

mt.cm

```
/*  
  
    Copyright (c) 2012–2015 Seppo Laakko  
    http://sourceforge.net/projects/cmajor/  
  
    Distributed under the GNU General Public License, version 3 (GPLv3).  
    (See accompanying LICENSE.txt or http://www.gnu.org/licenses/gpl.html  
    )  
  
*/  
  
// Copyright (c) 1994  
// Hewlett–Packard Company  
// Copyright (c) 1996  
// Silicon Graphics Computer Systems, Inc.  
// Copyright (c) 2009 Alexander Stepanov and Paul McJones  
  
// Copyright (C) 1997 – 2002, Makoto Matsumoto and Takuji Nishimura,  
// All rights reserved.  
  
// Mersenne Twister pseudo random number engine.  
// See: http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/MT2002/CODES/  
// mt19937ar.c  
  
using System.Collections;  
  
namespace System  
{  
    public static class MT  
    {  
        private const int n = 624;  
        private const int m = 397;  
        private const uint matrixA = 0x9908b0dfu;  
        private const uint upperMask = 0x80000000u;  
        private const uint lowerMask = 0x7fffffffu;  
  
        static nothrow MT()  
        {  
            InitWithRandomSeed();  
        }  
        public static nothrow void InitWithRandomSeed()  
        {  
            uint seed = get_random_seed_from_system();  
            Init(seed);  
        }  
//      Note: In general you do not have to call the Init function,  
//      because the static constructor calls it with random seed by  
//      default.  
//      Only if you want predictable sequence of pseudo random numbers,
```

```

//      call Init.
public static nothrow void Init(uint seed)
{
    mt[0] = seed;
    for (mti = 1; mti < n; ++mti)
    {
        mt[mti] = 1812433253u * (mt[mti - 1] ^ (mt[mti - 1] >> 30
            u)) + cast<uint>(mti);
    }
    mag[0] = 0u;
    mag[1] = matrixA;
}
public static nothrow uint GenRand()
{
    uint y = 0u;
    if (mti >= n)
    {
        int kk;
        for (kk = 0; kk < n - m; ++kk)
        {
            y = (mt[kk] & upperMask) | (mt[kk + 1] & lowerMask);
            mt[kk] = mt[kk + m] ^ (y >> 1u) ^ mag[cast<int>(y & 0
                x01u)];
        }
        for (; kk < n - 1; ++kk)
        {
            y = (mt[kk] & upperMask) | (mt[kk + 1] & lowerMask);
            mt[kk] = mt[kk + (m - n)] ^ (y >> 1u) ^ mag[cast<int>
                >(y & 0x01u)];
        }
        y = (mt[n - 1] & upperMask) | (mt[0] & lowerMask);
        mt[n - 1] = mt[m - 1] ^ (y >> 1u) ^ mag[cast<int>(y & 0
            x01u)];
        mti = 0;
    }
    y = mt[mti++];
    y = y ^ (y >> 11u);
    y = y ^ ((y << 7u) & 0x9d2c5680u);
    y = y ^ ((y << 15u) & 0xefc60000u);
    y = y ^ (y >> 18u);
    return y;
}
private static int mti;
private static uint[n] mt;
private static uint[2] mag;
}

public nothrow uint Rand()
{
    return MT.GenRand();
}
}

```