



Apprendre C++ avec Qt : Annexe 2 Utilisation de nombres pseudo-aléatoires

Il arrive assez fréquemment que l'on souhaite utiliser des nombres "aléatoires", tirés dans un intervalle donné. La librairie standard propose la fonction `rand()`, qui renvoie un nombre entier compris dans l'intervalle `[0, RAND_MAX]`. L'utilisation de cette fonction implique de prendre en charge plusieurs détails qui permettent de l'adapter aux besoins rencontrés.

Pour utiliser dans vos propres projets les fonctions décrites dans ce document, il vous suffit d'en copier le corps dans une fonction que vous aurez préalablement créée (il peut s'agir soit d'une fonction globale, soit d'une fonction membre d'une classe, selon vos besoins). Les opérations copier/coller étant très mal prises en charge par Adobe Acrobat Reader, le code de ces fonctions est également disponible sous la forme d'un [fichier texte](#).

1 - Amorçage du générateur

Si l'on ne souhaite pas obtenir la même séquence de nombres à chaque exécution du programme, il faut "amorcer" le générateur de nombres pseudo-aléatoires. Dans la librairie standard, c'est la fonction `srand()` qui est chargée de cet amorçage. La fonction `srand()` n'est pas plus capable de générer un nombre authentiquement aléatoire que ne l'est `rand()` : pour que l'amorçage effectué ne soit pas toujours le même (et donc ne conduise pas à la même séquence de nombres pseudo-aléatoires à chaque exécution), il convient de transmettre à `srand()` un paramètre dont nous ayons bon espoir qu'il sera différent lors de chaque exécution du programme. La solution traditionnellement adoptée est d'utiliser la fonction `time()`, qui renvoie le nombre de secondes séparant le 1er janvier 1970, 0 heures, du moment de l'appel de la fonction¹.

```
srand(time(NULL)); //amorçage de rand() à l'aide de l'heure courante
```

Les fonctions `rand()` et `srand()` sont déclarées dans le fichier `stdlib.h`, qui contient également la définition de la constante `RAND_MAX`. La fonction `time()` est, pour sa part, déclarée dans le fichier `time.h`. Selon le compilateur utilisé et le type de projet en cours, il peut être nécessaire de faire figurer explicitement les directives `#include` appropriées au début des fichiers dans lesquels ces fonctions et/ou cette constante sont utilisées.

L'amorçage du générateur de nombres pseudo-aléatoires ne doit être effectué qu'une seule fois au cours de l'exécution du programme, faute de quoi le caractère "aléatoire" des nombres renvoyés par `rand()` risque de disparaître.

2 - Tirage d'une série de valeurs

Une fois le générateur amorcé, on peut utiliser la fonction `rand()` pour obtenir un nombre. Si l'on dispose d'une fonction `affiche()` capable d'afficher à l'écran des valeurs entières, l'exécution du fragment de code suivant

```
//on suppose que srand() a déjà été appelée
1 int tirage;
2 int n;
3 for (n=0 ; n < 50 ; n = n + 1)
4 {
5     tirage = rand();
6     affiche(tirage);
7 }
```

se traduira par l'apparition d'une série du genre

¹ Il ne s'agit donc pas simplement de l'heure, qui risque d'être fâcheusement constante si notre programme est lancé automatiquement tous les jours à la même heure...

```
16682 795 29983 21753 15209 16146 29791 9228 13562 13890 6752 27027 12034
5265 5507 17471 4098 7880 28280 21710 1357 24124 5460 26321 6336 12207 17635
5811 20141 14705 12351 6391 24600 3274 11937 1729 25894 30365 22904 21400 978
17221 20664 20586 30434 31171 6517 24528 281 21019
```

3 - Contrôle de la plage de valeurs admissibles

La fonction `rand()` n'ayant aucun argument, il n'est pas possible de contrôler directement l'intervalle des valeurs possibles pour un tirage.

```
int unInt = rand(); //unInt est dans l'intervalle [0, RAND_MAX]
```

Dans la plupart des cas, un traitement s'impose donc, pour ramener la valeur renvoyée par `rand()` dans l'intervalle qui nous intéresse². La fonction suivante est capable de tirer un nombre compris entre des bornes qui lui sont communiquées par passage de paramètres :

```
1 int unIntEntre(int a, int b)
2 {
3     int minimum = a;
4     int intervalle = b - a;
5     if (a > b)
6     {
7         minimum = b;
8         intervalle = a - b;
9     }
10    return minimum + (rand() % intervalle);
11 }
```

Remarquez que, si `rand()` renvoie parfois `RAND_MAX`, la fonction `unIntEntre()` ne renvoie JAMAIS une valeur égale au paramètre `max`, ce qui signifie que, lorsque `min` vaut 0, `unIntEntre()` a exactement `max` valeurs différentes possibles.

