Chapitre 6

Nombres aléatoires en informatique

On peut se demander comment des machines prédictibles comme des ordinateurs peuvent engendrer de l'aléa. En réalité, la plupart des nombres aléatoires utilisés en informatique sont des *nombres pseudo-aléatoires*, au sens où ils sont engendrés de manière prédictible en utilisant une formule mathématique.

La raison de ce phénomène vient de ce qu'un algorithme est une suite d'opérations prédéfinies que l'on applique à des paramètres pour les modifier. Si l'on applique le même traitement aux mêmes paramètres, les résultats sont donc identiques. En ce sens, un algorithme est donc déterministe, à l'opposé de ce que l'on veut obtenir. Pourtant certaines opérations sont suffisamment imprévisibles pour donner des résultats qui semblent aléatoires. Les nombres obtenus sont alors dits pseudo-aléatoires.

Cela suffit dans de nombreux cas, même si ces nombres ne ressemblent pas tout à fait à ce que l'on obtient en lançant un dé ou effectuant des tirages des loterie. Pour "produire des nombres au hasard", on utilise donc souvent en informatique une suite de nombres déterminée, appelée *séquence pseudo-aléatoire* qui contient plusieurs centaines de milliers de nombres différents [1] et satisfait plusieurs propriétés statistiques, ce qui donne à penser, en regardant une partie de ces nombres, que ceux-ci sont des nombres complètement aléatoires. Par abus de langage, on dira que ces nombres sont aléatoires (et non seulement pseudo-aléatoires).

6.1 Génération de nombres aléatoires

La fonction `rand()` permet d'engendrer un nombre aléatoire. Lors du premier appel, la fonction retourne le premier nombre de la séquence pseudo-aléatoire. Ensuite, à chaque appel, elle retourne le nombre suivant dans la séquence. Ainsi, l'exécution du code :

```
int main()
{
  affichage("Premier nombre aléatoire : " , rand());
  affichage("Deuxième nombre aléatoire : " , rand());
  affichage("Troisième nombre aléatoire : " , rand());
  return 0;
}
```

affichera :

```
Premier nombre aléatoire : 1804289383
Deuxième nombre aléatoire : 846930886
Troisième nombre aléatoire : 1681692777
```

1. Cette séquence est une suite au sens mathématique du terme, c'est-à-dire que le $(i+1)^e$ terme est calculé en fonction du i^e terme.

Attention : Cette suite est déterminée et si l'on exécute plusieurs fois le programme, l'affichage sera le même à chaque fois !

Remarque : La séquence pseudo-aléatoire de la fonction `rand()` peut varier selon les compilateurs. Il est donc possible que lors de l'exécution du programme précédent, vous n'obteniez pas les mêmes valeurs. En revanche, à chaque exécution, ces nombres seront identiques.

Remarque : Le principe des séquences pseudo-aléatoires est commun à tous les langages de programmation. Cependant, les langages n'utilisent pas forcément tous la même séquence pseudo-aléatoire.

6.1.1 Entiers aléatoires dans un intervalle

Par défaut, les nombres retournés par la fonction `rand()` sont compris entre 0 et `RAND_MAX` (inclus), cette dernière étant une constante définie dans le langage correspondant à la plus grande valeur entière pouvant être stockée par le type `int`. Pour avoir une valeur comprise entre 0 (inclus) et `N` (exclus), on utilise alors l'opérateur modulo : `rand()%N`. Pour obtenir un nombre aléatoire dans l'intervalle $\{a, \ldots, b\}$, on engendre un nombre aléatoire entre 0 et $b - a$ (inclus) auquel on ajoute a. Ainsi, le code suivant permet d'afficher un nombre aléatoire entre 10 et 20 (inclus) :

```
1 int main()
2 {
3   affichage("Nombre aléatoire : " , rand()%11 + 10);
4   return 0;
5 }
```

► Sur ce thème : **Exercice 2, TD 5**

6.1.2 Nombres flottants aléatoires

Pour engendrer un nombre flottant aléatoire dans l'intervalle [0,1], une solution simple consiste à diviser le nombre retourné par la fonction `rand()` par la constante `RAND_MAX`. Si l'on souhaite tirer un nombre dans un intervalle différent, il faut alors multiplier le résultat obtenu par la taille de l'intervalle. Le code suivant affiche deux nombres flottants aléatoires, le premier compris entre 0 et 1 et le second compris entre 0 et 10.

```
1 int main()
2 {
3   affichage("Nombre flottant aléatoire entre 0 et 1 : ", (double) rand() / RAND_MAX);
4   affichage("Nombre flottant aléatoire entre 0 et 10 : ", 10.0 * rand() / RAND_MAX);
5   return 0;
6 }
```

Attention : Comme `RAND_MAX` et la valeur retournée par la fonction `rand()` sont de type `int`, il faut faire attention à ne pas avoir de division entière. Il faut donc transtyper l'un des deux nombres en `double`, soit explicitement, soit en le multipliant par une valeur de type `double`.

