



Projet informatique

Compte rendu - TP 4

Dans ce dossier, nous allons vous présenter deux méthodes pour simuler la croissance d'une population de lapins. L'une utilisant la suite de Fibonacci et l'autre utilisant le générateur de type Mersenne Twister implémenté par Makoto Matsumoto.

En effet, il n'est pas juste important d'observer le monde qui nous entoure, il faut aussi savoir comment il va évoluer de manière à anticiper les événements. La deuxième simulation présentée ici utilise des paramètres et des événements réalistes de manière à reproduire une situation qui pourrait exister. Ce type de techniques de simulations est également très utile dans le domaine de la biologie (évolution d'une tumeur, épidémie, ...) ou de la climatologie.



Code et rapport rédigé par :

BOURSAT Vincent
CORCOS Ludovic
L2 Informatique
Université Clermont-Auvergne



Table des matières

I) Notre organisation	2
a. Ce que nous espérions :	3
b. Ce qu'il s'est réellement produit :	4
II) Une simulation peu réaliste	5
a. Simulation de lapins utilisant la suite de Fibonacci.....	5
b. Présentation de l'implémentation	6
i. Forme récursive.....	6
ii. Forme impérative	6
iii. Comparaison des deux versions	7
III) Une simulation beaucoup plus réaliste.....	8
a. Scénario et contexte.....	8
b. Présentation des différentes fonctions et choix d'implémentation	9
i. Choix d'implantation	9
ii. Fonction : Nombre de portées	10
iii. Fonction : nombre de lapins par portées	10
iv. Fonction : sexe du lapin.....	11
v. Fonction : mortalité bébé et adulte lapin	12
c. Résultats de la simulation	13
d. Comparaison avec la simulation par la suite de Fibonacci.....	16
IV) Pour aller plus loin.....	17
ANNEXES	19




I) Notre organisation

Vu que ce dossier, et par conséquent ce programme était à faire en binôme, il était important d'être bien organisé dès le début. **C'est pour cela que nous avons créé un dossier partagé** sur Dropbox qui nous a permis d'échanger presque en temps réel nos différentes avancées. Nous avons dès le début créé une liste de tâches à faire. Cette liste regroupe les différentes fonctions du programme, mais aussi tout ce qui peut être **recherches auxiliaires** pour optimiser le programme et ses algorithmes. A cette liste, nous avons ajouté un diagramme de Gant, permettant de suivre notre évolution temporelle sur le projet.

Nous avons été déçus. En effet, nous avons, d'une certaine manière, été trop présomptueux sur nos capacités à réaliser un tel projet.

Voici la liste des tâches que nous souhaitons réaliser :

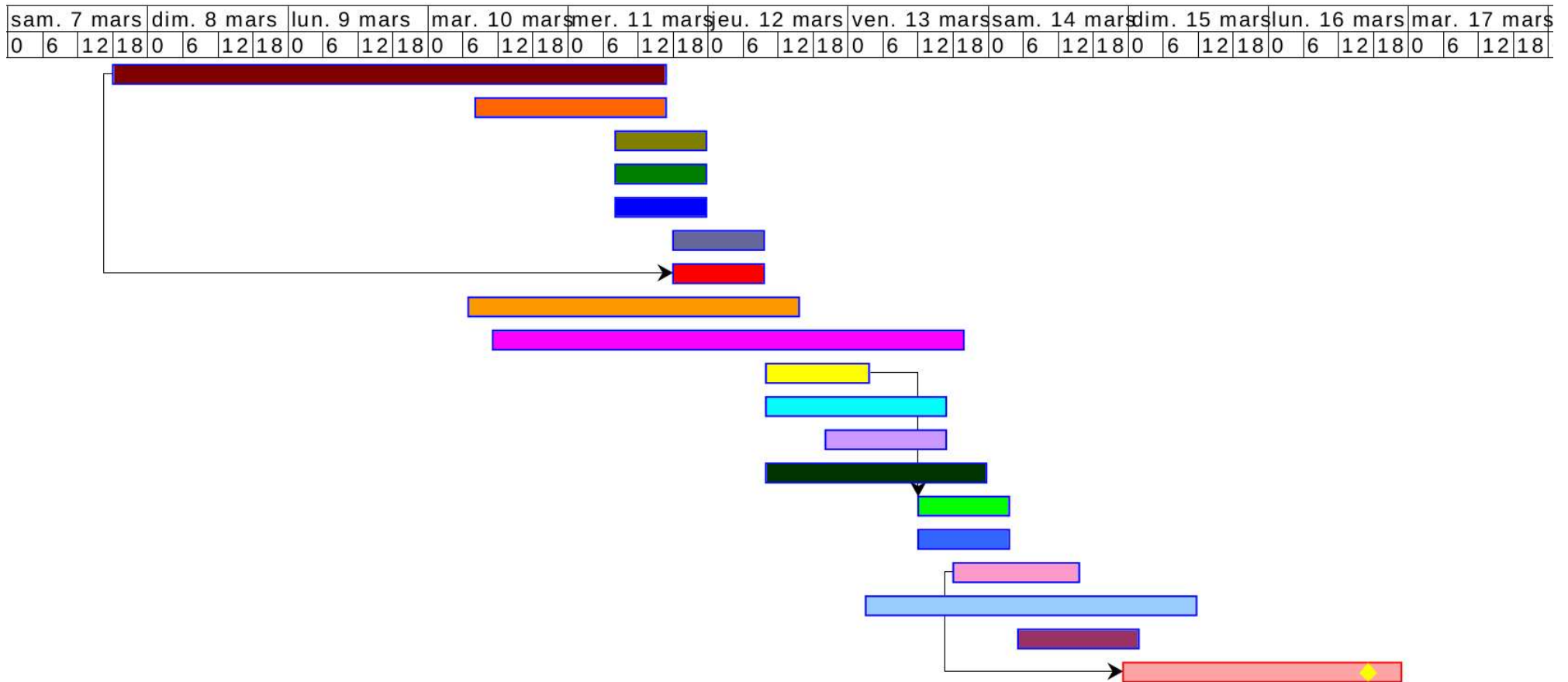
- Modélisation du projet
 - Choix de la méthode d'implémentation
 - Modélisation processus des naissances
 - Modélisation des portées
 - Modélisation de la maturité sexuelle
 - Modélisation de la mortalité
 - Incrustations de paramètres de mortalité différents
 - Recherche de données sur d'autres espèces
 - Optimisation des systèmes de calcul
 - Modélisation avec les autres espèces
 - Création d'un système de stockage des données générées
 - Modélisation de règles d'analyses statistiques
 - Création d'une interface en python
 - Mise en place du système de communication des algorithmes
 - Implémentation du système de lecture de données (python)
 - Application de l'interface graphique sur les codes de simulation
 - Résolutions des bugs et vérification
 - Exploitation des résultats
 - Rédaction du dossier
- 

Nous avons donné un code couleur sur ces tâches de manière que vous puissiez bien les visualiser sur le **diagramme de Gant** ci-après.

Les segments barrés concernent les tâches où le temps imparti était trop court pour que nous puissions les réaliser.

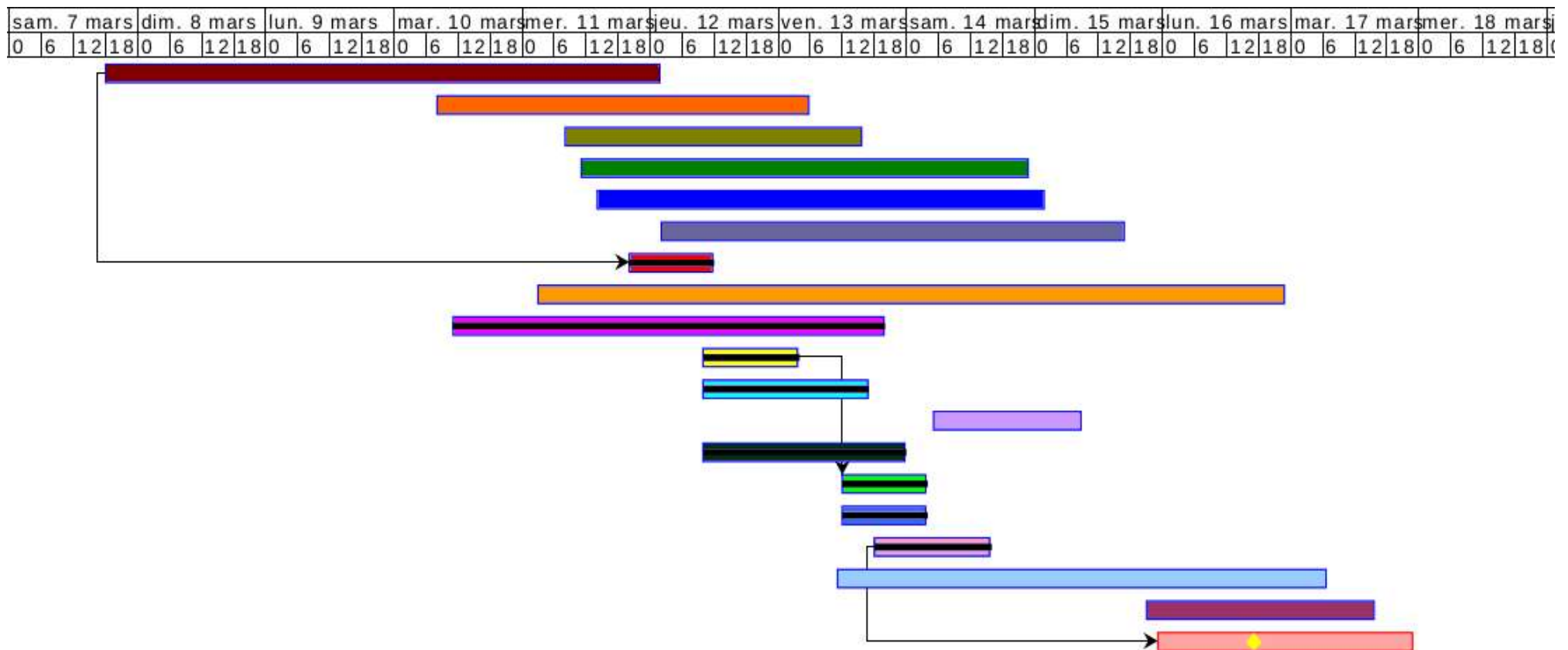


a. Ce que nous espérons :





b. Ce qu'il s'est réellement produit :





II) Une simulation peu réaliste

Le monde est fait d'histoires, de légendes et de mythes : il y a toujours une histoire ou une légende derrière chaque découverte faite par les hommes. On va maintenant s'intéresser à celle de **Léonardo Fibonacci** en attendant de voir la suite sur une simulation plus complète.

a. Simulation de lapins utilisant la suite de Fibonacci

Supposons que chaque mois, un couple de lapins donne naissance à un autre couple de lapin (**on mettra la consanguinité de côté, et on supposera que les lapins sont immortels**). En commençant par un couple, le mois suivant, il y en aura 2 puis 4, puis 8. En fait, **au mois n , ils seront 2^n** . Dans ce cas, la **progression est exponentielle**.