Le fragment de code suivant est exemple d'utilisation de la fonction `unIntEntre()`. Il simule 30 lancers successifs d'un dé à 6 faces, dont les résultats sont stockés dans une `QValueList` :

```
//on suppose que srand() a déjà été appelée
1 QValueList <int> lesValeurs;
2 while (lesValeurs.count() < 30)
3     lesValeurs.append(unIntEntre(1,7));
```

L'exécution de cette séquence placera dans la liste une série du genre :

```
2 1 3 4 4 6 4 4 6 6 3 4 2 2 4 5 1 3 3 4 2 2 5 1 6 1 4 1 6 6
```

4 - Tirage sans remise

La fonction `rand()` n'est capable d'effectuer que des tirages indépendants les uns des autres. En d'autres termes, `rand()` fonctionne comme un dé qui aurait `RAND_MAX+1` faces : lors d'un lancé, la valeur obtenue lors du lancé précédent a autant de chances d'apparaître que n'importe quelle autre valeur.

Cette situation est très différente de celle rencontrée, par exemple, lorsqu'on distribue des cartes : la probabilité qu'un joueur reçoive une carte identique à une carte déjà attribuée à un autre joueur est alors nulle. Lorsqu'un programme exige un tirage de ce type (ce qu'on appelle un tirage "sans remise"), il est nécessaire d'en assurer explicitement la gestion.

La fonction suivante stocke dans une `QValueList` des valeurs différentes les unes des autres, tirées dans un intervalle dont les bornes lui sont (tout comme la `QValueList` et le nombre de valeur à tirer) communiquées par passage de paramètres :

² Oubliez tout de suite, si vous l'avez eu, l'idée de changer la définition de la constante `RAND_MAX`. Cette constante, tout comme `INT_MAX` ou `CLOCKS_PER_SEC`, n'est là que pour vous permettre d'écrire du code indépendant des caractéristiques du compilateur, et en aucun cas pour vous permettre de connaître ou de modifier les valeurs en questions.

```

1  bool tireSansRemise(QValueList<int> &liste, unsigned int nbVal, int a, int b)
2  {
3      liste.clear();
4      int min = a;
5      int max = b;
6      if (min > max)
7      {
8          min = b;
9          max = a;
10     }
11     if (nbVal > max - min) //mission impossible !
12         return false;
13     //fabrication d'une urne contenant les valeurs de min à max, dans l'ordre croissant
14     int i;
15     QMap <int, int> urne;
16     for(i = min ; i < max; ++i)
17         urne[i - min] = i;
18     //tirage sans remise
19     int limiteTirage = urne.count();
20     for (i=0 ; i < nbVal ; ++i)
21     {
22         int positionTiree = unIntEntre(0, limiteTirage);
23         liste.append(urne[positionTiree]);
24         urne[positionTiree] = urne[--limiteTirage];
25     }
26     return true; //signale que tout s'est bien passé
27 }

```

Cette fonction s'assure qu'une valeur donnée ne figurera pas deux fois dans la liste en simulant la présence d'une urne, dans laquelle les valeurs utilisées seraient tirées : chaque fois qu'une valeur est placée dans la liste, elle est également rendue indisponible en la remplaçant dans l'urne par celle dont la position va être rendue inaccessible lors du tirage suivant.

La fonction `tireSansRemise()` peut être utilisée pour obtenir un nombre quelconque de valeurs, toutes différentes, tirées d'un intervalle quelconque (à condition, bien entendu, que cet intervalle comporte au moins autant de valeurs qu'il en est demandé).

Dans l'exemple suivant, la liste est garnie avec dix valeurs tirées dans l'intervalle $[0, 32[$, ce qui pourrait s'apparenter à tirer 10 cartes dans un jeu de 32 :

```

1  //on suppose que srand() a déjà été appelée
2  QValueList <int> lesValeurs;
3  tireSansRemise(lesValeurs, 10, 0, 32);

```

L'exécution de cette séquence placera dans la liste une série du genre :

```
22 3 21 19 10 17 20 13 6 28
```

Remarquez que, si le nombre de valeurs demandées est égal à la taille de l'intervalle de tirage, appeler la fonction `tireSansRemise()` revient à mélanger les valeurs plutôt qu'à en choisir certaines :

```

QValueList <int> valeursDansLeDesordre;
tireSansRemise(valeursDansLeDesordre, 32, 0, 32);

```

produira une série du type

```
23 24 17 30 8 26 1 20 4 29 10 31 28 6 0 9 2 5 7 27 21 25 22 19 13 18 11 15
16 3 14 12
```