TP d'Algorithmique et Programmation 4 : Autour du générateur de nombre aléatoires (test de norme qualité puis utilisation)

IMT Lille Douai – FISE L3

Les deux exercices sont indépendants, répartissez votre temps de tel manière que vous puissiez aborder les deux problèmes.

Test du générateur de nombres aléatoires de votre machine

L'aléatoire n'existe pas en informatique. Cependant, il peut être utile de le simuler, et ce dans de nombreuses applications. Par exemple, lorsque vous voulez faire défiler vos images ou lire votre musique dans un ordre non déterminé. Pour faire ceci, un ordinateur va utiliser un générateur appelé pseudo-aléatoire, qui sera initialisé par une « graine ». Celle-ci est une valeur qui va déterminer la « suite de nombres aléatoires ». Cette suite, qui n'a d'aléatoire que le nom car elle dépend de la graine utilisée, va cependant respecter des conditions sur la distribution des valeurs dont la densité doit être uniforme sur l'ensemble de définition.
Pour des applications demandant de la sécurité, la valeur de cette graine est en générale prise sur un capteur dont la sortie n'est pas déterminable à un instant donné (par exemple un capteur de température qui va mesurer le bruit thermique d'un composant).

Norme FIPS (Federal Information Processing Standard) 140-2

Pour tester si un générateur est « suffisamment aléatoire », plusieurs normes sont définies. Parmi les normes existantes, la norme FIPS 140-2 (publiée en 2001 par le gouvernement Américain), définit les pré-requis à tout système cryptographique et est utilisée et définie par le gouvernement américain. Dans cette norme, une partie est réservée au test des générateurs aléatoires, qui doivent, pour pouvoir être utilisés dans un contexte cryptographique, respecter certaines normes qualité.
Pour tester votre générateur suivant cette norme, il vous fait prendre 20000 bits consécutifs de votre générateur aléatoire, puis faire quatre tests successifs sur cet ensemble de bits :

1. Le *Monobit test* : pour ce test, il suffit de compter le nombre de « 1 » présent dans la suite de bits, que l'on notera X. Le test est passé si $9654 < X < 10346$.
2. Le *poker test* : il faut diviser notre ensemble de 20000 bits en 5000 ensembles de 4 bits consécutifs. Ces ensembles de 4 bits donneront, en décimale, une valeur entre 0 et 15 inclus. Il faut alors compter le nombre d'occurrences de chacune des 16 valeurs. En notant $f(i), 0 \leq i \leq 15$ ce nombre d'occurrences pour les 16 valeurs différentes, il faut ensuite évaluer la fonction
$$X = \left(\frac{16}{5000} \cdot \sum_{i=0}^{15} f(i)^2 \right) - 5000$$
Ce test est passé si $1.03 < X < 57.4$. Ceci revient à faire un test du Chi^2 sur 4 bits successifs (le nom de poker test est donné dans cette norme alors qu'un vrai poker test tel que défini par D.E. Knuth est différent).
3. Le *run test* : On définit le « run » comme étant la taille de la séquence consécutive maximale de 0 ou de 1. Pour ce test, il faut donc comptabiliser séparément la taille des différentes séquences consécutives de 0 et de 1 (par exemple 00111101010011 aura deux séquences de longueur 1 pour les 0 et 2 séquences de longueur 1 pour les 1, deux de longueur 2 pour les 0 et une de longueur 2 pour les 1 et une de longueur 4 pour les 1). Pour passer le test, ces nombres de séquences pris séparément pour les 0 et pour les 1 doivent respecter les critères suivants :

Longueur	Nombre de séquences
1	entre 2267 et 2733
2	entre 1079 et 1421
3	entre 502 et 748
4	entre 223 et 402
5	entre 90 et 223
6 et plus	entre 90 et 223

4. Le *Long run test* : Une séquence trop longue est une séquence de 0 ou de 1 de 34 bits minimum. Aucune séquence de cette taille ne doit être présente dans votre séquence aléatoire.

Implémentation de la norme

Utilisation du générateur pseudo-aléatoire en Java

Listing 1 – Utilisation du générateur

```
1  import java.util.Random;
2
3  public class Main {
4      public static void main(String[] args) {
5          Random R = new Random(); // Créé un nouveau générateur
6
7          int i = R.nextInt(); // Génère un nombre entiers
8          boolean b = R.nextBoolean(); // Génère un boolean
9      }
10 }
```

Le code ci-dessus vous montre l'utilisation des générateurs aléatoires. La ligne 5 déclare un nouveau générateur qui sera initialisé avec une valeur par défaut (qui est l'heure actuelle en milliseconde). Par la suite (lignes 7 et 8), ce même générateur est utilisé pour générer d'abord un entier puis un boolean (valeur vrai ou faux).

