**Random vs Pseudorandom numbers**

Inspired by my last journal entry about Perlin Noise, I will explore more on the problem of generating random noise or random numbers in this week journal entry. (Link to previous entry at the end)

**Introduction:**

Random numbers are a set numbers that can be pick out from, with the distribution of each number is uniform across all range of possible values (**uniform distribution**), and most importantly **impossible to predict** the future values based on the past ones.

However, there is no real way to generate such “true” sequence of random numbers without tapping into the “randomness” of our nature like rolling dice, radioactive decay, surround noise… and even if we can generate those sequences (which we could, but it’s very slow) with those “randomness”, how do we truly know if it is really random.

Instead of trying to come up with a true random generator, what we could do is generate a set of pseudorandom numbers, which mean a set of numbers that is random enough that most people won’t notice until it generates over a long period of time.

**Pseudorandom Number:**

We can generate pseudorandom numbers sequence by using some sort of mathematical function to do so. Since these numbers are generate by some pre-deterministic mathematical processes, we cannot say that these numbers are truly random.

For algorithms to generate pseudorandom numbers to be considered good enough, we need to be able to determine the **period** (how long it would take for the number to repeat) or not at all (If it won’t repeat, then we might have a true random generator).

A few algorithms that we can use are:

**The Middle-Square method:**

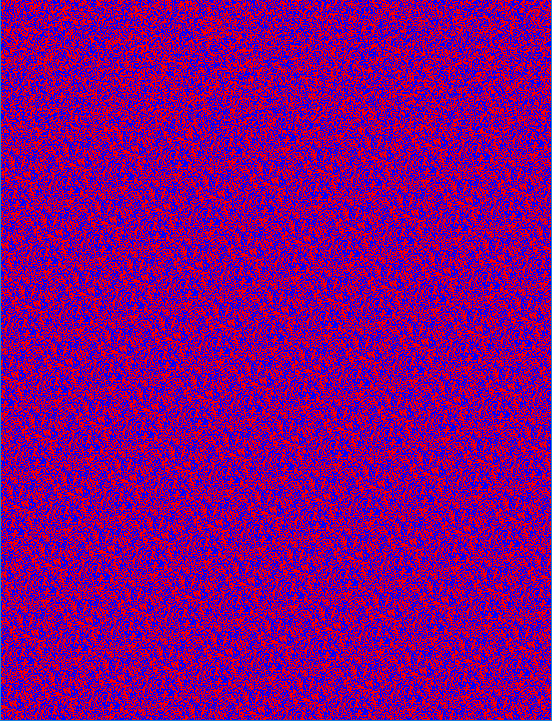
The middle square method is one the first pseudorandom numbers generator algorithms that was developed by von Neumann. The algorithms:

* Start with a n-seed number.
* Square it to obtain a 2n-digit number. Adding leading zero if necessary.
* Take the middle n digits as the next n-seed number.
* Repeat.

Let try a couple of seed: start with 1049

1004 80 64 40 16 2 0 0 0

After 6 iterations, we ran into problem of repeating 0 which isn’t really random after all.



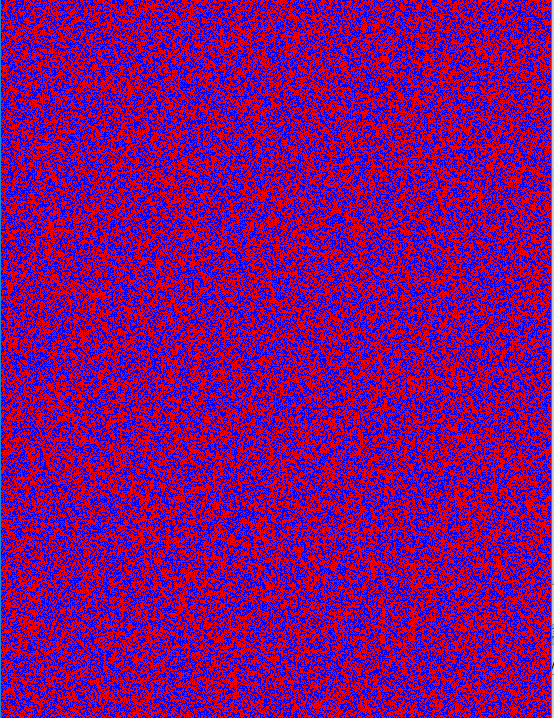
generated with the seed of 1234567890

**The Linear Congruence Method:**

This method is much more sophisticated, and reliable in a lot of cases, although **not perfect**.

The algorithms is: Xn+1 = (a ∙ Xn + c) mod m [given a seed value of X­­­n and integers a, c, and m]

Some numbers yield better result than other. A very good set of number can be:

M = 2^32, a = 1103515245, c = 12345, Xn = 1 which used by GCC compiler.

The reason this method is called Linear because when we plot the output values vs the seed values, it forms a linear function, and also the formula itself maybe a giveaway (mx+b).

**What are the uses?**

A lot applications use pseudorandom number generator like computer videogame, cryptography, sciences, arts, gambling…

Examples of video games application can be random terrain generation like my previous journal entry, or random shuffle of card, gambling …

With cryptography, it is very important to not use any of the algorithms above since it will start to break down at some point\*

**Demonstration:**

The program I used, is included in the zip file.

**Reference:**

Source: <https://github.com/Spyispie120/Random-Number-Generator>

Previous journal: <https://github.com/Spyispie120/2DMapGenerator>

<https://www.random.org/analysis/>

<http://www3.nd.edu/~mcbg/tutorials/2006/tutorial_files/randomNum/howItworks.html>

<https://en.wikipedia.org/wiki/Linear_congruential_generator>

<https://cs.gmu.edu/~sean/research/> Mersenne Twister