Brett Plemons

IT 312 Software Development with C++.NET

22 September 2023

4-4 Journal: Progress on Final Project

As with last week, the main goal of this project is to build a game that revolves around dice. As such, the most critical aspect of the game is the quality of the dice roles. To this end, I have done some research and came up with three different options:

1. Mersenne Twister engine provided by the C++ STL, `std::mt19937`. This generator has the following characteristics:
   1. Deterministic: given the same seed, it will produce the same sequence of numbers.
   2. Pseudorandom: It uses an algorithm to generate a sequence that only approximates true randomness.
   3. Period: It has a very long period (2^19937-1 iterations), meaning it takes a long time before the sequence repeats.
   4. Performance: It’s efficient and typically fast enough for simulation and modeling.
2. Using a “True Random” generator like the C++ STL Random Device engine `std::random\_device,` while the quality of the randomness is much higher than most other methods, it is not as performant. Here are its characteristics:
   1. Nondeterministic: It tries to generate true random numbers, usually based on hardware, so running the code multiple times should produce different sequences.
   2. True Random: It doesn’t use an algorithm to generate numbers (or if it does, the algorithm has some level of random input).
   3. Period: It doesn’t have a period because it isn’t pseudorandom
   4. Performance: Usually slower than pseudorandom number generators because generating true randomness involves more complex operations like measuring electronic noise.
3. Lastly, using an XOR-Shift Algorithm. It has the following attributes:
   1. Fast: The operations used in XOR-Shift are XOR and bit shifts, which are extremely fast.
   2. Pseudorandom: Like many PRNGs, XOR-Shift is deterministic, producing the same sequence of numbers when initialized with the same seed.
   3. Simple: The code is straightforward and can be implemented in a few lines.
   4. Fixed Period: Depending on the initial state and specific variant, an XOR-Shift generator can have a fixed period, after which the sequence repeats.
   5. Small state space: Typically only takes up 32 or 64 bits.
   6. The quality is good, but not cryptographic, but then again, none on this list are.
   7. Linearity: The algorithm is linear, meaning that you can predict future numbers if you know enough consecutive numbers in the sequence.

I will provide my C++ code in a separate file showing my test implementations. I decided to go with the Random Device generator as the scale of this application is relatively small. With my tests using six-sided dice, the quality was the best, and the speed variance was minimal.