オブジェクト指向とゲームプログラミング

§　アルゴリズム編 － 第１回　乱数　§

完全に不規則な数列(乱数列)をコンピュータで発生させることはできません。しかし、プログラムを工夫することにより、完全な乱数に近づけることができます。

乱数

乱数とは、数が不規則に並んだ数列の一つ一つの数のことを指します。このような乱数は、本来予測不能です。しかし、コンピュータを使って乱数を発生させた場合、一定の規則で生成するので、次に出る数は完全に決まっていることになり、予測不能とはいえません。不規則な数列を計算により生成する乱数を疑似乱数と呼びます。疑似乱数は、ある一定の確率で起こるイベントを再現する以外に、情報の暗号化といった分野でも使用されています。

計算により乱数を発生させることの最大のメリットは、再現性があることです。初期状態が同じであれば、発生する乱数もまったく同じものが得られます。

疑似乱数を生成する方法はたくさんありますが、ほとんどの乱数生成プログラムは、「種」と呼ばれる数値を受け取り、それに対応する乱数列を発生させます。種を別のものにすることによって、簡単に異なる乱数列を得ることができます。

rand関数

Ｃ言語の標準ライブラリには、線形合同法により疑似乱数を生成するrand関数があります。rand関数は、０からRAND_MAXで定義される範囲内で、int型の疑似乱数を返します。rand関数は、srand関数とペアで使われます。srand関数は、疑似乱数を生成する「疑似乱数ジェネレータ」を初期化する関数です。この関数には、乱数列の元となる数値を与えます。この数値を元に、rand関数が疑似乱数を計算する仕組みになっています。srand関数に与えた数値が同じなら、rand関数で生成される乱数のパターンも同じになります。srand関数に与える数値を固定的なものにしたくない場合は、標準ライブラリのtime関数や、APIのGetTickCount関数またはtimeGetTime関数を利用します。これらの関数は、時間を取得するものなので、呼び出すたびに違った数値を取得することができます。

```
－乱数の初期化－
    srand(GetTickCount());

－乱数の発生－
    int    random  = rand();

    // ０～１５までの乱数を発生
    int    random2 = rand() % 16;
```

rand関数が生成する疑似乱数には、周期性があります。周期性とは、以前生成した数列と同じものが現れるというものです。rand関数の周期は、2^{31}です。およそ２１億回rand関数が呼び出されると、乱数のパターンが一周し、再び同じパターンが現れます。また、rand関数には、パターンに偏りがあることも知られています。

線形合同法

線形合同法は、整数の乱数を生成するもっとも基本的な方法です。適当な初期値x_0から出発し、

$$x_i = (a x_{i-1} + c) \bmod M \qquad i = 1, 2, 3, \cdots$$

という式で次々に$0 \leqq x_i < M$の範囲の値を生成します(※p mod qは、pをqで割った余り)。多くのコンパイラでは、Mの値に2^{32}(4,294,967,296)が使われています。ほとんどの環境で、余りを求める演算をわざわざ行なわなくても、掛け算と足し算の結果オーバーフローした部分を無視することによって、自動的にこの結果が得られるようになっています。

x_iの持つ重要な性質は、最下位ビットの周期が２、すなわち０と１が交互に現れ、その隣のビットの周期が４、さらにその隣のビットは８となり、最上位ビットの周期は4,294,967,296となることです。

VisualC++のrand関数は、次のようなコードで乱数を生成しています。

```
static long x = 1;
void srand(unsigned s) { x = s; }
int rand() { x = x * 214013 + 2531011; return (int)(x >> 16) & 32767; }
```

線形合同法x_iの下位16ビットを捨て、上位16ビットを右シフトしてそのうちの下位15ビットを返しています。したがって、最下位ビットでも周期は131,072になるので、線形合同法をそのまま使ったときの下位ビットの問題は起きません。しかし、乱数の返す数値の範囲が、0から32,767までしかなくなります。言い換えれば、精度が15ビットしかないということです。

rand関数の大きな欠点は、移植性がないことです。環境によって、最大値が2,147,483,647である場合もあれば、VisualC++のように32,767という場合もあります。異なるコンパイラでプログラムの移植するときに、ソースコードを書き換えなくてはならなくなります。また、違う種をsrand関数に与えても、同じ乱数列が発生する場合があります。例えば、srand(0)とsrand(0x80000000)で初期化された乱数列は、互いに同じものになります。