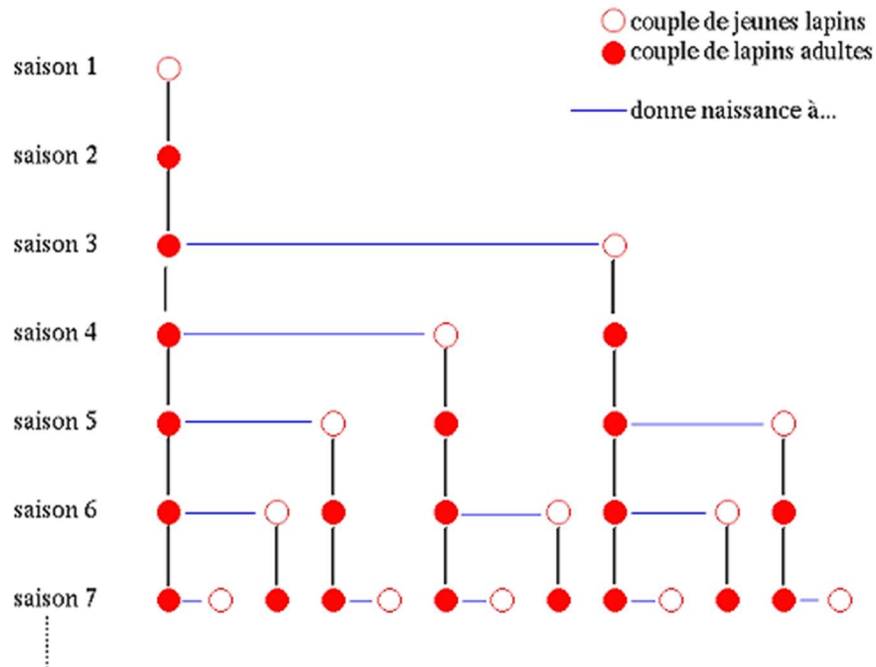
Mais un lapin pour pouvoir procréer doit être adulte.

Selon la suite de Fibonacci, on aura les règles suivantes :

- Un couple adulte donne naissance à un couple de lapereaux tous les mois.
- Un couple de lapereaux doit attendre un mois avant d'atteindre sa maturité et, adulte pour se mettre à procréer tous les mois.

Une question se pose donc, combien aurons-nous de couples de lapins selon le mois de l'année ?

C'est cette évolution que permet de prédire la suite de Fibonacci comme le montre l'illustration ci-contre.



Nous avons donc implémenté ce type de simulation en C, le code est disponible en annexe.



b. Présentation de l'implémentation

La suite de Fibonacci est définie mathématiquement comme suit :

$$U_{n+1} = U_n + U_{n-1}$$

Comme son nom l'indique, il s'agit d'une suite, donc **deux implémentations sont possibles**, voyons lesquelles.

i. Forme récursive

```
1. unsigned long long fibo1(int mois)
2. {
3.     if (mois <= 1)
4.     {
5.         return mois;
6.     }
7.     else
8.     {
9.         return fibo1(mois - 1) + fibo1(mois - 2);
10.    }
11. }
```

Aussi bien dans la forme récursive que dans la forme impérative, nous utilisons le type **unsigned long long**, nous verrons pourquoi plus tard dans ce dossier.

Cette fonction prend en entrée le nombre de mois à générer et retourne le nombre de lapins correspondant au mois donné.

ii. Forme impérative

```
1. unsigned long long fibo2(int mois)
2. {
3.     unsigned long long terme_suivant;
4.     unsigned long long premier_terme = 0;
5.     unsigned long long deuxieme_terme = 1;
6.     int i;
7.
8.     printf("%lld\n", premier_terme);
9.     for (i = 1; i <= mois; i++)
10.    {
11.        terme_suivant = premier_terme + deuxieme_terme;
12.        premier_terme = deuxieme_terme;
13.        deuxieme_terme = terme_suivant;
14.        printf("%lld\n", premier_terme);
15.    }
16.    printf("\n");
17.    return 0;
18. }
```

Cette fonction prend en entrée le nombre de mois à générer et affiche une liste correspondant au nombre de lapins par mois.

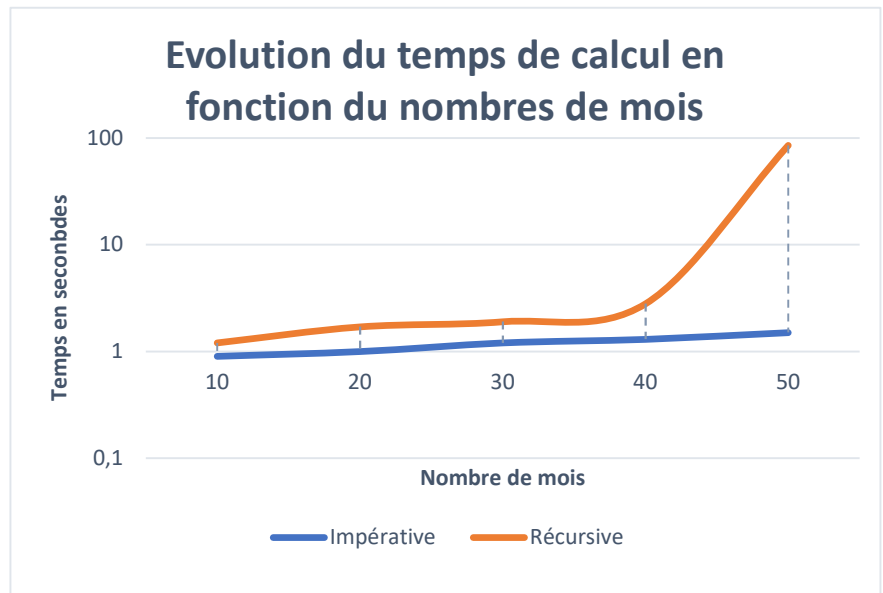


iii. Comparaison des deux versions

En sortie de la suite, on obtient : 0 - 1 - 1 - 2 - 3 - 5 - 8 - 13 - 21 - 34 - 55 - 89 - 144 - 233 - 377 - 610 - 987 - 1597 - 2584 - 4181 - 6765 - ... et ainsi de suite...

Mais on s'aperçoit vite que les choses ne se déroulent pas aussi bien ! Et que la **simulation commence à devenir de plus en plus longue** pour...

... le nombre de mois simulés. En effet, comme nous pouvons le constater sur le graphique ci-contre, **jusqu'à 40 mois, le temps de simulation est acceptable** mais au-delà, le temps devient vraiment trop important, c'est pour cela qu'il faut préférer l'algorithme impératif.



Pour revenir à la simulation des lapins :

Un des gros points faibles de la suite de Fibonacci pour la génération de lapins est leur invincibilité. En effet puisqu'ils ne peuvent pas mourir leur nombre ne peut que croître de **façon exponentielle**. Mais également, le fait qu'un couple de lapins ait forcément une portée par mois seulement 1 mois après leur naissance (ils sont mature sexuellement après un mois, et non 1 an), donnant ainsi naissance à un nouveau couple de lapins.

Soit, chaque couple donne naissance à 11 nouveaux couples durant leur première année puis 12 durant tout le reste de leur vie infinie (22 lapins la première année puis 24 par an).

Ce qui est très loin des statistiques de 4 à 8 portées par an mais sachant qu'il y'a 3 à 6 lapins par portée on s'en rapproche quand même.

Donc la suite de Fibonacci est loin d'être une bonne simulation cependant, elle peut nous donner un ordre d'idée sur la population de lapins au bout de quelques années.



III) Une simulation beaucoup plus réaliste

a. Scénario et contexte

Pour simuler cette colonie de lapins sur plusieurs années avec une certaine précision, on a **besoin de connaître quelques paramètres statistiques relatifs à l'évolution de nos lapins**. Nous cherchons donc à connaître la population de lapins dans une zone propice à l'environnement (**une grande île, Okunoshima, située dans la mer intérieure du Japon**) sans trop de prédateurs et avec des conditions favorables de température, d'hygrométrie et de nourriture. Pendant la seconde guerre mondiale, un couple de paysans a réussi à mettre sur cette île **10 lapins mâles et 10 lapins femelles**, tous âgés de 5 ans. Mais malheureusement, le couple est décédé et les lapins ont évolué d'eux-mêmes, seules des statistiques sur leurs évolutions ont été rapportées. Les voici :

- Une femelle peut donner naissance tous les mois, **entre 4 et 8 portées par année (mais il y a plus de chance d'en obtenir entre 5 et 7)**.
- Chaque portée possède une **équiprobabilité** d'obtenir 3 à 6 lapins.
- Chaque lapin a autant de chance d'être un mâle qu'une femelle (sex-ratio).
- Les lapins deviennent **matures sexuellement entre 5 et 8 mois** après leur naissance.
- Les lapereaux de moins de 1 an ont seulement 12 % de chance de survie pour l'année suivante. Ils sont **particulièrement sensibles au virus de l'entérite à Rotavirus** et de ce fait, une grande proportion de bébés lapins meurent.
- Les lapins adultes (c'est-à-dire de plus de 1 an) ont eu, 60 % de chances de survie pour l'année suivante, **mais à partir de 10 ans, leur chance de survie diminue de 10 % tous les ans**. De telle sorte que leur âge maximum est de 15 ans.

Nous sommes donc tombés sur ces informations, et vu qu'actuellement, nous **sommes dans l'incapacité de nous déplacer sur cette île ou dans le secteur pour en savoir plus** sur leur population, (pandémie du coronavirus COVID-19 oblige !), nous avons donc décidé de lancer une simulation sur 20 ans pour obtenir un ordre d'idée sur le nombre de lapins présents sur île.





b. Présentation des différentes fonctions et choix d'implémentation

i. Choix d'implantation

Pour implémenter cette simulation, nous avons fait le choix d'utiliser un **tableau à trois dimensions** qui serait utilisé de la manière suivante : [année] [ligne] [âge].

20	Âge	0	1	2	...	15
	Nombre de					
...	Âge	0	1	2	...	15
	Nombre de					
1	Âge	0	1	2	...	15
	Nombre de					
0	Âge	0	1	2	...	15
	Nombre de					
	Âge	0	1	2	...	15
	Nombre de femelles					
	Nombre de femelles mortes					
	Nombre de mâles					
	Nombre de mâles morts					

De cette manière, il est **facile de connaître et d'obtenir toutes les données nécessaires à la simulation, année par année sur les 4 données qui nous intéressent**, le nombre de femelles vivantes et mortes en fonction de leur âge et le nombre de mâles vivants et morts en fonction de leur âge.

