tout problème de texture), pour simuler la circulation automobile d'un réseau autoroutier, pour étudier des molécules de gaz dans un réservoir ou encore des interactions entre particules élémentaires, il faut des méthodes rapides produisant des millions de bits aléatoires, même s'ils ne le sont que superficiellement : un hasard faible suffit.

George Marsaglia, de l'Université de Floride, spécialiste des générateurs aléatoires, pense que dans bien des simulations même un médiocre hasard convient : «Un générateur aléatoire, écrit-il, c'est comme le sexe : quand c'est bon c'est merveilleux, et quand c'est mauvais c'est encore assez bon.»

Les fonctions *random* des langages de programmation procurent ce hasard peu coûteux en temps de calcul. On les utilise assez souvent avec satisfaction, mais mille exemples ont montré, dans le passé, qu'en cas d'expérimentations intensives ou cruciales de désagréables surprises se produisent. En 1993, des chercheurs de l'Université de Géorgie, explorant, par simulation, un problème de physique des solides, obtinrent des résultats en contradiction avec ceux obtenus par calculs directs. Une autre équipe découvrit un an plus tard que ces comportements aberrants étaient dus à l'utilisation de générateurs aléatoires insuffisants : le risque n'est pas très élevé en simulation, mais si l'on veut s'en garder, alors les difficultés commencent, et cette fois la règle est que, pour obtenir un hasard garanti, il ne faut surtout pas faire n'importe quoi.

LES GÉNÉRATEURS LES PLUS SIMPLES

Les générateurs congruentiels linéaires sont souvent utilisés pour les fonctions *random*. Leur mode de fonctionnement est le plus simple possible, et donc rapide.

On part de trois nombres entiers : M qui fixe l'intervalle où les nombres de la suite vont être engendrés, a et b qui sont pris entre 0 et $M - 1$. On choisit un quatrième nombre, la *graine* $x(0)$, elle aussi entre 0 et $M - 1$. On changera cette graine si l'on veut, avec le même générateur, obtenir une autre suite, ou on la reprendra si l'on souhaite obtenir exactement le même «hasard» lors d'un second essai, ce qui est souvent utile pour la mise au point d'un programme. La suite pseudo-aléatoire est donnée par les calculs suivants :

$x(1)$=reste de la division de $ax(0)+b$ par M
$x(2)$=reste de la division de $ax(1)+b$ par M
etc.

Chaque $x(i)$ est un nombre entier compris entre 0 et $M - 1$, qu'on utilise pour engendrer des bits aléatoires (0 ou 1) ; ce choix est fait, par exemple, en prenant la parité de $x(i)$, – 0 si $x(i)$ est pair, et 1 si $x(i)$ impair – (ou mieux, en calculant la parité de la somme des chiffres de $x(i)$, 0 si cette somme est paire, 1 si cette somme est impaire).

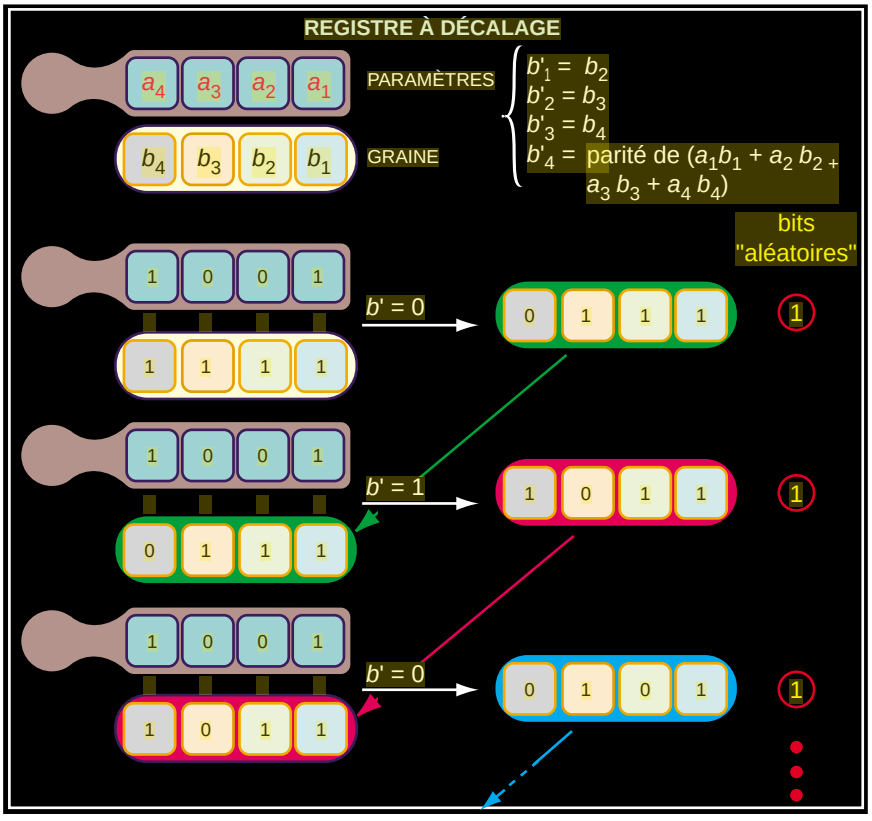
On peut transformer les entiers $x(i)$ pour donner des nombres réels. En prenant par exemple $u(i) = x(i)/M$, on obtient une suite de nombres réels $u(i)$ compris entre 0 et 1, dont la distribution est, en première approximation, uniforme : chaque sous-intervalle de $[y, z]$ de [0, 1] contient $x(i)$ un nombre de fois proportionnel à sa longueur $z - y$.

