth() const;
    size_t getNumberOfRabbit() const;
    size_t getNumberOfFemaleRabbit() const;
    size_t getNumberOfMaleRabbit() const;
    size_t getDeath() const;
    size_t getBirth() const;
    double getMeanDeathAge() const;

    void exportToCSV(std::string const & filename);

private:
    std::list<Rabbit*> _population; //!< liste de lapins constituants la population
    unsigned int _month; //!< nombre de mois passé par la population
    unsigned int _nbDeath; //!< nombre de morts dans la population (depuis le début)
    unsigned int _nbBirth; //!< nombre de naissances dans la population (depuis le début)
    unsigned int _sumDeathAge; //!< somme des ages auquels meurent les lapins
};

std::ostream& operator<<(std::ostream& out, Population const & pop);
```

Afin de "faire passer le temps" aux lapins, on appelle la méthode *grow()* chaque mois pour chaque lapin, méthode qui peut ajouter des lapins dans la liste grâce aux femelles, et d'appeler la méthode *hasToDie()* pour tester si le mois passé fais mourir des lapins. Ces lapins sont détruits et retirés de la liste.

FIGURE 7 – Implémentation de la méthode *passTime*, boucle principale faisant grandir la population

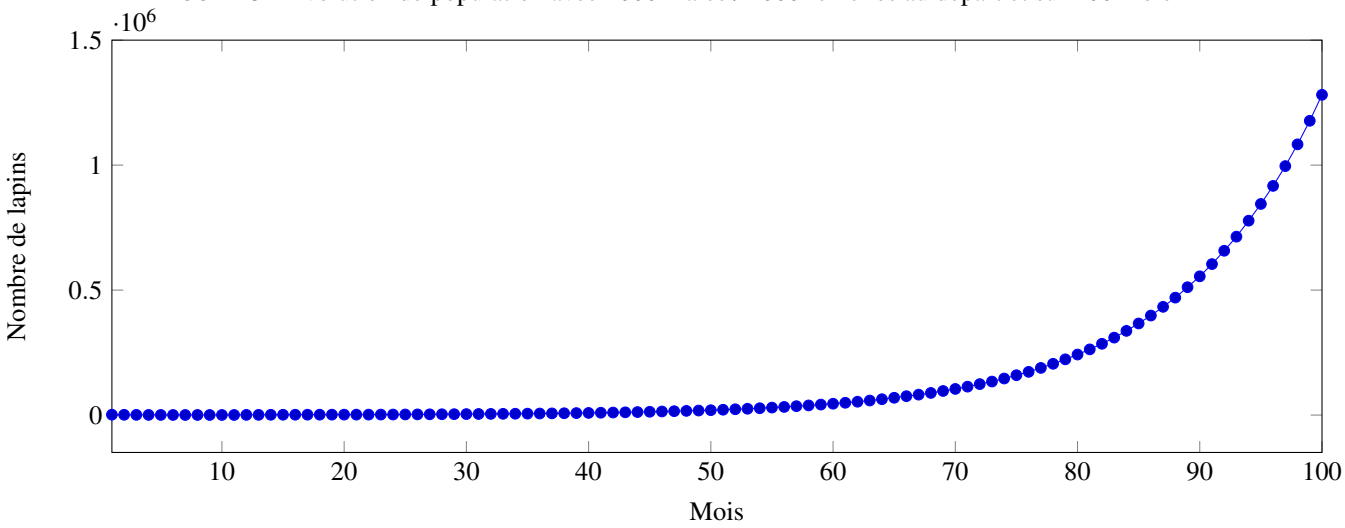
```cpp
/**
 * @brief passe le temps pour la population de lapins
 *
 * Passe un nombre de mois spécifié pour tous les lapins de la population
 * La méthode les fait grandir avec la fonction grow, et si des lapins meurent, les supprime de la liste
 *
 * @param[in] nbOfMonths nombre de mois que l'on veut passer
 */
void Population::passTime(unsigned int nbOfMonths)
{
    for (size_t i = 0; i < nbOfMonths; i++)
    {
        size_t sizeBeforeGrowth = _population.size();

        // On fait grandir tous les lapins de 1 mois
        std::for_each(_population.begin(), _population.end(),[](Rabbit* const & r)
        {
            r->grow();
        });

        // On compte le nombre de naissances de mois
        _nbBirth += _population.size() - sizeBeforeGrowth;