6.2 Différentes générations de nombres aléatoires

Tous les programmes que nous avons écrits jusqu'à maintenant ont un défaut majeur. Même s'ils engendrent des nombres qui paraissent aléatoires, les nombres sont les mêmes à chaque exécution du programme ! Ils ne sont donc pas aléatoires puisque l'on peut les prédire !

Pour pallier à ce problème, on indique au programme de ne pas toujours commencer la séquence par la même valeur. Pour cela, on utilise la fonction `srand()` prenant en paramètre une valeur de type `int` appelée *graine*. Cette graine sert à déterminer le nombre de départ dans la séquence pour

la fonction `rand()` [2]. Dans le programme suivant, on engendre deux nombres aléatoires. Comme la graine utilisée est 12, les nombres aléatoires engendrés sont alors 1687063760 puis 945274514.

```
int main()
{
  srand(12);
  affichage(rand() , " " , rand());
}
```

À retenir : Il faut appeler la fonction `srand()` une seule fois dans le programme.

Si l'on exécute plusieurs fois ce programme, les nombres engendrés seront toujours 1687063760 puis 945274514 ! Il faut donc modifier la graine à chaque exécution du programme. Or, on ne peut pas demander à l'ordinateur de choisir aléatoirement une graine différente !

Pour modifier la graine, on utilise la fonction `time(NULL)` retournant le nombre de secondes écoulées depuis le 1er janvier 1970. L'initialisation du générateur est alors faite à l'aide de l'instruction `srand(time(NULL))`. Si le programme est exécuté plusieurs fois (avec une seconde d'intervalle au minimum entre les exécutions), la graine est alors différente et les nombres retournés par la fonction `rand()` le sont aussi.

▶ Sur ce thème : **Exercices 1 et 3 à 5, TD 5**

2. En fait, le calcul du ie terme de la suite se fait en fonction de la graine et du terme précédent.

TD6 : Nombres aléatoires (Corrigé)

✓ Exercice 1 : Test de compréhension*

Question 1.1 : [Fonction rand] À quoi sert la fonction `rand()` ? Quel paramètre prend-t-elle en argument ?

Correction :

La fonction `rand()` retourne un entier (`int`). Cet entier est un élément de la séquence pseudo-aléatoire. C'est le premier nombre de cette série ou le nombre suivant le dernier nombre retournée par `rand()`. La fonction `rand()` ne prend aucun paramètre. ◇

Question 1.2 : [Fonction srand] À quoi sert l'instruction `srand(time(NULL))` ?

Correction :

Cette instruction sert à initialiser le générateur de nombres pseudo-aléatoires avec une valeur correspondant au nombre de secondes écoulées depuis le 1er janvier 1970. À chaque exécution, le générateur est initialisé avec une graine différente. Ceci permet à la fonction `rand()` d'engendrer des nombres différents à chaque exécution. ◇

Question 1.3 : [Vérification de code] Le programme suivant engendre-t-il correctement les nombres aléatoires ? Pourquoi ?

```
int main()
{
  int i = 0
  while(i < 10)
  {
    srand(time(NULL));
    affichage(rand());
              i++;
  }
  return 0;
}
```

Correction :

Tous les nombres engendrés par la fonction `rand()` sont les mêmes. En effet, la fonction `srand()` est appelée 10 fois au lieu d'une. Comme le programme s'exécute en moins d'une seconde, le générateur est initialisé 10 fois avec la même valeur. De plus, la fonction `rand()` retourne le premier terme de la série déterminé par la graine. Ces nombres sont donc tous les mêmes. ◇

✓ Exercice 2 : Somme de 10 nombres entiers aléatoires entre 1 et 10*

Écrire un programme affichant 10 nombres aléatoires entre 1 et 10 inclus et calculant la somme de ces nombres.

