Universal RNG (URng)

Universal RNG (URng) is a next-generation, high-performance pseudorandom number generator (PRNG) suite for Unity. It combines the rigorous algorithm design of Rust with the power of Unity's Burst Compiler and SIMD to deliver unparalleled speed and reliability. Whether you are generating infinite voxel terrains, managing thousands of active particles, or running complex economic simulations, URng provides the performance headroom you need.

🚀 Key Performance Metrics (N=100M)

🥇 **Pcg32**: 79.5 ms (~11.0x faster than System.Random) 🥈 **Sfc64**: 82.0 ms (~10.6x faster) 🥉 **Mt19937**: 88.4 ms (~9.9x faster)

Benchmarks performed in Unity with Burst enabled.

✨ Why Universal RNG?

Zero-Copy Memory Safety: Utilizes modern C# `Span<T>` APIs for zero-allocation bulk generation. **Burst & Job System Ready**: Every algorithm is handwritten and optimized to be fully compatible with Unity's Job System. **Rust-Native Performance**: Leveraging a Rust-based logic core, ensuring high statistical quality and cryptographic-inspired robustness. **Automatic Buffer Management**: Efficiently handles large-scale data without the overhead of the C# Garbage Collector.

💡 Why SplitMix64 for Initialization?

Most high-quality PRNGs, such as **Mersenne Twister**, require a large internal state to be initialized. Providing a simple seed isn't enough to fill this state with high-entropy data.

We use SplitMix64 to expand a single 64-bit seed into the full internal state. It is a fast, deterministic, and fixed-increment generator that ensures even a "weak" seed (like 1) is transformed into a robust, well-distributed starting point for the main generator.

🛠️ Compatibility

Unity Version: 2021.3+ (Requires .NET Standard 2.1) **Platforms**: Windows, macOS, Linux, Android, and iOS. **Dependencies**: Uses high-performance native binaries with seamless C# wrappers.

📜 Sample Code

```csharp
using Cet.Rng;

using var mt = new Mt19937(1, 1024);
Debug.Assert(mt.Next() == 244660247);
Debug.Assert(mt.NextU32s(3).SequenceEqual(new uint[] { 1183135446, 982747242,
221797415 }));

using var sfc = new Sfc64(1);
```

```
Debug.Assert(sfc.Next() == 4613303445314885483);
Debug.Assert(sfc.NextU64s(3).SequenceEqual(new ulong[] { 9250377774926580674,
11474594222839663884, 10182195164632316717 }));

using var pcg = new Pcg32(1);
Debug.Assert(pcg.Next() == 4164751464);
Debug.Assert(pcg.NextU32s(3).SequenceEqual(new uint[] { 4202279039, 2817093042,
1081118559 }));
```

"*Stop letting random number generation be the bottleneck of your simulation. With URng, you can focus on building your world while we handle the heavy lifting at 80ms.*"

Ready for production deployment! 🚀