さらに、VisualC++では、srand関数に0～5,050を与えた場合、最初に返される乱数の最上位ビットが必ず0になるという問題もあります。

MT法

rand関数の欠点を解消するものに、MT(Mersenne Twister)法という方法があります。MT法は、M系列乱数と呼ばれるアルゴリズムを応用しています。M系列乱数とは、組み合わせ論から導き出された乱数で、各ビットが独立してk次均等分布しています。k次均等分布とは、k次元のベクトルを取ったときにk個がすべて0の場合を除いて、すべての組み合わせが全周期で等しい回数だけ現れるというものです。MT法では、６２３次元のベクトルを発生させても、全周期ですべての組み合わせのベクトルが現れます。

そのほか、MT法には、以下のような特徴があります。

- 従来の様々な生成法の欠点を考慮して設計されている
- 周期は$2^{19937}-1$と長周期である(10進法で６千桁以上)。
- 高次元均等分布を持つ(623次元超立方体の中に均等に分布することが証明されている)
- アルゴリズムが複雑なわりに、乗算や除算など遅い命令を使わないので、それなりに高速である

MT法の詳しい解説やソースコードは、以下のURLを参照してください。

http://www.math.keio.ac.jp/~matumoto/mt.html

ランダムな順列

n個の数v_0, v_1, v_2, …, v_{n-1}をランダムに切り混ぜるには、乱数xを生成し、v_{n-1}とv_xの値を交換します。これでv_{n-1}は固定し、v_0, v_1, v_2, …, v_{n-2}に対して同様の操作を行います。

以下の関数ShuffleTbl(vec[], n)は、要素数がnの配列vecを受け取り、ランダムに切り混ぜます。

```
// シャッフル
void ShuffleTbl(int vec[], const int n)
{
    for(int i = n - 1; i > 0; i--) {
        const int   x    = rand() % (i + 1);
        const int   work = vec[i];
        vec[i] = vec[x];
        vec[x] = work;
    }
}

void main()
{
    // 配列の初期化
    int   array[100];
    for(int i = 0; i < 100; i++)
        array[i] = i + 1;

    ShuffleTbl(array, 100);      // シャッフル
}
```

課　　題

以下のプログラムは、MT法を使って疑似乱数を生成する関数群(下記URL)をクラス化したものです。

http://www.math.keio.ac.jp/matumoto/CODES/MT2002/MTARCOK/mt19937ar-cok.c

従来のrand関数、srand関数に換えてこのクラスを使用してみましょう。

－MTRand.hpp－

```
/*
================================================================================================
                                オブジェクト指向ゲームプログラミング
     Programmed by Hibikino software. Copyright (c) 2004 Hibikino software. All rights reserved.
================================================================================================

  【対象ＯＳ】
     Microsoft  Windows95/NT4.0以降

  【コンパイラ】
     Microsoft  VisualC++ 6.0J ServicePack6

  【プログラム】
     MTRand.hpp
                MT乱数クラスヘッダ

  【履歴】
     * Version    1.00       2004/04/dd  hh:mm:ss

================================================================================================
*/

#pragma once

/*********************************************************************************************/
/*                                          MTRandクラス定義                                 */
/*********************************************************************************************/
class CMTRand {
public:
    CMTRand() { CMTRand::srand(DEFAULT_SEED); }
    CMTRand(const unsigned long seed) { CMTRand::srand(seed); }
    virtual ~CMTRand() {}

    void srand(const unsigned long inSeed);
    unsigned long rand();

    double frand () { return  CMTRand::rand()        * (1.0 / 4294967295.0); }      // [0, 1]
    double frand2() { return  CMTRand::rand()        * (1.0 / 4294967296.0); }      // [0, 1)
    double frand3() { return (CMTRand::rand() + 0.5) * (1.0 / 4294967296.0); }      // (0, 1)

    long rand31() { return CMTRand::rand() >> 1; }
    unsigned long rand(const unsigned long inRandMax) { return (unsigned long)(frand2() * inRandMax); }

private:
    void NextState();

    unsigned long MixBits(const unsigned long u, const unsigned long l) const
    { return (u & UPPER_MASK) | (l & LOWER_MASK); }

    unsigned long Twist  (const unsigned long u, const unsigned long v) const
    { return (MixBits(u, v) >> 1) ^ (v & 0x01 ? MATRIX_A : 0); }

    // 定数定義
    enum {
        N            = 624,
        M            = 397,
        MATRIX_A     = 0x9908b0df,
        UPPER_MASK   = 0x80000000,
        LOWER_MASK   = 0x7fffffff,
        DEFAULT_SEED = 5489
    };

    unsigned long    m_State[N];
    unsigned long*   m_pNext;
    int              m_Left;
};
```