Correction :

```
int main()
{
  srand(time(NULL));
  int somme = 0;
  int nombre;
  int i = 0;

```

```
  while(i < 10)
  {
    nombre = rand()%10 + 1;
    affichage(nombre);
    somme += nombre;
    i++;
  }
  affichage("Somme = " , somme);
  return 0;
}
```

✓ Exercice 3 : Lancer de dés*

Le but de cet exercice est de créer des fonctions permettant de simuler des lancers de dés. Lancer un dé correspond, d'un point de vue informatique, à tirer un entier aléatoire entre 1 et 6 inclus.

Question 3.1 : Créer une fonction qui simule le lancer d'un dé et renvoie le résultat.
Correction :

```
int lancerDe()
{
  return 1 + rand()%6;
}
```

Question 3.2 : Créer une fonction qui simule le lancer d'un nombre quelconque de dés donné en paramètre et renvoie leur somme. Utiliser cette fonction pour afficher le résultat d'un lancer de deux dés puis de trois dés.
Correction :

```
int lancerPlusieursDes(int n)
{
  int somme = 0 ;
  int i = 0;
  while (i<n)
  {
    somme += lancerDe();
    i++;
  }
  return somme;
}

int main()
{
  srand(time(NULL));

  affichage("Lancer de deux dés : " , lancerPlusieursDes(2));
  affichage("Lancer de trois dés : " , lancerPlusieursDes(3));

  return 0;
}
```

Question 3.3 : Certains dés ont un nombre de faces différent de 6, comme certains dés utilisés pour les jeux de rôles. Écrire une fonction `lancerDe2` prenant en paramètre le nombre de faces du dé et simulant un lancer de ce dé.

Correction :

```
int lancerDe2(int nbFaces)
{
  return 1 + rand()% nbFaces;
}
```

Question 3.4 : Écrire une fonction qui simule le lancer d'un nombre quelconque de dés donné en paramètre, tous les dés ayant le même nombre de faces (également donné en paramètre), et renvoie leur somme. Utiliser cette fonction pour afficher le résultat de dix lancers d'un dé à dix faces.

Correction :

```
int lancerPlusieursDes2(int n, int nbF)
{
  int somme = 0 ;
  int i = 0;
  while (i<n)
  {
    somme += lancerDe2(nbF);
    i++;
  }
  return somme;
}

int main()
{
  srand(time(NULL));
  affichage("Lancer de dix dés à dix faces : " , lancerPlusieursDes2(10,10));
  return 0;
}
```

✓ Exercice 4 : Soldes aléatoires**

Pour les soldes, un magasin propose une formule originale. Lors du paiement en caisse, une réduction de 30, 50 ou 70% est appliquée sur le montant total. Le taux de réduction est choisi aléatoirement.

Question 4.1 : Écrire une fonction `taux()` retournant 30, 50 ou 70, la valeur étant choisie aléatoirement.

Correction :

```
int taux()
{
  return 30 + rand()%3 * 20;
}
```

Question 4.2 : Écrire le programme principal permettant de déterminer le prix à payer après la réduction. Le prix sera saisi par l'utilisateur et la réduction choisie aléatoirement.

Correction :

```
int main()
{
  srand(time(NULL));

```

```
  double prix;
  affichage("Prix ? ");
  saisie(prix);

  int reduction = taux();

  affichage("Bravo, vous avez obtenu un taux de " , 100 - reduction,
  "%, le prix de votre commande est ", reduction / 100. * prix , " euros.");
  return 0;
}
```

Exercice 5 : Nombre magique**

Le nombre magique est un jeu où l'utilisateur doit trouver un nombre entier aléatoire compris entre 1 et 100 inclus en au plus 10 essais. À chaque essai, le programme répond en indiquant si le nombre à déterminer est plus petit ou plus grand que la valeur saisie par l'utilisateur. Écrire un programme permettant de jouer à ce jeu.

Correction :

```
int main ()
{
  //Initialiser le générateur de nombres aléatoires
  //Ne faire qu'une seule fois dans le programme!!!!
  srand(time(NULL));

  int nombreAleatoire = 1 + rand()%100;;

  int nombre = -1;
  int tentative = 0;

  while ( nombre!=nombreAleatoire && tentative < 10 )
  {
    tentative ++;
    affichage("Saisissez un nombre : ");
    saisie(nombre);
    if (nombre < nombreAleatoire)
      affichage("Le nombre magique est plus grand");
    else if (nombre > nombreAleatoire)
      affichage("Le nombre magique est plus petit");
  }

  if ( nombre==nombreAleatoire)
    affichage("Vous avez trouvé le nombre magique en " , tentative , " coups");
  else
    affichage("Perdu");
  return 0;
}
```