Génération de la séquence

Créez une méthode qui va initialiser un générateur aléatoire puis créer une nouvelle séquence de N bits (avec N boolean, N étant reçu en paramètre). Cette séquence vous servira de suite de 0 (faux) et de 1 (vrai). Votre méthode renverra ce tableau de N booléens ainsi créé.
Écrivez également une méthode permettant d'afficher le contenu d'un tableau. Cette méthode pourra être utilisée pour vérifier certains algorithmes en considérant des tableaux de plus petite taille.

Méthodes de tests

Créez un ensemble de méthodes renvoyant une valeur booléenne selon si le test en question est vrai ou faux afin de réaliser les quatre tests sur votre ensemble de 20000 valeurs. Chacune des méthodes recevra en paramètre la séquence bits à tester ainsi que la taille de cette séquence.
Indications :

- Pour le monobit test, il vous suffit simplement de comptabiliser le nombre de valeurs à « true ».
- Pour le poker test, il vous faut créer un tableau de 16 éléments, chacun de ces éléments va ensuite représenter le nombre d'occurrences de la valeur numérique correspondant à son indice. Par exemple, la case 0 comptera le nombre d'occurrences de la valeur 0 (les 4 bits à 0), alors que la valeur de la case d'indice 10 (la 11ème case) va compter le nombre d'occurrences de 10 c'est à dire la valeur binaire 1010). Transformez votre suite de 4 valeurs binaires en un entier et indexez le tableau avec cet entier.
- Pour le run test et le long run test, la démarche est la même. Vous pourrez créer un tableau à deux dimensions (2x6 cases). La case [0][3] sera le nombre de suites de 4 bits à 0 et la case [1][2] le nombre de suites de 3 bits à 1. Vous enregistrerez la valeur du bit précédent pour savoir lorsque votre suite change et lorsque celle-ci change, vous avez la fin d'une suite pour laquelle vous notez la taille afin d'augmenter la bonne case du tableau. Pour le long run test, la méthode de recherche des suites est plus simple, mais vous vous limitez à chercher des suites de taille au moins 34. À noter que vous pouvez aussi combiner les deux méthodes pour plus de simplicité, en traitant le long run comme un cas particulier du run test (cas où l'on a 34 éléments consécutifs de même valeur), et renvoyer par exemple un tableau de 2 booléens, un pour chaque test.

Main

Écrivez la méthode main() appelant votre fonction de génération et vos méthodes de tests de la séquence générée. Les méthodes de tests recevront toutes la même séquence, vous ne générerez pas 4 séquences différentes avec votre générateur. Testez, à la fin du main, si l'ensemble des quatre tests est passé avec succès par le générateur et par conséquent si celui-ci est conforme ou non.

Utilisation du générateur de nombres aléatoires : le jeu de l'escargot

Ce jeu est très simple. Il se joue à deux avec un dé. Chacun lance le dé à son tour. Pour que l'escargot démarre, il faut faire un 6. Ensuite, vous avancez du nombre de mètres qui aura été décidé par le dé. Le premier qui a parcouru les 100 mètres a gagné.

Version simplifiée

Pour cette première version, vous initialiserez votre générateur aléatoire puis, pour chaque tour, vous tirerez un nombre aléatoire entre 1 et 6. Cette valeur de dé sera utilisée, en suivant la règle du jeu, pour faire avancer l'un ou l'autre des escargots.
Pour tirer un nombre entier entre une valeur *min* et une valeur *max*, regardez du côté de la fonction nextInt(int) de Random qui prend en paramètre un entier N et génère un entier entre 0 et N-1 inclus.

Version semi-aléatoire

La première version est tout de même complètement aléatoire, c'est à dire que vous n'avez aucun contrôle sur la valeur que peut prendre le dé (contrairement à un vrai dé avec lequel l'aléatoire est biaisé par la manière de le lancer). Pour changer ceci, à chaque tour, vous demanderez à l'utilisateur d'entrer une valeur entière (cf. le complément en fin de sujet de TP pour la saisie au clavier) N et vous appellerez votre générateur N fois et prendrez la dernière valeur pour faire avancer l'escargot.