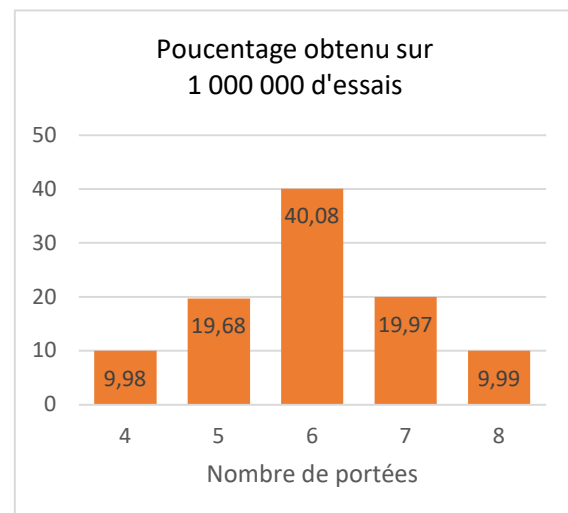
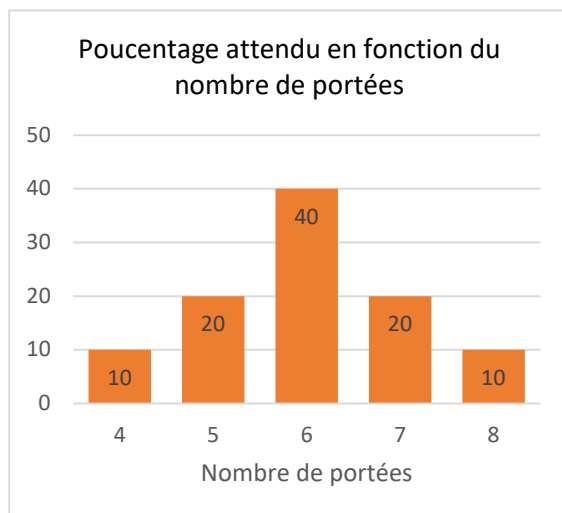
Découvrons sans plus attendre les différentes fonctions qui nous ont permis de modéliser la simulation.



ii. Fonction : Nombre de portées

```
1. int nbPortee()  
2. {  
3.  
4.     int i;  
5.     double valGene = genrand_real1();  
6.     double pourcentage[5] = {0.1, 0.3, 0.7, 0.9, 1.0};  
7.  
8.     for (i = 0; i <= 4; i++)  
9.     {  
10.  
11.         if (valGene <= pourcentage[i])  
12.         {  
13.             return (4 + i);  
14.         }  
15.     }  
16.  
17.     return EXIT_SUCCESS;  
18. }
```

Sur cette fonction, nous nous attendons à obtenir **4 à 8 protéés par année** mais avec une **chance plus importante d'en obtenir entre 5 et 7**. On peut donc construire les histogrammes, l'un montrant la répartition non homogène attendue et l'autre, ce qui a été obtenu.



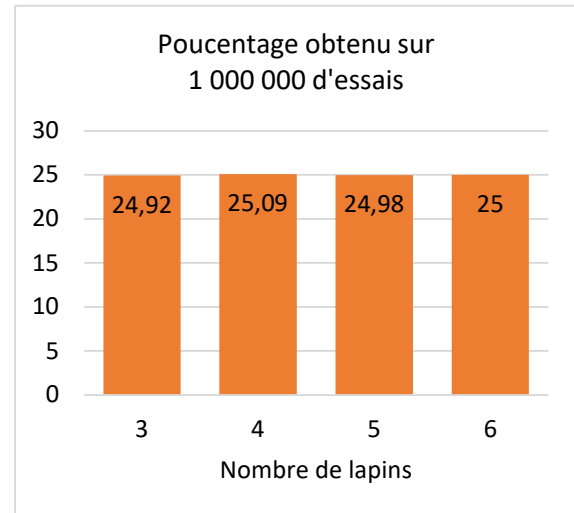
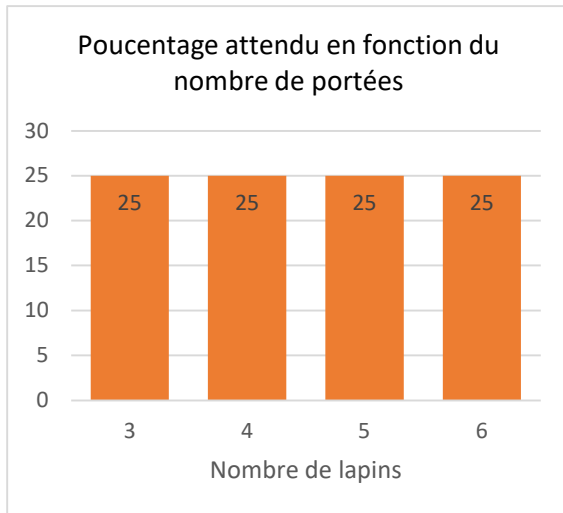
On constate que nos valeurs sont très proches de ce que l'on attendait statistiquement.

iii. Fonction : nombre de lapins par portées

```
1. int nbLapinPortee()  
2. {  
3.  
4.     int x = (int)(Uniform(2.0, 6.0) + 1);  
5.  
6.     return x;  
7. }
```



Cette fonction permet de générer de **manière homogène des nombres compris entre 3 et 6 de manière uniforme**. Il y a une équiprobabilité d'obtenir le nombre 3, 4, 5 ou 6. On peut donc construire les histogrammes, l'un montrant la répartition non homogène attendue et l'autre, ce qui a été obtenu.

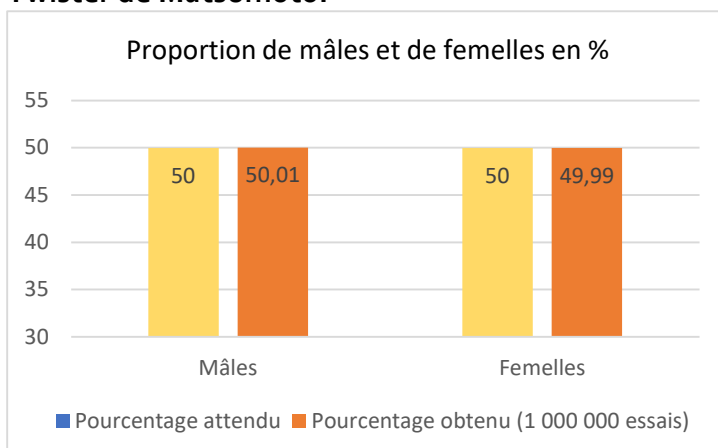


On constate que nos valeurs sont très proches de ce que l'on attendait statistiquement, on respecte parfaitement l'équiprobabilité.

iv. Fonction : sexe du lapin

```
1. int Sexelapin()  
2. {  
3.  
4.     double val = genrand_real1();  
5.     if (val <= 0.5)  
6.     {  
7.         return 0;  
8.     }  
9.     else  
10.    {  
11.        return 1;  
12.    }  
13. }
```

Cette fonction permet de déterminer le sexe du lapin, soit mâle soit femelle avec 50 % de chance pour l'un ou l'autre. Il utilise directement la **fonction genrand_real1() du Mersenne Twister de Matsumoto**.



On a également, dans ce cas, une bonne fidélité vis-à-vis des critères scénaristiques fixées plus haut.



v. Fonction : mortalité bébé et adulte lapin

Fonction pour la mortalité des bébés

```
1. int MortPetit()
2. {
3.
4.     double val_aleatoire = genrand_real1();
5.     int val_retour = 0;
6.     if (val_aleatoire >= 0.12)
7.     {
8.         val_retour++;
9.     }
10.
11.     return val_retour;
12. }
```

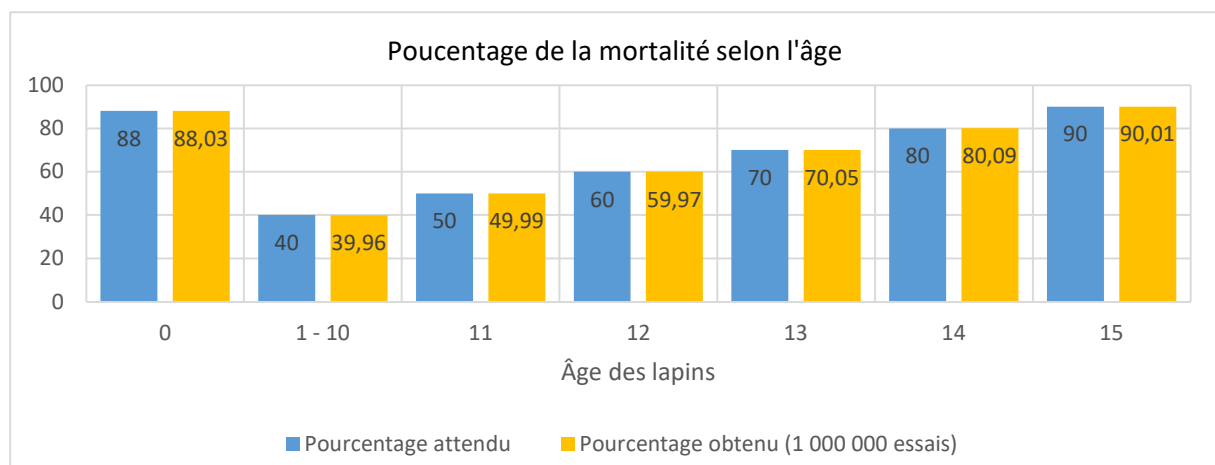
Fonction pour la mortalité des adultes

```
1. int MortAdulte(double decroissance)
2. {
3.
4.     double val_aleatoire = genrand_real1();
5.     int val_retour = 0;
6.
7.     if (val_aleatoire >= (0.60 - decroissance))
8.     {
9.         val_retour++;
10.    }
11.
12.    return val_retour;
13. }
```

Ces deux fonctions sont **utilisées dans la « grande » fonction de mortalité permettant de remplir les tableaux au fur et à mesure de l'évolution des lapins.**

Un bébé lapin a 12 % de chances de survie donc 88 % de chances de mourir.

On peut donc tracer encore une fois de nouveaux histogrammes pour illustrer la situation.



La fonction MortAdulte prend en paramètre la décroissance, elle est utilisée dans la grande fonction de mortalité, à partir de 10 ans, l'espérance de vie des lapins diminue de 10 % chaque année.

