🎲 Random Toolkit 🎲

Welcome to Random Toolkit! This asset provides a number of tools and benefits for you to use in your games.

1. It implements 2 new random number generators which improve upon Unity's built in one.
    a. Xorshift1204* and Mersenne Twister.

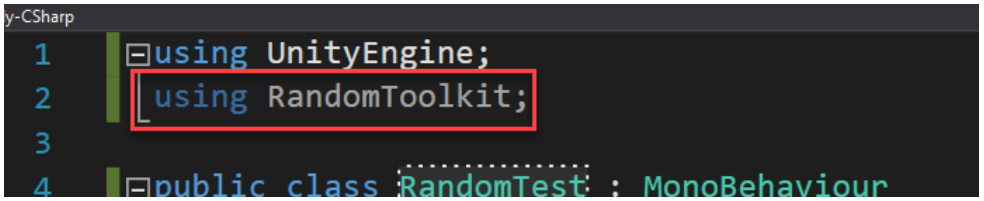
2. It provides a large number of improved and new methods for utilizing randomness.
    a. Randomized array and list elements, weighted randomness, colors, points in a range of 3D and 2D shapes, and uniform rotations.

3. An editor window to implement randomness in your workflow.
    a. Offsetting the position, rotation, and scale of selected GameObjects.

Quick Start Guide

If you're looking to jump into using Random Toolkit, just follow these steps.

1. First, make sure you are using the **RandomToolkit** library.



2. Next, create your random number generator object. This is where we'll pull random numbers from and access the range of functions as listed in the Scripting API section.
    a. **Xorshift1024*** - a more varied and strong rng algorithm.
    b. **Mersenne Twister** - a more varied and strong rng algorithm.
    c. **Unity** - Xorshift128*, which Unity's rng is a variation of.
    d. **System** - a wrapper for System.Random.

```
// New random number generators
private RTXorshift1024Star rng0 = new RTXorshift1024Star(123);
private RTMersenneTwister rng1 = new RTMersenneTwister(123);

// Existing random number generators
private RTUnity rng2 = new RTUnity(123, 456);
private RTSystem rng3 = new RTSystem(123);
```

3. Here are some quick functions to get you started.

```
// Equivalent to Random.value
rng.NextFloat();

// Equivalent to Random.Range()
rng.Range(30, 180);

// Change the rng's seed
rng.Seed(10983);
```

Intro to Random Number Generators

You may be thinking: why do I need a different random number generator? Or: how does one even work? Let's go over how these RNG's work, and some common terminology you might come across.

1. **Start with a seed**
   - The RNG begins with an initial number called a seed.
   - Think of it like planting a seed in the ground: from the same seed, you always get the same plant.
   - In RNG terms: if you start with the same seed, you get the same sequence of random numbers.

2. **Use a formula**
   - The RNG applies a mathematical formula to the seed to get the next number.
   - This formula usually involves things like:
     - Multiplying the current number.
     - Adding another number.
     - Taking the remainder after dividing (this keeps the number from getting too big).

3. **Produce the output**
   - After applying the formula, the RNG:
     - Keeps the new number to use as the next seed.
     - Gives you part (or all) of that number as the random number.

4. **Repeat**
   - Every time you ask for a new random number, the RNG:
     1. Uses the last number as the new seed.
     2. Runs it through the formula again.
     3. Gives you the next number.

Comparing Generators

Here are the different RNG's that are implemented in Random Toolkit:

Name	State Size	Period	Notes
UnityEngine.Random	128 bits	$2^{128}-1$	Unity uses a variant of the Xorshift128 RNG.
System.Random	48 bits	2^{31}	This is a Linear Congruential Generator (LCG).
Xorshift1024*	1024 bits	$2^{1024}-1$	Very long period, good quality.
Mersenne Twister	2.5 KB	$2^{19937}-1$	Very long period, large state size, good quality.

Terminology

State size? Period? What do those things mean? Let’s go over some terminology:

- **RNG** - Random number generator.
- **Seed** - The initial number you give the RNG to start the sequence.
- **Sequence** - The ordered list of numbers an RNG produces after it’s been seeded.
- **State** - The internal memory the RNG keeps track of to know where it is in the sequence.
- **State Size** - The amount of data needed to produce each next number.
- **Period** - The number of random numbers an RNG can generate before it repeats the same sequence.
- **Pseudorandom Number Generator (PRNG)** - A deterministic algorithm that produces numbers which look random.
- **Linear Congruential Generator (LCG)** - One of the simplest PRNG’s. It’s fast but has poor statistical quality, and a short period.

