Compte Rendu TP4
Simulation d'une population de lapin

Jérémy ZANGLA

16 novembre 2019

Table des matières

1 Analyse

1.1 Rappel du sujet

Le but de ce TP est de simuler l'évolution d'une population de lapin au cours du temps. La population ne sera affectée par aucun élément extérieur. Les seuls facteurs influants l'évolution seront les naissances et morts naturelles. Ainsi, les données que nous implémenterons seront les suivantes :

— Une lapine peut donner vie à une portée de lapins tous les 1 mois (temps de gestation).
— Une lapine aura de 4 à 8 portée par ans.
— Un laperau aura 20% de chances de survivre avant sa maturité.
— Un lapin aura 50% de chance de survie au dela de ses 11 ans.
— Pour chaque année suivante les lapins auront 10% de chances de mourir supplémentaire.
— Ainsi aucun lapin ne pourra vivre plus de 15 ans.

Enfin le travail devra être fait en programmation objet dans le langage C++.

1.2 Choix d'implémentation

1.2.1 Génération de nombres aléatoires

Comme pour les travaux précédents, nous allons utiliser l'algorithme de Mersenne-Twister pour la génération des nombres pseudos aléatoires. Nous utiliserons son implémentation dans la libraire standard du C++ (bibliothèque : random).
Cependant les nombres pseudo-aléatoires ne seront pas utilisés directement, contrairement aux travaux en C. Nous utiliserons les distributions de libraire standard. Nous parlerons par la suite de nombres aléatoires pour alléger le discours, il faut bien garder en tête que tous les nombres seront générés à partir de ce générateur pseudo-aléatoire.

1.3 Définition du temps

Nous ne pouvions ici pas faire une simulation en temps réelle de l'évolution de la population, pour une raison simple : le but est d'étudier des résultats obtenus après plusieurs mois ou années. Nous avons donc décidé d'accélérer le temps en intégrant un intervalle de temps. Cet intervalle correspond à la durée simulée entre deux mises à jours du code. Cet intervalle a été choisi à 1 semaine. Nous obtenons donc une simulation "précise" à la semaine près.

1.4 Fonctionnement des mises à jours

Au lieu de faire fonctionner notre simulation au fur et à mesure de son évolution, le choix a été fait de déterminer toute la vie d'un lapin dès sa naissance. Ainsi à la naissance d'un lapin, la semaine à laquelle il mourra sera décidée. De plus, pour une femelle, la date ainsi que les spécificités de chaque portée qu'elle aura seront fixée au même moment. Il y aura tout de même une unique vérification en "temps réel", c'est la présence ou non d'un lapin mâle pour la reproduction.

1.5 Déroulement des mises à jours et lancement d'une simulation

Chaque mise à jour se fera de la manière suivante, traitement de la semaine actuelle : naissances puis morts des lapins. On passera ensuite à la semaine suivante. Une fois la simulation créée, on doit pourra la lancer pour une durée choisis et la reprendre autant de fois que désirée.

1.6 Performances

Le choix de la déterminisation de la vie d'un lapin à sa naissance, devrait permettre d'améliorer les performances du simulateur en simplifant les calculs, et en évitant d'en faire certains. L'espace mémoire occupée par la simulation peut être un point sensible, cependant nous avons décidés ici de ne pas s'en occuper.

2 Implémentation

2.1 Nombres aléatoires

2.1.1 Distributions de nombres aléatoires

Une distribution est un objet implémenté dans la librairie standard (bibliothèque random), qui d'obtenir un nombre aléatoire qui suit une loi de probabilité. Nous en utiliserons ici 4 types en expliquant leur intérêt. Nous utilisons l'implémentation standard du générateur de Mersenne-Twister pour la génération de nombres pseudo-aléatoires.