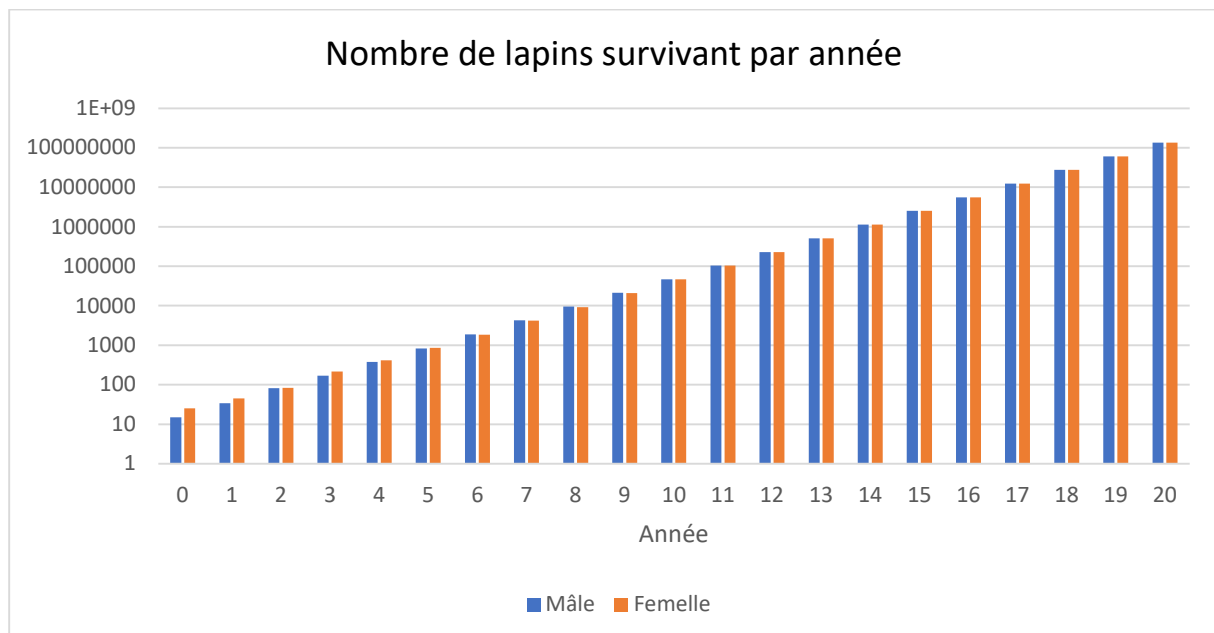


c. Résultats de la simulation

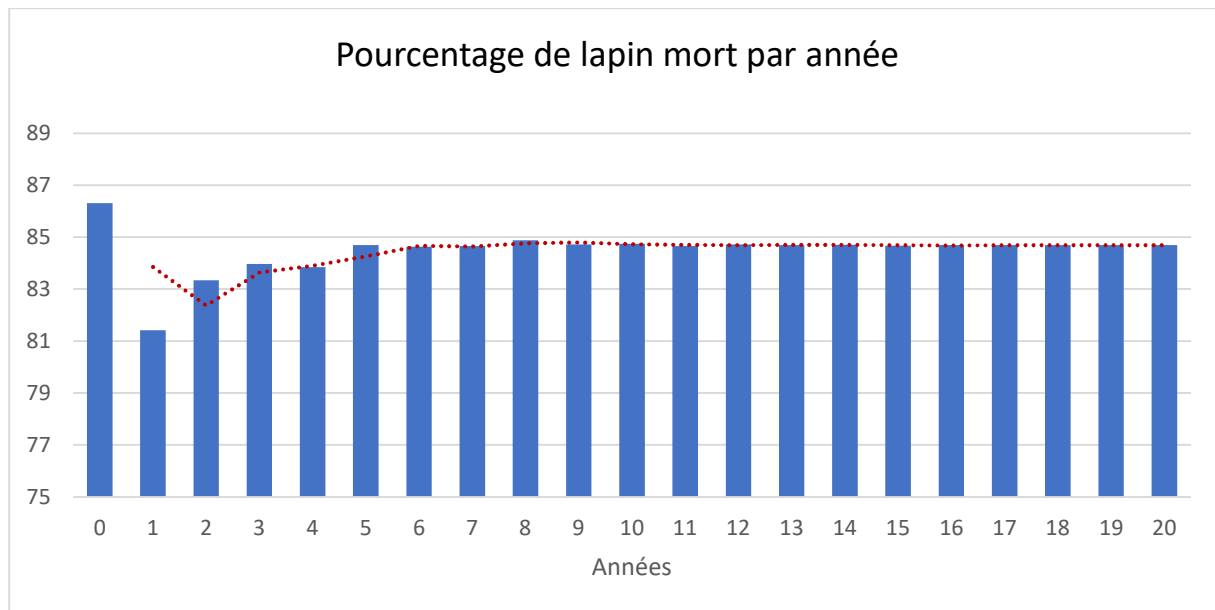
Après vous avoir expliqué les fonctions qui permettent de réaliser la simulation de manière correcte, **on effectue une simulation de 20 ans** (les résultats sont en annexes) et on va tenter de les exploiter ici. Ils nous permettront de comprendre comment va évoluer la situation sur cette île par rapport à la population de lapins.

Une échelle logarithmique contrairement à une échelle linéaire permet de représenter des valeurs ayant des tailles très différentes. Vu que nous cherchons à comparer des valeurs qui peuvent être très grandes et très petites à la fois, **l'échelle logarithmique nous permet de représenter ces valeurs au même niveau sans perdre la qualité informative qu'elles peuvent produire.**

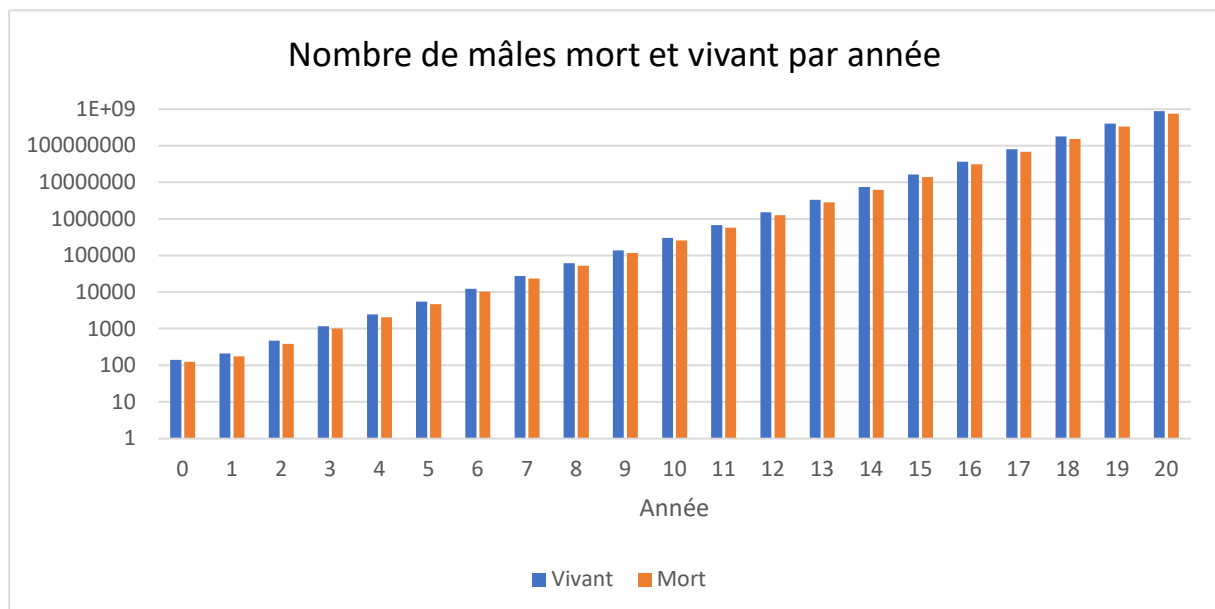
On rappelle que la simulation est initialisée avec 10 lapins mâles et femelles âgés de 5 ans.



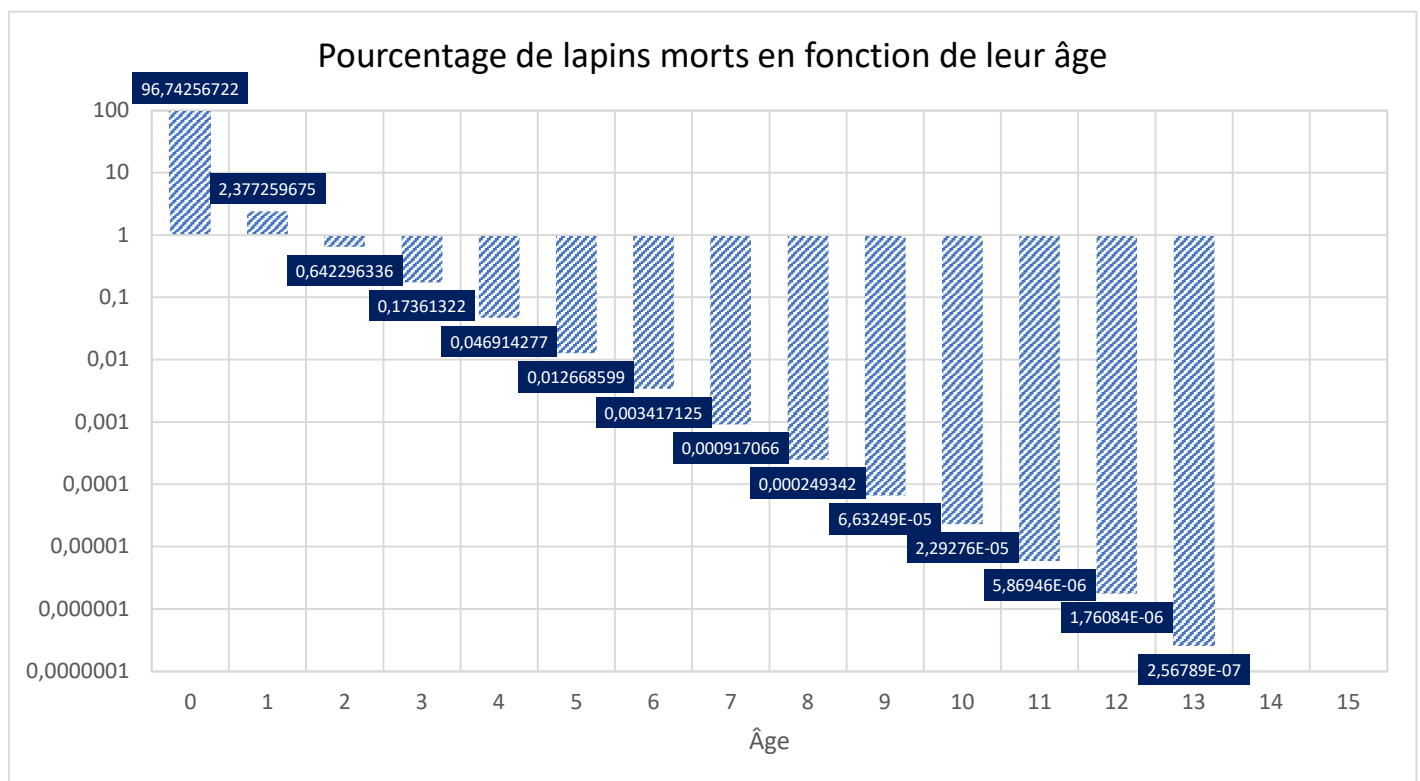
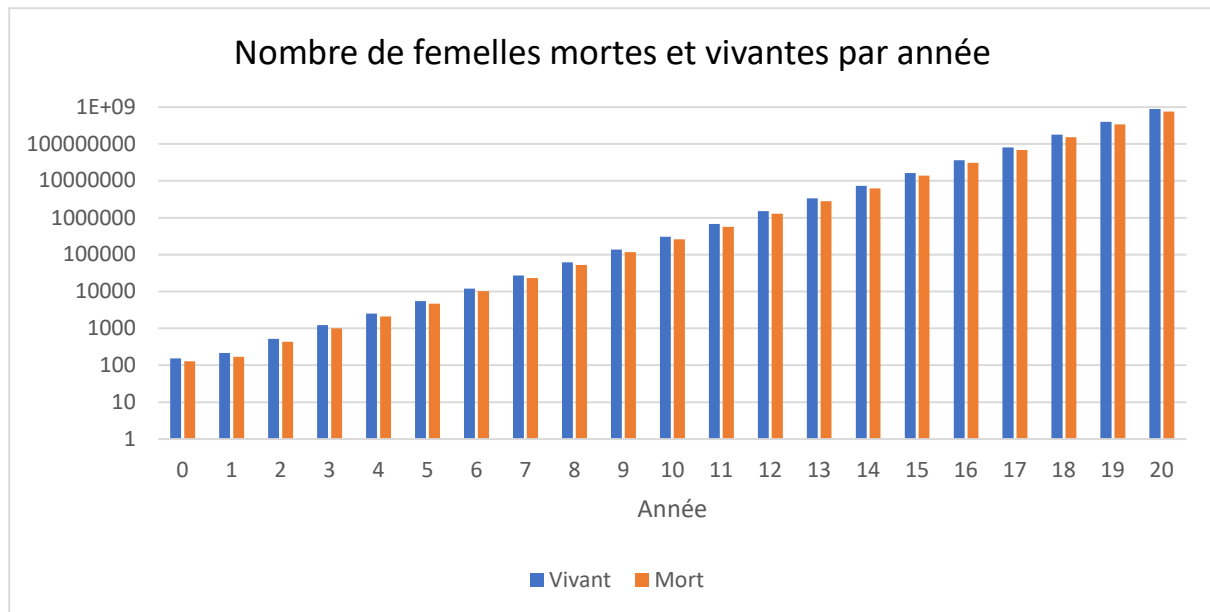
On peut voir sur ce graphique le nombre de mâles et de femelles qui survivent chaque année, sur les 20 ans de la simulation. Dans un premier temps, on constate que **le nombre de mâles et de femelles est assez équivalent**, mais on constate surtout que **la proportion de lapins augmente de manière exponentielle** (on n'oublie pas que nous sommes dans une représentation avec une échelle logarithmique) au fur et à mesure des années. Mais dans un second temps, on se rend bien compte qu'il nous manque des éléments pour pouvoir pleinement tirer parti de ces informations. **On va maintenant voir le pourcentage de lapins mort par années.**