Les paramètres suivants ont été utilisés : $M = 2^{31} - 1$, $a = 16807$, $b = 0$. Avec $x(0) = 12345$, on obtient :
$x(1) = 207482415$ $u(1) = 0{,}0966165285$
$x(2) = 1790989824$ $u(2) = 0{,}8339946274$
$x(3) = 2035175616$ $u(3) = 0{,}9477024977$
etc.

La question est : comment obtenir des garanties sur le caractère aléatoire de la suite ? Persuadons-nous d'abord qu'une suite engendrée par le générateur précédent finira par tourner en rond. En effet, il n'y a qu'un nombre fini de valeurs possibles pour $x(i)$ (les entiers entre 0 et $M - 1$), et donc quand apparaît une valeur déjà obtenue – ce qui se produit nécessairement – la suite repasse par les mêmes valeurs, dans le même ordre, et ce, indéfiniment. Le nombre d'étapes pour revenir aux mêmes calculs est la période.

Cette ronde n'est pas si grave qu'elle en a l'air, et tout générateur fondé sur une formule du type précédent présentera ce défaut. Essayons d'apprivoiser cette ronde.

Il est clair qu'on ne saurait se satisfaire, pour la simulation, d'une suite dont la période serait trop petite. On souhaite que la suite ait une grande période, voire la plus grande possible, c'est-à-dire M. L'analyse mathématique du problème a conduit au résultat suivant, démontré au début des années 1960 : si les nombres b et M n'ont pas de diviseurs communs, si tout nombre premier qui



2. Le registre à décalage (à rétroaction linéaire) est défini par le mécanisme suivant. Une série de *N* bits est fixée a_1, a_2, a_3, a_4. Ce sont les paramètres, ici, 1 0 0 1. On initialise le générateur avec quatre bits b_1, b_2, b_3, b_4. (Ils constituent la *graine*. Ici, 1 1 1 1). À chaque étape, un nouveau bit b' est introduit : $b' = 0$ si ($a_1\ b_1 + a_2\ b_2 + a_3\ b_3 + a_4\ b_4$) est pair, 1 sinon (dans notre exemple, il suffit de compter le nombre de 1 parmi les bits b_1 et b_4 : s'il y en a 0 ou 2, b' vaut 0, sinon b' vaut 1). Le nouveau bit b' remplace b_4, lequel remplace b_3, lequel remplace b_2, lequel remplace b_1. Le bit b_1, lui, est utilisé comme bit aléatoire et disparaît du quadruplet. Dans notre exemple, on obtient successivement les quadruplets : 1111 0111 1011 0101 1010 1101 0110 0011 1001 0100 0010 0001 1000 1100 1110, puis la suite se répète. Les chiffres «aléatoires» engendrés sont obtenus en prenant le dernier de chaque série de quatre : 111101011001000...

3. LES DIFFÉRENTS HASARDS UTILISÉS

Hasard faible

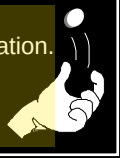
Bon mélange, uniformité, longue période, satisfaction des tests statistiques. Peut être produit par des algorithmes rapides. Utile en simulation.

Hasard moyen

Imprévisibilité pour un observateur ne disposant que de moyens de calcul réalistes.
Peut être produit (vraisemblablement) par algorithmes moyennement rapides. Utile en cryptographie.

Hasard fort

Imprévisibilité totale, incompressibilité, contenu maximum en information.
Ne peut jamais être produit par algorithme,
mais vraisemblablement par des moyens physiques.
Utile en cryptographie et en théorie de l'information.



divise M divise aussi $a - 1$, et si M n'est pas divisible par 4, alors la période du générateur est maximale, c'est-à-dire M. Si M est un multiple de 4, il faut que $a - 1$ le soit aussi pour que la période soit maximale

Autrement dit, si vous vous y prenez bien, en partant d'une valeur $x(0)$ quelconque (entre 0 et $M - 1$), vous n'y reviendrez qu'après être passé par tous les nombres entre 0 et $M - 1$. Cette promenade par tous les nombres possibles entre 0 et $M - 1$ n'assure pas que la suite est vraiment désordonnée, mais montre qu'on ne va pas osciller entre des valeurs trop proches, ce qui est déjà rassurant.