        // On regarde si des lapins meurent ce mois
        std::for_each(_population.begin(), _population.end(),[this](Rabbit*& r)
        {
            if (r->hasToDie())
            {
                _sumDeathAge += r->getAge();
                delete r;
                r = nullptr;
                // On compte le nombre de morts ce mois
                _nbDeath++;
            }
        });
        // On retire les lapins morts de la population
        _population.remove(nullptr);
    }
    _month += nbOfMonths;
}
```

4 Résultats

4.1 Tests simple d'une population

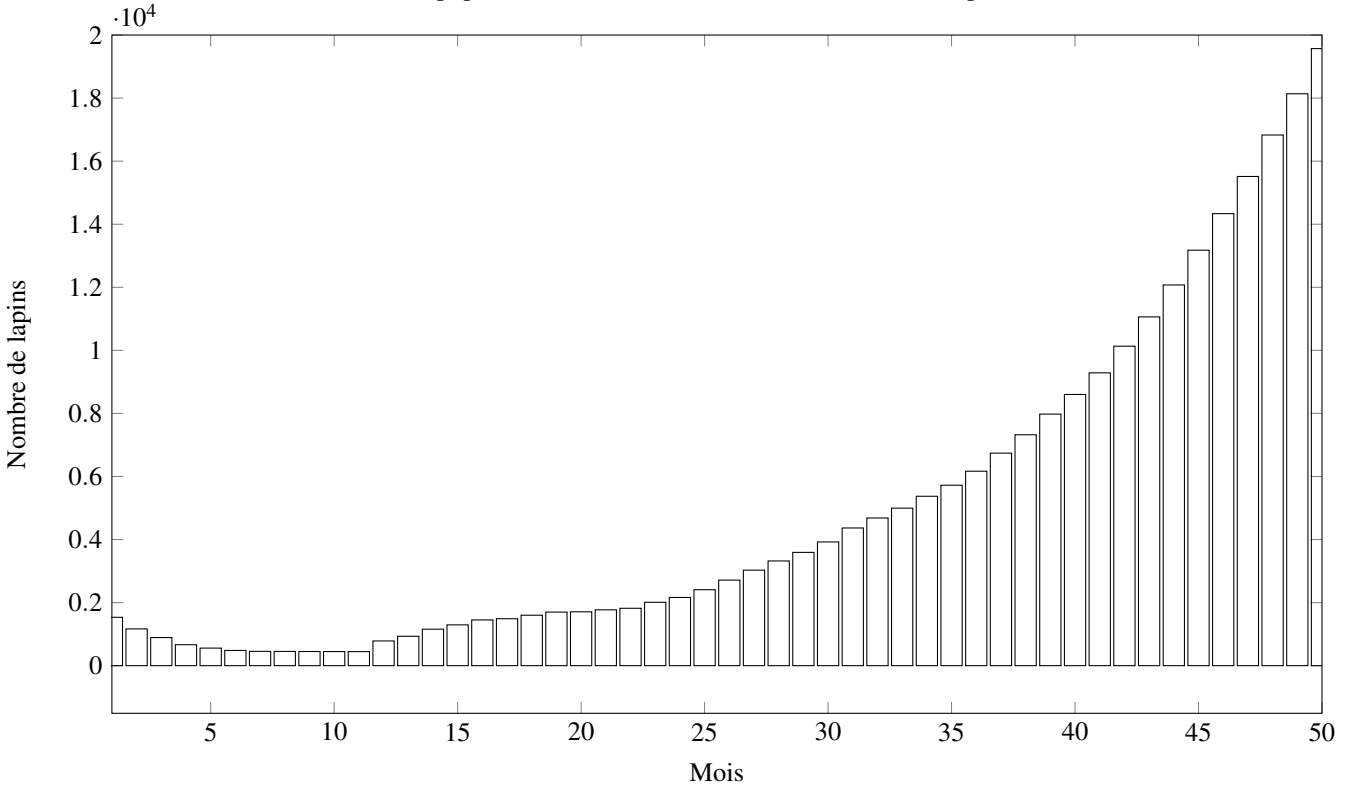
Si on observe l'évolution d'une population simple, prenons par exemple 2000 lapins au départ avec 1000 mâles et 1000 femelles, sur 100 mois, alors on observe qu'à partir d'un certain mois (aux environs de 40-50), la courbe démographique ressemble à une exponentielle.

FIGURE 8 – Evolution de population avec 1000 mâles / 1000 femelles au départ et sur 100 mois



Si on zoom un peu sur les 50 premiers mois, on obtient un graphique qui nous permet de voir que la population décroit un peu au début puis part en exponentielle. On peut sans doute expliquer cela du fait que les femelles ne peuvent pas enfanter tout de suite, puisqu'elles doivent atteindre la majorité. Ainsi dès que les premiers enfants arrivent dans la population, il y a plus de naissances que de morts, donc la population augmente de façon exponentielle.

FIGURE 9 – Evolution de population avec 1000 mâles / 1000 femelles au départ et sur 50 mois



4.2 Tests sur plusieurs populations

On peut également observer le nombre moyens de lapins dans une population au bout d'un certain nombre de mois. Le principe est simple, on a une boucle où l'on créé une population à chaque itération , on lui fait passer un certain nombre de mois et on garde les statistiques à la fin. Par exemple, on peut observer les moyennes pour 20 populations à 50 mois :