Rabbit.cpp

```
1  std::mt19937 Rabbit::generator(8);
```

2.1.2 Distribution uniforme

La distribution uniforme est la plus simple, elle nous permet d'obtenir un nombre aléatoire compris dans un intervalle suivant une loi de probabilité uniforme, c'est à dire que chaque tirage est équiprobable. Il existe deux versions de cette distribution : réelle et entière, comme on pourrait le supposer la première permet la génération d'un nombre réelle et la seconde d'un nombre entier.

La première utilisée permet de choisir le sexe d'un lapin à naître lors de la génération des portées d'une lapine. C'est donc une distribution uniforme de nombres entiers avec deux possibilitées : 1 et 2. Ces possibilités correspondent respectivement à Mâle et Femelle.

RabbitFemale.hpp

```
1      // Distribution générant le sexe de chaque laperau
2      static std::uniform_int_distribution<short> dist_gender;
```

RabbitFemale.cpp

```
1  std::uniform_int_distribution<short> RabbitFemale::dist_gender(1, 2);
```

La seconde est une distribution de nombres réels dans l'intervalle $[0\,;1]$. Le nombre obtenu est très facilement comparable à un pourcentage. On l'utiliser pour décidé à quel période de sa vie le lapin va mourir (avant la maturité ou non, puis avant 11 ans ou non).

Rabbit.hpp

```
1      // Distribution permettant de faire un nombre aléatoire
2      // de type pourcentage
3      static std::uniform_real_distribution<> dist_rand;
```

Rabbit.cpp

```
1  std::uniform_real_distribution<double> Rabbit::dist_rand(0, 1);
```

La dernière sert à calculer la date de maturité d'un laperau, donc entre 4 et 8 mois (converti en semaine).

Rabbit.hpp

```
1      // Age en semaine à partir duquel un lapin peut devenir mature
2      static constexpr unsigned int MATURITY_MIN = 4 * 4;
3      // Age en semaine jusqu'auquel un lapin peut devenir mature
4      static constexpr unsigned int MATURITY_MAX = 8 * 4;
```

Rabbit.hpp

```
1       // Distribution générant la date de maturité du lapin
2       static std::uniform_int_distribution<> dist_maturity;
```

Rabbit.cpp

```
1  std::uniform_int_distribution<>
2       Rabbit::dist_maturity(MATURITY_MIN, MATURITY_MAX);
```

2.1.3 Distribution linéaire

Une distribution linéaire, permet d'avoir une densité de probabilité qui suit une fonction linéaire (ou un ensemble de fonctions linéaires). Nous n'en utilisons qu'une seule, qui n'utilise qu'une seule pente. L'implémentation dans la librairie standard utilise la méthode de Piecewise et a besoin de 2 tableaux pour le paramétrage. Le premier contient les intervalles des pentes, et le second contient les poids de chaque points, pour définir l'inclinaison de la pente.

Nous avons décidé d'utiliser cette distribution pour simuler l'accroissement de la mortalité à partir de 11 ans. Cette accroissement étant de 10% par an, il est linéaire.

Rabbit.hpp

```
1       // Variable d'initialisation d'une distribution
2       // linéaire de piecewise
3       static std::array<double,3> intervals;
4       static std::array<double,3> weights;
```

Rabbit.hpp

```
1       // Distribution générant la date de mort d'un papy lapin
2       static std::piecewise_linear_distribution<double> dist_death_end;
```

Rabbit.cpp

```
1  std::array<double, 3> Rabbit::intervals
2       {14 * WEEK_PER_YEAR, 14 * WEEK_PER_YEAR, MAX_LIFESPAN};
3  std::array<double, 3> Rabbit::weights {0, 0, 1};
```

Rabbit.cpp

```
1  std::piecewise_linear_distribution<double> Rabbit::dist_death_end
2       (Rabbit::intervals.begin(),
3       Rabbit::intervals.end(),
4       Rabbit::weights.begin());
```

2.1.4 Distribution géométrique

2.1.5 Distribution normale