TP6 : Nombres aléatoires (Corrigé)

Exercice 6 : Probabilité d'avoir 7 avec deux dés*

Question 6.1 : Écrire une fonction retournant le résultat d'un lancer de dés (à six faces).
Correction :

```
int de()
{
  return rand() % 6 + 1;
}
```

◇

Question 6.2 : Écrire une fonction retournant la somme de deux lancers de dés.
Correction :

```
int somme2des()
{
  return de() + de();
}
```

◇

Question 6.3 : On sait théoriquement que la probilité de faire 7 avec deux dés est de 1/6. Écrire un programme simulant 100 lancers de deux dés et calculant la fréquence (pourcentage) où le résultat d'un lancer est 7. Le résultat calculé est-il proche du résultat théorique ? Qu'en est-il en simulant maintenant non pas 100 mais 1 million de lancers ?
Correction :

```
int main()
{
  srand(time(NULL));

  int N = 100;
  int nbFois7 = 0;
  int i = 0;

  while(i < N)
  {
   if (somme2des()==7)
      nbFois7++;
     i++;
  }
  affichage(100. * nbFois7 / N, "%");

  return 0;
}
```

Pour simuler 1 million de lancers, il suffit de modifier la valeur de `N` à 1 milion. Plus le nombre de lancers simulés est grand, plus le résultat expérimental trouvé se rapproche de la valeur théorique. ◇

Exercice 7 : Jeu du 7**

Le jeu du 7 se joue à deux joueurs avec deux dés. Le jeu se joue en 5 manches. Pour chaque manche, les deux joueurs jouent à tour de rôle. Lors de son tour, le joueur lance les deux dés.

S'il fait 7, il marque 7 points puis s'arrête. Autrement, le joueur peut relancer les dés. À chaque relance, si le joueur fait 7, son tour est terminé et il marque 0 points. Autrement, il additionne le résultat du lancer de dés à son total. S'il décide de s'arrêter, il marque le total des points qu'il a obtenu avec ses différents lancers (s'il n'a pas fait 7 auparavant).

On reprendra les fonctions `de()` et `somme2des()` de l'Exercice 6, on pourra également utiliser la fonction `confirmation()` définie dans le Chaptre 5 : Fonctions II.

Question 7.1 : Écrire la fonction `tour()` simulant le tour d'un joueur.
Correction :

```
int tour()
{
  int total = 0;
  bool rejoue = true;
  int lance = 0;

  while (rejoue)
  {
    lance = somme2des();
    affichage("LANCER : ", lance);
    if (lance == 7)
    {
      affichage( "Vous venez de faire 7, c'est perdu !" );
      if (total==0)//Si c'est le premier lancer
        return 7;
      return 0; //Ce n'est pas le premier lancer, on retourne donc 0
    }
    total += lance;
    affichage( "Voulez-vous rejouer ? ");
    rejoue = confirmation();
  }
  return total;
}
```

Question 7.2 : Écrire la fonction `jouer()` permettant de jouer à une partie de 7. Cette fonction affichera uniquement le total réalisé par les deux joueurs à la fin des 5 manches et retournera 1 si le joueur 1 a réalisé un score plus grand que le joueur 2, 0 en cas d'égalité, ou -1 sinon.
Correction :

```
int jouer()
{
  int j1 = 0;
  int j2 = 0;
  int i = 0;

  while(i < 5)
  {
    affichage( "==================" );
    affichage( "TOUR DU JOUEUR 1 : " );
    affichage( "==================" );
    j1 += tour();
    affichage( "==================" );
    affichage( "TOUR DU JOUEUR 2 : " );
    affichage( "==================" );
    j2 += tour();
    i++;
  }

```

```
20   affichage( ) , "TOTAL : " );
21   affichage( "\t JOUEUR 1 : " , j1 );
22   affichage( "\t JOUEUR 2 : " , j2 );
23
24   if (j1==j2)
25     return 0;
26   if (j1 > j2)
27     return 1;
28   return -1;
29 }
```

Question 7.3 : Écrire le programme principal. Il devra permettre de jouer au jeu du 7 et devra afficher le gagant.
Correction :

```
1  int main()
2  {
3    srand(time(NULL));
4    int res = jouer();
5
6    if (res==1)
7      affichage( "JOUEUR 1 a gagné" );
8    else if (res==0)
9      affichage( "EGALITE" );
10   else
11   affichage(  "JOUEUR 2 a gagné" );
12
13   return 0;
14 }
```

Exercice 8 : Approximation de la valeur π à l'aide de la méthode de Monte-Carlo***

Le terme *méthode de Monte-Carlo*, ou méthode Monte-Carlo, désigne toute méthode visant à calculer une valeur numérique en utilisant des procédés aléatoires, c'est-à-dire des techniques probabilistes [3].