```
20 populations sur 50 mois:
Moyenne du nombre de lapins:            23241.8
Moyenne du nombre de mâles:             11607.5
Moyenne du nombre de femelles:          11634.4
Moyenne du nombre de morts:             45436.2
Moyenne du nombre de naissances:        68678
Moyenne des moyennes d age de mort:     2.6958
```

On observe que la répartition mâle / femelles est en effet de 50/50. On voit aussi qu'il y a plus de mort que de naissance. Voici un autre exemple avec 10 itérations de 100 mois :

```
10 populations sur 100 mois:
Moyenne du nombre de lapins:            1.496e+06
Moyenne du nombre de mâles:             747749
Moyenne du nombre de femelles:          748252
Moyenne du nombre de morts:             2.91845e+06
Moyenne du nombre de naissances:        4.41445e+06
Moyenne des moyennes d age de mort:     2.70848
```

5 Conclusion

Malheureusement, pour faire une serie de tests plus concluants, il faudrait regarder beaucoup plus de paramètre et exécuter des tests plus variés (le nombre de lapins au départ, le nombre de mois, les probabilités de mourir, ...). De plus les simulations sont assez limitées par le temps d'exécution du programme. En effet au delà de 120 mois, il y a trop de lapins à mettre à jour et l'implémentation prends trop de temps. Il est par contre possible de réflechir à d'autres implémentations. Par exemple, il sera possible de "pré-générer" les lapins lors de leurs naissance en choisissant toute sa vie à l'avance (ses dates d'accouchements, sa date de mort, ...) et ainsi éviter beaucoup de tests inutiles.

Il serait également intéressant de transformer l'implémentation actuelle avec des *templates* afin de choisir des conteneurs de populations différents (*list, forward-list, vector, deque*) et d'évaluer lesquelles sont les plus efficaces.

A Manuel d'utilisation

Le programme possède 2 sortes d'utilisation, une avec 1 population et une autre en en faisant plusieurs.

A.1 Gestion d'une seule population

Si on veut regarder l'évolution d'une population sur 50 mois

```
./lapin.exe
```

Si on veut préciser le nom du fichier de sortie

```
./lapin.exe fichierSortie.txt
```

Si on veut préciser le nombre de mois à passer (il faut obligatoirement un nom de fichier si on veut indiquer ce nombre)

```
./lapin.exe fichierSortie.txt 100
```

A.2 Moyenne sur plusieurs populations

Si on veut observer les statistiques de plusieurs évolutions de populations (par défaut 10 populations sur 50 mois)

```
./lapin.exe -s
```

Si on préciser le nombre de populations qu'on veut observer

```
./lapin.exe -s 20
```

Si on préciser le nombre de mois qu'on passe pour chaque population (il faut obligatoirement mettre le nombre de populations aussi)

```
./lapin.exe -s 20 100
```

B Compilation

Un makefile a été utilisé pour simplifier la compilation de ce TP. Voici son contenu

```
SRC=$(wildcard *.cpp)
EXE=lapin.exe

CXXFLAGS+=-Wall -Wextra -MMD -O2 -std=c++17 -pg
LDFLAGS=

OBJ=$(addprefix build/,$(SRC:.cpp=.o))
DEP=$(addprefix build/,$(SRC:.cpp=.d))

all: $(OBJ)
  $(CXX) -o $(EXE) $^ $(LDFLAGS)

build/%.o: %.cpp
  @mkdir -p build
  $(CXX) $(CXXFLAGS) -o $@ -c $<

clean:
  rm -rf build core *.gch

-include $(DEP)
```

On a seulement besoin de la STL du c++. Notamment de la bibliothèque <*random*> pour générer tous les nombre pseudo-aléatoires du programme, de la bibliothèque <*list*> et de <*algorithm*> afin de gérer les populations.

C Documentation

Lien vers la documentation Doxygen