―MTRand.cpp―

```
/*
================================================================================================
                                オブジェクト指向ゲームプログラミング
    Programmed by Hibikino software. Copyright (c) 2004 Hibikino software. All rights reserved.
================================================================================================

  【対象ＯＳ】
    Microsoft  Windows95/NT4.0以降

  【コンパイラ】
    Microsoft  VisualC++ 6.0J ServicePack6

  【プログラム】
    CMTRand.cpp
                MT乱数クラス

  【履歴】
    * Version    1.00       2004/04/dd  hh:mm:ss

================================================================================================
*/

/*********************************************************************************************/
/*                                      インクルードファイル                                 */
/*********************************************************************************************/
#include "MTRand.hpp"

/*********************************************************************************************/
/*                                        乱数初期化                                         */
/*********************************************************************************************/
void CMTRand::srand(const unsigned long inSeed)
{
    m_State[0] = inSeed & 0xffffffff;   // 種設定
    for (int i = 1; i < N; i++) {
        m_State[i]  = (1812433253 * (m_State[i - 1] ^ (m_State[i - 1] >> 30)) + i);
        m_State[i] &= 0xffffffff;
    }
    m_Left = 1;
}

/*********************************************************************************************/
/*                                        乱数発生                                           */
/*********************************************************************************************/
unsigned long CMTRand::rand()
{
    if(--m_Left == 0)
        NextState();

    unsigned long   random = *m_pNext++;

    random ^= (random >> 11);
    random ^= (random <<  7) & 0x9d2c5680;
    random ^= (random << 15) & 0xefc60000;
    random ^= (random >> 18);

    return random;
}

/*********************************************************************************************/
/*                                        次要素設定                                         */
/*********************************************************************************************/
void CMTRand::NextState()
{
    unsigned long*   pState = m_State;

    m_Left   = N;
    m_pNext = m_State;

    int   i;
    for(i = N - M + 1; --i > 0; pState++)
        *pState = *(pState + M) ^ Twist(*pState, *(pState + 1));

    for(i = M; --i > 0; pState++)
```

```
        *pState = *(pState + (M - N)) ^ Twist(*pState, *(pState + 1));
    *pState = *(pState + (M - N)) ^ Twist(*pState, m_State[0]);
}
```

CMTRand(MT法による疑似乱数列の生成)

－機能説明－

MT法による疑似乱数列を生成するクラスです。

－クラスメンバ－

メンバ	説明
CMTRand()	CMTRandオブジェクトを生成します
CMTRand(種)	CMTRandオブジェクトを「種」を用いて初期化します
srand(種)	乱数列を「種」を元に初期化します
rand()	０～4,294,967,295(２^{32}－１)の乱数を返します
rand(最大値)	０から最大値－１の乱数を返します
rand31()	０～2,147,483,647(２^{31}－１)の乱数を返します
frand()	０以上～１以下の乱数を実数で返します
frand2()	０以上～１未満の乱数を実数で返します
frand3()	０より大きく１より小さい乱数を実数で返します

応用問題　CMTRandクラスをJavaで作成しましょう。