On constate cette fois ci, que sur le nombre total de lapins pour chaque année, **on observe un pic à 86,30 % à l'année 0** (l'année où il y a un fort taux de mortalité de bébés) **puis une chute à 81,41 % l'année suivante**, et enfin **une stabilisation aux alentours de 84,6 %**. On enregistre **une moyenne aux alentours de 84,6 % de morts par année**. On peut penser à première vue que c'est beaucoup, mais vu que le nombre de lapins augmente de manière exponentielle chaque année, on peut donc dire que **les lapins ont un fort taux de reproduction**.



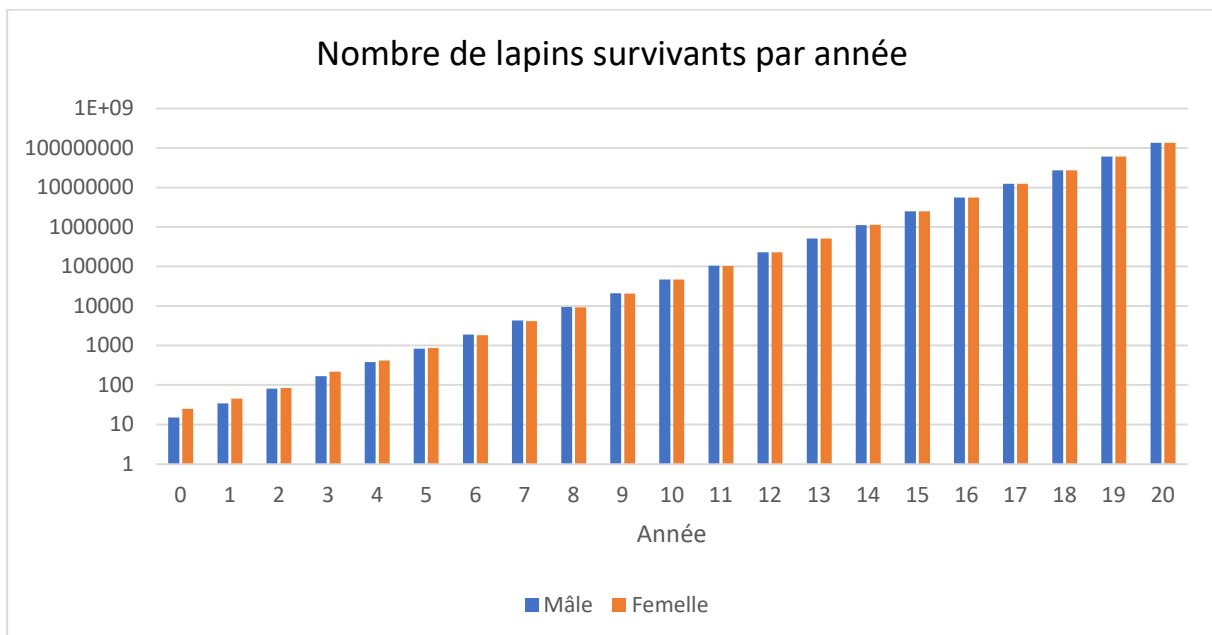
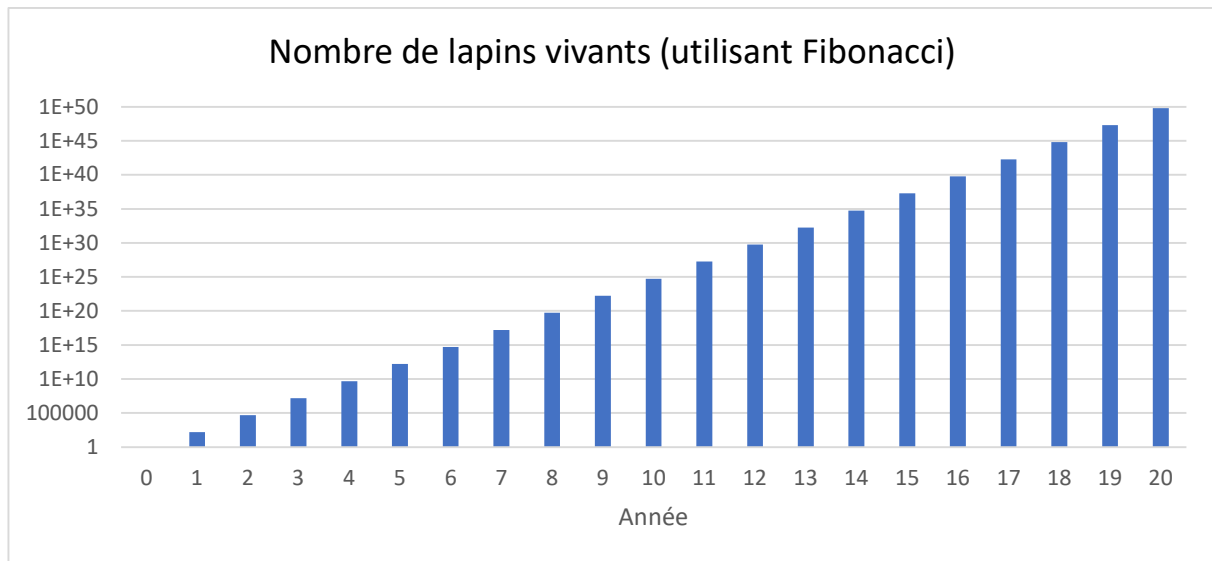
Sur ces deux diagrammes en bâton (ci-dessus et ci-dessous), on constate deux choses. La première **c'est qu'ils se ressemblent à un point tel que l'on pourrait les confondre**, ceci est dû au fait que l'on a une **équiprobabilité d'obtenir un mâle ou une femelle**. La deuxième chose que l'on remarque, c'est le fait qu'il y ait un peu plus de mâles et de femelles vivants que de morts chaque année, **c'est cela qui permet d'augmenter le nombre de lapins l'année suivante** (car ils vivent 15 ans au maximum).



Ce dernier diagramme nous permet d'observer le pourcentage de lapins morts pour chaque tranche d'âge parmi tous les morts de la simulation. De ce fait, **on remarque que 96,74 % des lapins qui sont morts durant cette simulations sont morts entre 0 et 1 an (mortalité juvénile)**, ce qui est assez considérable. **A l'âge 1, on passe à 2,38 % de morts représenté**, et ainsi de suite de manière décroissante. On remarque également que nous n'avons enregistré aucun décès après l'âge de 13 ans car **aucun de nos lapins n'a eu la chance de survivre assez longtemps**.



d. Comparaison avec la simulation par la suite de Fibonacci



La différence entre ces deux modèles repose sur le fait que dans le cas de notre simulation, on initialise une situation de départ, avec un nombre précis de lapins mâles et femelles, ce qui n'est pas le cas, pour Fibonacci. Ce dernier modèle part du principe, que c'est un seul couple qui génère tous les autres.

On constate par ailleurs qu'à la fin de la 20^{ème} année, on arrive avec un nombre de lapins égal à 64 202 014 863 723 094 126 901 777 428 873 111 802 307 548 623 680 lapins (comment prononcer ce nombre d'ailleurs???) pour la simulation utilisant Fibonacci. A contrario, avec notre simulation, on obtient un nombre total de 270 810 867 lapins. On remarque donc une infime différence entre ces deux résultats (ironiquement parlant).

On peut donc affirmer que la simulation utilisant la suite de Fibonacci est très loin d'être une bonne simulation pour une population de lapins.



IV) Pour aller plus loin...

Comme nous avons pu le voir tout au long de ce dossier, nous avons tenté de simuler l'évolution d'une population de lapins. Or, **nous nous sommes vite heurtés à un problème : le temps de calcul**. Passer des secondes aux minutes puis aux heures en fonction du nombre d'années qui passe est un problème. Car si une simulation qui met 10 minutes pour être exécutée, ne pose pas réellement de soucis si on la fait 20 fois ; une simulation qui met 2 h pour être exécutée posera des problèmes si on doit en faire 20 ! **Il faudra très très vite, prévoir beaucoup de temps.**

Dans notre cas, même si le temps de simulation était assez faible, **nous n'avons effectué qu'une seule simulation**, de ce fait, les résultats présentés peuvent ne pas être suffisamment précis si on souhaite les comparer à une autre situation par exemple.