Une des illustrations de la méthode de Monte-Carlo est la détermination de la surface d'un lac. Supposons que l'on ait un lac dont on souhaite calculer la superficie. On détermine une zone contenant le lac dont on connaît l'aire (On peut choisir un carré dont on connaît le côté). Pour trouver la superficie du lac, on demande à une armée de tirer un certain nombre de boulets de canon et l'on regarde la proportion des projectiles tombés dans le lac. Si suffisamment de boulets de canon ont été tirés et si l'on suppose que les missiles ont été tirés aléatoirement dans la zone, alors cette proportion correspond à la proportion de l'aire du lac par rapport à celle de la zone. Ainsi, si N projectiles ont été lancés et X sont tombés dans le lac, alors la surface du lac correspond à X/N fois la surface de la zone.

On peut utiliser cette technique pour déterminer une valeur approchée de π. Considérons un quart de cercle de rayon 1 contenu dans un carré de côté 1 (*c.f.* figure 6.1). On connaît cette fois-ci l'aire du carré (1) représentant la zone et l'aire du quart de cercle ($\pi/4$) représentant le lac. À l'aide de la méthode de Monte-Carlo, on peut donc déterminer une valeur approchée de l'aire du quart de cercle. Cette valeur multipliée par 4 donne alors une valeur approchée de la valeur π.

L'algorithme est le suivant. On tire aléatoirement un grand nombre de points dans le carré (des points (x, y) où x et y sont des valeurs aléatoires dans $[0, 1]$). On compte le nombre de fois

3. Définition de Wikipedia. L'exemple du calcul de l'aire du lac provient également de Wikipedia.

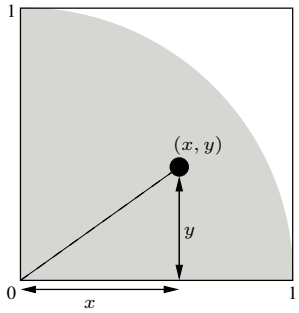

FIGURE 6.1 – Approximation de la valeur π par la méthode de Monte-Carlo

où les points appartiennent au quart de cercle, ce qui nous donne la proportion de l'aire du quart de cercle par rapport à l'aire du carré. On en déduit ensuite une valeur approchée de π.

Question 8.1 : Écrire une fonction `coordonnee()` retournant une coordonnée aléatoire dans l'intervalle $[0, 1]$.
Correction :

```
double coordonnee()
{
  return 1. * rand() / RAND_MAX;
}
```

◇

Question 8.2 : Écrire une fonction `carre()` prenant une valeur de type `double` en paramètre et retournant le carré de cette valeur.
Correction :

```
double carre(double a)
{
  return a * a;
}
```

◇

Question 8.3 : Écrire une fonction `estDansCercle()` prenant en paramètre un point du carré (donné par ses coordonnées en x et y) et retournant `true` si ce point est un point du cercle, ou `false` sinon.
Remarque : Pour déterminer si le point (x, y) est dans le cercle, il faut calculer la longueur du segment en pointillé sur la figure 6.1. Cela revient à calculer l'hypothénuse d'un triangle rectangle dont les segments adjacents à l'angle droit ont pour longueur respectives x et y (autrement dit, la valeur est donnée par $\sqrt{x^2 + y^2}$). Si cette valeur est inférieure ou égale à 1, alors le point (x, y) appartient au cercle.
Correction :

```
bool estDansCercle(double x, double y)
{
  return carre(x) + carre(y) <= 1;
}
```

◇

Question 8.4 : Écrire le programme permettant de déterminer une valeur approchée de π à l'aide de la méthode de Monte-Carlo.

Correction :

```
int main()
{
  srand(time(NULL));

  int nbPoints = 100000000;
  int nbPointsDansCercle = 0;
  int i = 0 ;

  while(i < nbPoints)
    {
    if (estDansCercle(coordonnee(),coordonnee()))
      nbPointsDansCercle++;
     i++;
    }
  affichage("Valeur approchée de pi avec " , nbPoints , " : ",
            4. * nbPointsDansCercle / nbPoints );
  return 0;
}
```