Il s'agit d'une propriété dite structurelle. Les spécialistes ont analysé d'autres propriétés structurelles identifiant des conditions mathématiques suffisantes pour qu'une suite engendrée par des générateurs congruentiels linéaires soit acceptable, c'est-à-dire produise des points assez équitablement distribués ou ne s'alignant pas trop quand on les représente dans des espaces de test.

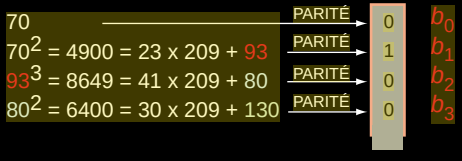
Dans une simulation utilisant n nombres pseudo-aléatoires, il est recommandé d'utiliser un générateur de période supérieure à n^2. On connaît des combinaisons de générateurs congruentiels linéaires engendrant des suites de période de 2^{200} et même plus. Récemment un générateur de période $2^{19937} - 1$ a été proposé. Un mot de prudence : il est moins risqué d'utiliser un générateur simple dont on connaît les propriétés qu'un générateur complexe pour lequel rien n'est démontré.

En plus des propriétés structurelles prouvées, on met en œuvre (sans cette fois chercher à faire de démonstrations mathématiques) des tests statistiques de toutes sortes portant sur les fréquences des différents chiffres et des suites de ces chiffres. Des jeux de tests ont ainsi été développés (en particulier, la série *Diehard* de George Marsaglia) et sont disponibles sur Internet pour qui souhaite tester la qualité d'une suite aléatoire : http://stat.fsu.edu/~geo/diehard.html

Reste que les spécialistes des générateurs aléatoires sont d'accord : même la démonstration de bonnes propriétés structurelles et le passage réussi à travers une volumineuse batterie de tests statistiques ne garantissent pas qu'un générateur donné sera satisfaisant pour une simulation donnée. Mathématiques et tests peuvent disqualifier une suite, mais pas en assurer la qualité.

4. LE PROCÉDÉ DE L. BLUM, M. BLUM ET M. SHUB (BBS)

Le générateur BBS est vraisemblablement satisfaisant en cryptographie. Soit n un produit de deux entiers premiers, chacun de la forme $4m + 3$. On prend comme graine un entier x sans facteur commun avec n et l'on calcule $x(0) = x^2 \bmod n$. On définit ensuite $x(i + 1) = x(i)^2 \bmod n$. La parité de $x(i)$ donne la suite pseudo-aléatoire de bits proposée par BBS.



Si la graine est choisie aléatoirement, et si l'hypothèse QRA est vraie (elle exprime que «trouver les racines carrées modulo n est un problème difficile») alors la suite de bits $b_0 b_1 \ldots b_m$ proposée par BBS est imprévisible (aucun algorithme rapide ne fait mieux que le hasard pour prédire le m-ième digit à partir des précédents). Il en résulte, d'après un résultat de A. Yao de 1982, que la suite de bits $b_0 b_1 \ldots b_m$ est indiscernable, par des tests fonctionnant en temps polynomial, d'une suite produite par une variable aléatoire équitable.

PRÉVISIBILITÉ

Cette situation est particulièrement dommageable en cryptographie, où l'on a souvent besoin de suites aléatoires et où le fait qu'une suite ne le soit pas vulnérabilise un procédé de codage. On raconte que, juste après la guerre, un algorithme de codage qui devait utiliser des clefs aléatoires toutes différentes fut utilisé deux fois de suite par les Russes avec la même clef, ce qui permit le déchiffrage des messages par les Américains et les conduisit à arrêter Klaus Fuchs, Julius et Ethel Rosenberg, Donald MacLean. L'officier russe responsable fut fusillé.

Les générateurs congruentiels linéaires sont depuis longtemps déconseillés en cryptographie. La raison tient à une série de résultats mathématiques qui indiquent qu'ils sont *prévisibles* : à partir de quelques points successifs de la suite $x(n)$, il est possible de calculer en un temps court les paramètres du générateur, et donc de prévoir son comportement futur avec une exactitude absolue, ce qui évidemment est contraire aux nécessités de la cryptographie. En cryptographie, conformément à l'idée que le hasard est imprévisible, il faut être certain que la suite aléatoire ne pourra pas être prédite *rapidement* à partir de la connaissance de quelques-uns de ses points.

«Rapidement», en informatique, signifie «ce qui peut être calculé en un temps plus petit qu'un polynôme de la taille des paramètres». Si la durée du calcul est inférieure à $n^3 + 2n^2 + 12$, où n est la taille des paramètres, on considère que le calcul est rapide. Si, en revanche, on ne sait faire un calcul qu'en un temps 2^n ou 3^n, etc., on considère que la longueur du calcul est inacceptable.

En utilisant cette idée, on arrive à une définition précise de la notion de «géné-