Cependant, nous avons tenté de nous renseigner et d'appliquer quelques méthodes, avec un succès relatif parfois des **techniques de calcul parallèle permettant d'augmenter la rapidité de notre programme.**

Qu'est-ce qu'un ordinateur ? C'est :

- Un ou des programmes stockés en mémoire
- Des mémoires pour les programmes et les données
- Une ou plusieurs unités de calcul et unités logiques

La vitesse est donc limitée soit :

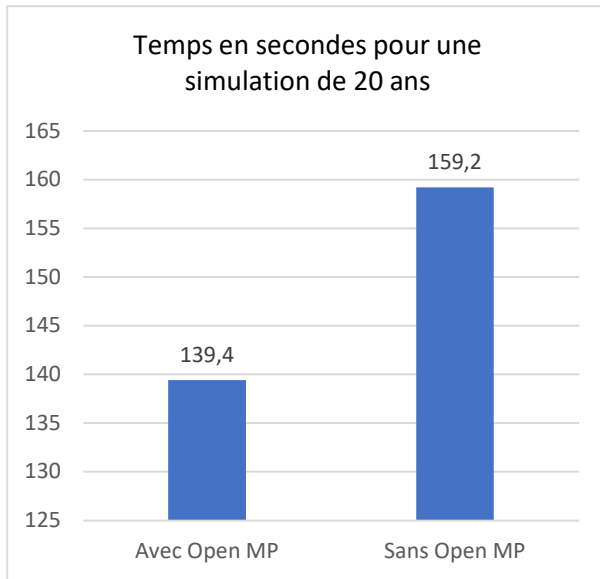
- Par la vitesse du processeur
- Par le taux de transfert du bus entre la mémoire et le CPU
- Par les conflits entre opérations et transferts dans les machines vectorielles et multicœurs.

Toutes **les opérations sont traduites en binaire** (algèbre de Boole) et implémentées par des **portes logiques** (sur des plaques de silicium). Ce mode de fonctionnement pourrait être remis en question pour les **ordinateurs quantiques** qui ne fonctionnent pas de la même manière.

Le parallélisme est un ensemble de techniques logicielles et matérielles permettant l'exécution simultanée de séquences d'instructions « indépendantes » sur plusieurs cœurs de calcul. En effet, lorsque l'on compile un programme avec **gcc** par exemple, lors de son exécution, il tournera sur un seul cœur du processeur, ce qui est dommage car aujourd'hui, la grande majorité des machines possèdent un processeur multicœur.

Nous avons donc tenté, pour accélérer notre programme, **d'implémenter des techniques de calcul parallèle en utilisant OpenMP**. Les directives sont données au compilateur grâce au **#pragma** indiqué aux endroits où on veut lancer le parallélisme. La difficulté est de bien configurer ces directives, **car le programme risquerait de durer plus longtemps que le même programme non parallèle.**

Il se compile avec le flag **-fopenmp** dans gcc.



On constate bien le gain de temps provoqué par l'utilisation d'openMP lors d'une simulation de 20 années initialisée avec les mêmes paramètres.

Par ailleurs, compte tenu du fort taux de reproduction des lapins, des expressions ont vu le jour, comme par exemple :

- **Être une mère lapine** : avoir beaucoup d'enfants
- **Se reproduire comme des lapins** : faire beaucoup d'enfants

Puis des blagues un peu douteuses...

Quelle est la différence entre la myxomatose et la blennorragie ?

- La myxomatose, c'est une maladie de lapin, et la blennorragie une maladie de la pine.



ANNEXES

Année 0

[illegible]

Année 1

195	10	0	0	0	5	0	0	0	0	0	0	0	0	0
172	1	0	0	0	3	0	0	0	0	0	0	0	0	0
190	19	0	0	0	6	0	0	0	0	0	0	0	0	0
158	9	0	0	0	3	0	0	0	0	0	0	0	0	0

Année 2

432	23	9	0	0	0	2	0	0	0	0	0	0	0	0	0
372	8	4	0	0	0	1	0	0	0	0	0	0	0	0	0
473	32	10	0	0	0	3	0	0	0	0	0	0	0	0	0
414	15	3	0	0	0	3	0	0	0	0	0	0	0	0	0

Année 3

[illegible]

Année 4

[illegible]

Année 5

5144	288	63	25	3	2	0	0	0	0	0	0	0	0	0
4531	117	32	13	1	2	0	0	0	0	0	0	0	0	0
5113	300	89	23	4	2	0	0	0	0	0	0	0	0	0
4482	141	36	6	2	1	0	0	0	0	0	0	0	0	0

Année 6

11366	613	171	31	12	2	0	0	0	0	0	0	0	0	0
9977	243	67	11	3	1	0	0	0	0	0	0	0	0	0
11186	631	159	53	17	2	1	0	0	0	0	0	0	0	0
9858	255	70	26	6	1	0	0	0	0	0	0	0	0	0

Année 7

25783	1389	370	104	20	9	1	0	0	0	0	0	0	0	0
22644	556	159	39	8	2	0	0	0	0	0	0	0	0	0
25491	1328	376	89	27	11	1	1	0	0	0	0	0	0	0
22404	562	126	40	12	4	1	0	0	0	0	0	0	0	0

Année 8															
57308	3139	833	211	65	12	7	1	0	0	0	0	0	0	0	0
50429	1221	336	86	28	6	1	0	0	0	0	0	0	0	0	0
57856	3087	766	250	49	15	7	0	1	0	0	0	0	0	0	0
51053	1314	314	99	17	5	4	0	1	0	0	0	0	0	0	0

Année 9															
128085	6879	1918	497	125	37	6	6	1	0	0	0	0	0	0	0
112722	2705	737	191	52	19	2	1	1	0	0	0	0	0	0	0
127229	6803	1773	452	151	32	10	3	0	0	0	0	0	0	0	0
112085	2690	688	178	47	11	4	2	0	0	0	0	0	0	0	0

Année 10															
283180	15363	4174	1181	306	73	18	4	5	0	0	0	0	0	0	0
249159	6175	1675	492	129	28	8	2	2	0	0	0	0	0	0	0
285244	15144	4113	1085	274	104	21	6	1	0	0	0	0	0	0	0
251089	6163	1674	415	105	39	13	3	0	0	0	0	0	0	0	0

Année 11															
628914	34021	9188	2499	689	177	45	10	2	3	0	0	0	0	0	0
553365	13502	3554	1005	268	82	20	2	0	1	0	0	0	0	0	0
628982	34155	8981	2439	670	169	65	8	3	1	0	0	0	0	0	0
553167	13800	3654	954	293	72	21	4	1	0	0	0	0	0	0	0

Année 12															
1398965	75549	20519	5634	1494	421	95	25	8	2	2	0	0	0	0	0
1231559	30242	8319	2307	571	176	46	8	0	2	0	0	0	0	0	0
1399192	75815	20355	5327	1485	377	97	44	4	2	1	0	0	0	0	0
1231484	30336	8184	2134	593	149	36	19	2	0	0	0	0	0	0	0

Année 13															
3101945	167406	45307	12200	3327	923	245	49	17	8	0	2	0	0	0	0
2730396	67061	18096	4825	1295	381	99	18	8	3	0	1	0	0	0	0
3095178	167708	45479	12171	3193	892	228	61	25	2	2	1	0	0	0	0
2723586	66858	18432	4820	1255	328	95	25	8	0	1	1	0	0	0	0

Année 14															
6879007	371549	100345	27211	7375	2032	542	146	31	9	5	0	1	0	0	0
6054794	149195	40320	10989	2879	871	201	62	13	2	3	0	1	0	0	0
6875076	371592	100850	27047	7351	1938	564	133	36	17	2	1	0	0	0	0
6050520	148688	40354	10765	2962	800	230	58	9	8	0	1	0	0	0	0

Année 15															
15244047	824213	222354	60025	16222	4496	1161	341	84	18	7	2	0	0	0	0
13411568	329479	88753	24085	6592	1836	473	123	31	8	3	1	0	0	0	0
15246166	824556	222904	60496	16282	4389	1138	334	75	27	9	2	0	0	0	0
13414649	329351	89398	24187	6473	1767	426	125	32	13	5	1	0	0	0	0

Année 16															
33879343	1832479	494734	133601	35940	9630	2660	688	218	53	10	4	1	0	0	0
29809590	732238	198406	53473	14558	3822	1069	281	86	21	5	1	1	0	0	0
33882185	1831517	495205	133506	36309	9809	2622	712	209	43	14	4	1	0	0	0
29818206	733373	198131	53412	14534	4031	1099	262	79	15	10	3	0	0	0	0

Année 17															
75269612	4069753	1100241	296328	80128	21382	5808	1591	407	132	32	5	3	0	0	0
66235986	1628448	440214	118994	31922	8440	2332	651	181	54	17	2	3	0	0	0
75253960	4063979	1098144	297074	80094	21775	5778	1523	450	130	28	4	1	1	0	0
66224737	1625315	439294	119268	31987	8631	2264	642	178	48	15	2	1	1	0	0

Année 18															
167084285	9033626	2441305	660027	177334	48206	12942	3476	940	226	78	15	3	0	0	0
147038593	3613254	975458	263433	71082	19393	5192	1358	365	85	39	7	2	0	0	0
167101450	9029223	2438664	658850	177806	48107	13144	3514	881	272	82	13	2	0	0	0
147047484	3611631	974386	263182	70929	19359	5223	1366	380	104	39	4	1	0	0	0

Année 19															
370882466	20045692	5420372	1465847	396594	106252	28813	7750	2118	575	141	39	8	1	0	0
326367964	8020966	2168720	586170	158612	42333	11441	3053	873	238	64	16	6	1	0	0
370887432	20053966	5417592	1464278	395668	106877	28748	7921	2148	501	168	43	9	1	0	0
326381792	8017614	2165020	585472	158043	42721	11352	3201	879	183	81	23	6	1	0	0

Année 20															
823535999	44514502	12024726	3251652	879677	237982	63919	17372	4697	1245	337	77	23	2	0	0
724732105	17811079	4809391	1300961	351773	94969	25704	6755	1843	509	176	42	15	1	0	0
823481467	44505640	12036352	3252572	878806	237625	64156	17396	4720	1269	318	87	20	3	0	0
724660063	17808755	4814796	1300608	351829	95057	25790	6978	1824	514	167	55	12	3	0	0

```

1 /*****
2 *
3 *
4 *
5 *
6 *
7 *
8 *
9 *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 *
25 *
26 *
27 *
28 *
29 *
30 *
31 *
32 *
33 *
34 *
35 *****/
36
37 #include <stdio.h>
38
39 /* ----- */
40 /*          Prototypes des fonctions          */
41 /* ----- */
42
43 unsigned long long fibo1 (int mois);
44
45 unsigned long long fibo2 (int mois);
46
47 /* ----- */
48 /*          Fonction 'main' principale          */
49 /* ----- */
50
51 /*****
52 *
53 * On vient tout d'abord récupérer de choix de l'utilisateur sur les 2
54 * méthodes proposées. Puis nous récupérons le nombre de mois qui correspond
55 * au nombre d'itérations que devra effectuer la suite de fibonacci. Pour
56 * ensuite avec un switch venir appeler la fonction qui correspond a son
57 * choix et écrire dans le terminal la réponse attendue.
58 *
59 *****/
60
61 int main()
62 {