Version réaliste

Pour faire une version plus réaliste, nous allons utiliser un timer (un objet qui va répéter des événements à une fréquence donnée). Le code que nous utilisons permet de faire évoluer la valeur du dé de 1 toutes les 150 ms. Une boîte de dialogue s'affiche à l'écran pour demander à l'utilisateur de cliquer sur OK, lorsque celui-ci le fait, votre programme prendra la valeur du dé actuelle. Cette valeur n'étant jamais affiché, vous avez un choix « aléatoire » cependant conditionné à l'appuie sur OK par l'utilisateur. Une simulation qui se rapproche plus de votre dé manuel que les précédentes.
Le code pour faire ceci n'étant pas simple à écrire, nous avons décidé de vous le fournir. Vous allez donc être confronté, maintenant, à l'utilisation d'un code compilé qui vous est donné avec sa documentation. Ce code est dans le fichier *ComplémentTP4.zip*. Vous allez extraire ce .zip dans le répertoire de votre TP4. Il contient un fichier TP4.jar et un dossier javadoc.
Dans Netbeans, dans votre projet, cliquez sur le bouton droit sur *Libraries* puis *Add Jar/Folder*. Sélectionnez le fichier TP4.jar que vous venez d'extraire. Recliquez ensuite sur *Libraries* avec le bouton droit et sélectionnez *Properties*. Vous verrez le fichier TP4.jar dans « Compile-Time libraries ». Cliquez sur celui-ci puis sur *Edit* et dans le champ javadoc, sélectionnez le répertoire javadoc que vous venez d'extraire.
Ensuite, en cliquant sur le + de librairies, vous verrez cette librairie TP4.jar. Vous pouvez consulter sa documentation en cliquant sur *Show Javadoc* (en cliquant sur le bouton droit sur TP4.jar).
Pour l'utiliser, dans votre Main.java, vous devez indiquer, au début du fichier après la ligne package, **import tp4.* ;**.
Une fois ceci fait, si vous déclarez une instance de l'objet défini par cette librairie **timersEtDialoguesTP4 t = new timersEtDialoguesTP4() ;** vous pourrez alors utiliser l'ensemble des fonctions fournies.
Vous verrez, en déclarant une nouvelle instance, qu'il vous affiche aussi la documentation javadoc. Lorsque vous utiliserez l'une des méthodes, cette documentation apparaîtra aussi pour la méthode en cours.

Utilisez maintenant cette classe pour créer un lancé de dé plus dépendant de l'utilisateur.

N.B. : Dans ce .jar, vous pouvez également consulter le .java que nous avons écrit.

Complément : La saisie au clavier

IMT Lille Douai – FI 1A

Lecture au clavier

Pour lire une valeur au clavier, la méthode la plus simple est d'utiliser un objet « Scanner ». L'extrait de code suivant permet de l'utiliser :

Listing 1 – Saisie au clavier d'un nombre entier

```
1  import java.util.Scanner;
2
3  public class Main {
4      public static void main(String[] args) {
5          int a;
6          System.out.println("Veuillez saisir un entier au clavier");
7          Scanner sc = new Scanner(System.in);
8          a = sc.nextInt();
9          System.out.println("L'entier saisie est : " + a);
10     }
11 }
```

La ligne 1 permet d'importer la classe des Scanner afin de l'utiliser par la suite. La ligne 7 va déclarer un Scanner qui va lire les données sur System.in, notre clavier. La lecture d'un entier est faite ensuite avec sc.nextInt().
Attention, ici le code ne contient pas de vérification de la validité de l'entrée. Si il y a un soucis lors de la saisie (que l'utilisateur ne saisit pas un entier mais un mot par exemple), alors votre programme s'arrêtera sur une erreur (une exception est levée).
Vous pouvez également mettre ceci dans une méthode de saisie de réels par exemple, ce qui vous donnera :

Listing 2 – Saisie au clavier d'un nombre entier

```
1  import java.util.Scanner;
2
3  public class Main {
4      public static void main(String[] args) {
5          double a = saisirDouble();
6          System.out.println("L'entier saisie est : " + a);
7      }
8
9      public static double saisirDouble() {
10         System.out.println("Veuillez saisir un entier au clavier");
11         Scanner sc = new Scanner(System.in);
12         return sc.nextDouble();
13     }
14 }
```