```



```

63     int mois;
64     int choix = 0;
65
66     do
67     {
68         printf("\t\tMenu : Simulation utilisant Fibonacci\n\n");
69         printf("1- Méthode recursive\n");
70         printf("2- Méthode impérative\n\n");
71         printf("Choix : ");
72         scanf("%d", &choix);
73     } while (choix < 1 || choix > 2);
74
75     printf("Nombres de mois à simuler : ");
76     scanf("%d", &mois);
77
78     switch (choix)
79     {
80     case 1:
81         printf("%lld\n", fibo1(mois));
82         break;
83     case 2:
84         fibo2(mois);
85         break;
86     }
87     return 0;
88 }
89
90 /* ----- */
91 /*           Fonctions servant au programme           */
92 /* ----- */
93
94 /*****
95  *
96  * Fonction : unsigned long long fibo1 (int mois)
97  *
98  * Sert à calculer le résultat de la suite de fibonacci au mois m de manière
99  * récursive.
100  *
101  * En entrée : Le nombre de mois à simuler
102  *
103  * En sortie : La valeur de fibo1(mois-1) + fibo(mois-2)
104  *
105  *****/
106
107 unsigned long long fibo1 (int mois)
108 {
109     if (mois <= 1)
110     {
111         return mois;
112     }
113     else
114     {
115         return fibo1(mois - 1) + fibo1(mois - 2);
116     }
117 }
118
119 /*****
120  *
121  * Fonction : unsigned long long fibo2 (int mois)
122  *
123  * Sert à calculer le résultat de la suite de fibonacci du mois 0 au mois m
124  * de manière impérative.

```

```
125 | *
126 | * En entrée : Le nombre de mois à simuler
127 | *
128 | * En sortie : Rien, juste de l'affichage
129 | *
130 | *****/
131 |
132 | unsigned long long fibo2 (int mois)
133 | {
134 |     unsigned long long terme_suivant;
135 |     unsigned long long premier_terme = 0;
136 |     unsigned long long deuxieme_terme = 1;
137 |     int i;
138 |
139 |     printf("%lld\n", premier_terme);
140 |     for (i = 1; i <= mois; i++)
141 |     {
142 |         terme_suivant = premier_terme + deuxieme_terme;
143 |         premier_terme = deuxieme_terme;
144 |         deuxieme_terme = terme_suivant;
145 |         printf("%lld\n", premier_terme);
146 |     }
147 |     printf("\n");
148 |     return 0;
149 | }
150 |
```

```

1 /*****
2 *
3 *
4 *
5 *
6 *
7 *
8 *
9 *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 *
25 *
26 *
27 *
28 *
29 *
30 *
31 *
32 *
33 *
34 *
35 *
36 *
37 *****/
38
39 #include <stdio.h>
40 #include <stdlib.h>
41 #include <limits.h>
42 #include <omp.h>
43
44 #include "mt19937ar.h"
45
46 /* ----- */
47 /*                               Prototypes des fonctions                               */
48 /* ----- */
49
50 void AfficheTableau(unsigned long long ***tableau, int nb_annee_simu);
51
52 double Uniform(double borne_inf, double borne_sup);
53
54 int nbLapinPortee();
55
56 int nbPortee();
57
58 int SexeLapin();
59
60 int MortPetit();
61
62 int MortAdulte(double decroissance);

```

```

63
64 unsigned long long *NaissanceSexuee(unsigned long long ***tableau, int annee);
65
66 unsigned long long **Mortalite(unsigned long long ***tableau, unsigned long long
67 *tab_naissances, int annee);
68
69 unsigned long long ***Evolution(unsigned long long ***tableau, int nb_annee);
70
71 unsigned long long ***AllocationTab3D(int nb_annee_simu, int ligne, int age);
72
73 /* ----- */
74 /*
75
76 /*****
77 *
78 * Pour expliquer la manière dont est construit l'algorithme, on représente
79 * un tableau sous cette forme, ce tableau représente une année. On se sert
80 * d'un tableau en trois dimensions pour représenter chaque année de la
81 * même manière.
82 *
83 *
84 *
85 *
86 *
87 *
88 *
89 *
90 *
91 *
92 *****/
93
94 int main(int argc, char *argv[])
95 {
96
97     int i;
98     int nombre_annee_simu = 23;
99     unsigned long long ***matrix_result, ***result_fin;
100
101     printf("Nombre d'arguments passes au programme : %d\n", argc);
102     for (i = 0; i < argc; i++)
103     {
104         printf(" argv[%d] : '%s'\n", i, argv[i]);
105     }
106
107     // On alloue de l'espace en mémoire pour les tableaux matrix_result et result_fin
108     // Ces deux tableaux ont la même représentation qu'indiqué dans les commentaires
109     // ci-dessus.
110
111     matrix_result = AllocationTab3D(nombre_annee_simu, 4, 16);
112     result_fin = AllocationTab3D(nombre_annee_simu, 4, 16);
113
114     // On initialise ici la tableau matrix_result avec des valeurs pour les premiers
115     // lapins. Ici en l'occurrence, on initialise avec 10 lapins mâles et femelles
116     // qui ont respectivement 4 ans.
117     matrix_result[0][0][4] = 10;
118     matrix_result[0][2][4] = 10;
119
120     // On met dans ce tableau les résultats de la simulation calculés sur le nombre
121     // d'années pris en deuxième paramètre de la fonction Evolution.
122     result_fin = Evolution(matrix_result, 22);
123

```

```

124 // On affiche maintenant Le tableau pour visualiser les résultats.
125 AfficheTableau(result_fin, nombre_annee_simu);
126
127 //desallocation(matrix_result, 20, 4);
128 //desallocation(result_fin, 20, 4);
129
130 return EXIT_SUCCESS;
131 }
132
133 /* ----- */
134 /*          Fonctions servant au programme          */
135 /* ----- */
136
137 /*****
138 *
139 * Fonction : ***Evolution (int ***tableau, int nb_annee)
140 *
141 * Permet de calculer le nombre de simulations correspondant à l'année entrée
142 * sur une population de lapins initialisée avant son appel.
143 *
144 * En entrée : Un tableau en 3 dimensions avec juste les premières valeurs
145 *              initialisées.
146 *              Le nombre d'années sur lequel l'algorithme doit simuler la
147 *              population de lapins.
148 *
149 * En sortie : Un tableau complet en 3 dimensions contenant les résultats
150 *              générés.
151 *
152 * Le tableau naissance est représenté de la manière suivante :
153 *
154 * | Nb bb femelles | Nb bb mâles |
155 *
156 *
157 * Le tableau mort est représenté de la manière suivante :
158 *
159 * | | Age | | | | |
160 * |-----|-----|-----|-----|-----|
161 * | Nombre de femelles mortes | 0 | 1 | 2 | 3 | ... | 16 |
162 * | Nombre de mâles morts    | 0 | 1 | 2 | 3 | ... | 16 |
163 *
164 *
165 *****/
166
167 unsigned long long ***Evolution(unsigned long long ***tableau, int nb_annee)
168 {
169
170     int i, annee;
171     unsigned long long *naissance;
172     unsigned long long **mort;
173
174     for (annee = 1; annee < nb_annee; annee++)
175     {
176
177         // On remplit ici le tableau des naissances avec le nombre de bébés
178         // lapins mâles et femelles obtenu durant l'année précédente.
179         naissance = NaissanceSexuee(tableau, annee - 1);
180
181         // On remplit ici le tableau des morts avec, le nombre de lapins morts en
182         // fonction de leur âge que l'on obtient à la fin de l'année précédente
183         // en prenant en considération le nombre de naissances.
184         mort = Mortalite(tableau, naissance, annee - 1);
185

```

```

186     printf("\rAnnées simulées : %d sur %d", annee, nb_annee);
187     fflush(stdout);
188
189     tableau[annee - 1][0][0] = naissance[0];
190     tableau[annee - 1][2][0] = naissance[1];
191
192 #pragma omp parallel for
193     for (i = 0; i < 16; i++)
194     {
195         tableau[annee - 1][1][i] = mort[0][i];
196         tableau[annee - 1][3][i] = mort[1][i];
197     }
198
199     // On calcule le nombre de lapins de l'année n - 1 à l'année n :
200     // On remplit le tableau de l'année en cours avec le nombre de lapins qui on
201     // survécue à l'année précédente en les vieillissant d'un an.
202
203     // #pragma omp parallel for
204     for (i = 1; i < 16; i++)
205     {
206         tableau[annee][0][i] = tableau[annee - 1][0][i - 1] - tableau[annee - 1][1][i - 1];
207         tableau[annee][2][i] = tableau[annee - 1][2][i - 1] - tableau[annee - 1][3][i - 1];
208     }
209 }
210
211 return tableau;
212 }
213
214 /*****
215 *
216 * Fonction : **Mortalite (int ***tableau, int *tab_naissances, int annee)
217 *
218 * Permet de calculer la mortalité des lapins en fonction de leur âge et de
219 * leur sexe.
220 *
221 * En entrée : Un tableau en 3 dimensions initialisé au fur et à mesure
222 *             grâce à la fonction Evolution.
223 *             Un tableau correspondant au nombre de naissances (mâles et
224 *             femelles).
225 *             L'année sur laquelle on veut calculer la mortalité.
226 *
227 * En sortie : Un tableau de mortalité contenant les résultats générés.
228 *
229 * Le tableau naissance est représenté de la manière suivante :
230 *
231 * | Nb bb femelles | Nb bb mâles |
232 *
233 *
234 * Le tableau mort est représenté de la manière suivante :
235 *
236 * | | Age | | | | |
237 * | | | | | | | |
238 * | Nombre de femelles mortes | 0 | 1 | 2 | 3 | ... | 16 |
239 * | Nombre de mâles morts | 0 | 1 | 2 | 3 | ... | 16 |
240 *
241 *
242 *****/
243
244 unsigned long long **Mortalite(unsigned long long ***tableau, unsigned long long
*tab_naissances, int annee)
245 {
246

```

```

247     int i, j, k;
248     double decroissance = 0;
249     unsigned long long **tab_mort = (unsigned long long **)calloc(2, sizeof(unsigned long long
*)));
250
251     // Allocation dynamique d'un tableau en 2 dimensions.
252     for (i = 0; i < 2; i++)
253     {
254         tab_mort[i] = (unsigned long long *)calloc(16, sizeof(unsigned long long));
255     }
256
257     // Remplissage du tableau mort avec Le nombre de bébé Lapins morts
258     // mâles et femelles générés.
259     for (i = 0; i < 2; i++)
260     {
261
262         for (j = 0; j < tab_naissances[i]; j++)
263         {
264             tab_mort[i][0] += MortPetit();
265         }
266     }
267
268     // Remplissage du tableau mort avec Le nombre de Lapins adultes morts
269     // mâles et femelles générés, sauf qu'à partir de 10 ans, Leur chance
270     // de survie diminue de 10 % chaque année.
271
272     for (i = 0; i < 2; i++)
273     {
274         decroissance = 0;
275         for (j = 1; j < 16; j++)
276         {
277
278             if (j >= 10)
279             {
280                 decroissance += 0.1;
281             }
282
283             for (k = 0; k < tableau[annee][2 * i][j]; k++)
284             {
285                 tab_mort[i][j] += MortAdulte(decroissance);
286             }
287         }
288     }
289
290     return tab_mort;
291 }
292
293 /*****
294  *
295  * Fonction : int MortPetit()
296  *
297  * Sert à calculer la mortalité des bébés. Ils ont 12 % de chance de survie.
298  *
299  * En entrée : Rien.
300  *
301  * En sortie : 1 si le bébé lapin est mort
302  *              0 sinon.
303  *
304  *****/
305
306 int MortPetit()
307 {

```

```

308
309     double val_aleatoire = genrand_real1();
310     int val_retour = 0;
311     if (val_aleatoire >= 0.12)
312     {
313         val_retour++;
314     }
315
316     return val_retour;
317 }
318
319 /*****
320 *
321 * Fonction : int MortAdulte (double decroissance)
322 *
323 * Sert à calculer la mortalité des lapins selon leur âge.
324 * Ils ont 60 % de chance de survie de 1 à 10 ans, mais à partir de 10 ans,
325 * leur chance de survie diminue de 10 % tous les ans.
326 *
327 * En entrée : La décroissance, elle est modifiée dans la fonction Evolution
328 *               lorsque que le lapin est âgé de plus de 10 ans.
329 *
330 * En sortie : 1 si le lapin est mort
331 *              0 sinon.
332 *
333 *****/
334
335 int MortAdulte(double decroissance)
336 {
337
338     double val_aleatoire = genrand_real1();
339     int val_retour = 0;
340
341     if (val_aleatoire >= (0.60 - decroissance))
342     {
343         val_retour++;
344     }
345
346     return val_retour;
347 }
348
349 /*****
350 *
351 * Fonction : *NaissanceSexuee (int ***tableau, int annee)
352 *
353 * Permet de calculer le nombre de bébés lapins mâles et femelles en fonction
354 * du nombre de portées et du nombre de lapins par portée.
355 *
356 * En entrée : Un tableau en 3 dimensions initialisé au fur et à mesure
357 *               grâce à la fonction Evolution.
358 *               L'année sur laquelle on veut calculer le nombre de naissances
359 *               ainsi que le sexe des nouveaux lapins.
360 *
361 * En sortie : Un tableau de naissance contenant les résultats générés.
362 *
363 * Le tableau naissance est représenté de la manière suivante :
364 *
365 * | Nb bb femelles | Nb bb mâles |
366 *
367 *
368 * Il y a 50 % de chances d'obtenir un mâle ou une femelle.
369 * Il y a 4 à 8 portées chaque année par lapine adulte, mais il est plus

```



```

370 * probable d'en avoir 5 à 7. *
371 * Il y a une équiprobabilité d'obtenir entre 3 et 6 lapins par portée. *
372 * *
373 *****/
374
375 unsigned long long *NaissanceSexuee(unsigned long long ***tableau, int annee)
376 {
377     int i, j, k;
378     unsigned long long nb_femelles_mature = 0,
379         nb_bb_males = 0,
380         nb_bb_femelles = 0,
381         nb_bb_portee = 0,
382         nb_bb_tot = 0;
383
384     // tab_result est le tableau où seront stocké les informations des
385     // naissances. C'est pour cela que l'on lui alloue de la mémoire ici.
386     unsigned long long *tab_result = (unsigned long long *)malloc(2 * sizeof(unsigned long
387 long));
388
389     for (k = 1; k <= 15; k++)
390     {
391         nb_femelles_mature += tableau[annee][0][k];
392     }
393
394     unsigned long long *tab_portee = (unsigned long long *)malloc(nb_femelles_mature *
395 sizeof(unsigned long long));
396     unsigned long long *tab_naissance = (unsigned long long *)malloc(nb_femelles_mature *
397 sizeof(unsigned long long));
398
399     // On définit ici le nombre de mâles et de femelles créé pour chaque
400     // femelle mature (âge supérieur à 1 an) et pour chaque portée qu'elles
401     // donneront.
402
403     for (i = 0; i < nb_femelles_mature; i++)
404     {
405         tab_portee[i] = nbPortee();
406
407         for (j = 0; j < tab_portee[i]; j++)
408         {
409             nb_bb_portee = nbLapinPortee();
410
411             for (k = 0; k < nb_bb_portee; k++)
412             {
413                 tab_naissance[k] = SexeLapin();
414
415                 if (tab_naissance[k] == 1)
416                 {
417                     nb_bb_males++;
418                 }
419                 else
420                 {
421                     nb_bb_femelles++;
422                 }
423             }
424             nb_bb_tot += nb_bb_portee;
425         }
426     }
427
428     // On remplit le tableau

```

```

430     tab_result[0] = nb_bb_femelles;
431     tab_result[1] = nb_bb_males;
432
433     return tab_result;
434 }
435
436 /*****
437 *
438 * Fonction : int nbPortee()
439 *
440 * Sert à calculer le nombre de portées total par lapine sur une année, il y
441 * a environ 4 à 8 portées par an, mais il y a plus de chance d'en obtenir
442 * entre 5 et 7.
443 *
444 * En entrée : Rien.
445 *
446 * En sortie : Le nombre de portée.
447 *
448 * La répartition du nombre de portées se fait comme suit :
449 *
450 *          █ █ █ █ 4 - 10 %
451 *          █ █ █ █ █ 5 - 20 %
452 *          █ █ █ █ █ █ 6 - 40 %
453 *          █ █ █ █ █ 7 - 20 %
454 *          █ █ █ █ 8 - 10 %
455 *
456 *****/
457
458 int nbPortee()
459 {
460     int i;
461     double valGene = genrand_real1();
462     double pourcentage[5] = {0.1, 0.3, 0.7, 0.9, 1.0};
463
464     for (i = 0; i <= 4; i++)
465     {
466         if (valGene <= pourcentage[i])
467         {
468             return (4 + i);
469         }
470     }
471
472     return EXIT_SUCCESS;
473 }
474
475
476
477
478 /*****
479 *
480 * Fonction : void AfficheTableau (int ***tableau, int nb_annee_simu)
481 *
482 * Permet simplement d'afficher un tableau en 3 dimensions.
483 *
484 * En entrée : Un tableau à 3 dimensions
485 *              Le nombre d'années sur lesquelles on doit afficher le tableau
486 *
487 * En sortie : Rien, cette fonction ne fait que de l'affichage.
488 *
489 *****/
490
491 void AfficheTableau(unsigned long long ***tableau, int nb_annee_simu)

```

```

492 {
493
494     int i, j, k;
495
496     for (i = 0; i < nb_annee_simu; i++)
497     {
498
499         printf("Année %d\n", i);
500
501         for (j = 0; j <= 3; j++)
502         {
503
504             for (k = 0; k <= 15; k++)
505             {
506
507                 printf("%11ld\t", tableau[i][j][k]);
508             }
509             printf("\n");
510         }
511
512         printf("\n\n");
513     }
514 }
515
516 /*****
517 *
518 * Fonction : int SexeLapin()
519 *
520 * Permet de déterminer si un lapin est un mâle ou une femelle, il y a 50 %
521 * de chance que se soit l'un ou l'autre.
522 *
523 * En entrée : Rien.
524 *
525 * En sortie : 1 si le bébé lapin est mâle
526 *              0 si c'est une femelle
527 *
528 *****/
529
530 int SexeLapin()
531 {
532
533     double val = genrand_real1();
534     if (val <= 0.5)
535     {
536         return 0;
537     }
538     else
539     {
540         return 1;
541     }
542 }
543
544 /*****
545 *
546 * Fonction : int nbLapinPortee()
547 *
548 * Permet de calculer le nombre de lapin par portée. Il y a environ 4 à 8
549 * portées par an, mais il y a plus de chance d'en obtenir entre 5 et 7.
550 *
551 * En entrée : Rien.
552 *
553 * En sortie : Le nombre de lapins par portée
554 */

```

```

554 *
555 * La répartition du nombre de lapins se fait comme suit :
556 *
557 *
558 *
559 *
560 *
561 *
562 *****/
563
564 int nbLapinPortee()
565 {
566
567     int x = (int)(Uniform(2.0, 6.0) + 1);
568
569     return x;
570 }
571
572 /*****
573 *
574 * Fonction : double Uniform (double borne_inf, double borne_sup)
575 *
576 * Permet de générer aléatoirement un nombre de type double compris entre
577 * borne_inf et borne_sup.
578 *
579 * En entrée : Une borne inférieure : borne_inf
580 *             Une borne supérieure : borne_sup
581 *
582 * En sortie : Le nombre compris entre ces bornes.
583 *
584 *****/
585
586 double Uniform(double borne_inf, double borne_sup)
587 {
588
589     return (borne_inf + (borne_sup - borne_inf) * genrand_real1());
590 }
591
592 /*****
593 *
594 * Fonction : ***AllocationTab3D(int nb_annee_simu, int ligne, int age)
595 *
596 * Permet d'allouer de manière dynamique la mémoire pour un tableau de 3
597 * dimensions.
598 *
599 * En entrée : Le nombre d'années à simuler
600 *             Le nombre de lignes du tableau
601 *             L'âge maximum des lapins + 1
602 *
603 * En sortie : Un tableau alloué prêt à être utilisé
604 *
605 * Le tableau est de ce type : [Année][Ligne][Âge]
606 *
607 *****/
608
609 unsigned long long ***AllocationTab3D(int nb_annee_simu, int ligne, int age)
610 {
611     unsigned long long ***matrix_result;
612     int i, j;
613
614     matrix_result = malloc(nb_annee_simu * sizeof(unsigned long long **));
615     matrix_result[0] = malloc(nb_annee_simu * ligne * sizeof(unsigned long long *));

```

```
616 matrix_result[0][0] = malloc(nb_annee_simu * ligne * age * sizeof(unsigned long long));
617
618 for (i = 0; i < nb_annee_simu; i++)
619 {
620     matrix_result[i] = matrix_result[0] + i * ligne;
621     for (j = 0; j < ligne; j++)
622     {
623         matrix_result[i][j] = matrix_result[0][0] + i * ligne * age + j * age;
624     }
625 }
626 return matrix_result;
627 }
628
```