PRNG Families:

- **Xorshift** - Uses bitwise XOR and shifts.
- **Xoshiro** - Improved successors to Xorshift.
- **PCG** - Uses small state and permutation for good distribution.
- **WELL** - A family designed to improve on MT.
- **Philox** - Counter-based RNG suitable for parallel computing.
- **Mersenne** Twister - Long period and good statistical quality.

New Number Generators

Random Toolkit features 4 different RNG's for you to use in your projects.

The first 2: RTUnity and RTSystem, are essentially wrappers for the existing RNG's you have access to.

- An important difference being, RTUnity is a rewrite of Unity's RNG, utilizing the Xorshift128* algorithm.

That leaves RTXorshift1024Star and RTMersenneTwister.

Xorshift1024*

Name	State Size	Period	Notes
Xorshift1024*	1024 bits	$2^{1024}-1$	Very long period, good quality.

This PNRG uses bitwise XOR and shift operations to produce sequences of random numbers.

- The **state size** is quite large at 1024 bits. That's 700% larger than Unity's, but since we're working with bits here, it's not much in the grand scheme of things.
- The **period** is also very large, a number with 309 digits. It's safe to say you'll never get around to repeating the sequence.

```
private RTXorshift1024Star rng = new RTXorshift1024Star(123);
```

Mersenne Twister

Name	State Size	Period	Notes
Mersenne Twister	2.5 KB	$2^{19937}-1$	Very long period, large state size, good quality.

This PNRG uses bitwise XOR and shift operations to produce sequences of random numbers.

- The **state size** is very large at 2.5 KB. That's 160,000% larger than Unity's, but since we're still working in only a couple KB's here, it's no worry.
- The **period** is also very large, a number with 6002 digits. You'll never get around to repeating the sequence.

```
private RTMersenneTwister rng = new RTMersenneTwister(123);
```

Scripting API

`uint NextUInt ()`
Returns the next unsigned integer in the sequence.

`int NextInt ()`
Returns the next integer in the sequence.

`float NextFloat ()`
Returns the next float in the sequence.

`double NextDouble ()`
Returns the next unsigned integer in the sequence.

`ulong NextULong ()`
Returns the next ulong in the sequence.

State Serialization

`byte[] SerializeState()`
Serializes the RNG's state into a byte array, which can be saved to disk or sent over a network.

`void LoadState(byte[] serializedState)`
Deserializes a byte array into an RNG state which is then applied.

Range

`int Range (int min, int max)`
Returns a random integer value between the min and max, with the max being exclusive.

`float Range (float min, float max)`
Returns a random float value between the min and max.

`double RangeDouble (double min, double max)`
Returns a random double value between the min and max.

`long RangeLong (long min, long max)`
Returns a random long value between the min and max.

`Vector3 RangeVector3 (Vector3 min, Vector3 max)`
Returns a random Vector3 value between the min and max.

`Vector2 RangeVector2 (Vector2 min, Vector2 max)`
Returns a random Vector2 value between the min and max.

Weighted Values

`float WeightedValue (AnimationCurve curve)`
Same as the NextFloat() function, but weighted against an animation curve.

`int WeightedInt (AnimationCurve curve, int exclusiveMax)`
Returns a random integer from 0 to exclusiveMax - 1, but weighted against an animation curve.

`int WeightedInt (int[] weights)`
Returns a random integer from 0 to weights.Length - 1, weighted against the value of each element. e.g. with [1, 3, 44, 5], element 2 will return the majority of the time.

`int WeightedInt (List<int> weights)`
Returns a random integer from 0 to weights.Count - 1, weighted against the value of each element.

`int WeightedInt (float[] weights)`
Returns a random integer from 0 to weights.Count - 1, weighted against the value of each element.

`int WeightedInt (List<float> weights)`
Returns a random integer from 0 to weights.Count - 1, weighted against the value of each element.

Array Methods

`void Shuffle<T> (T[] array)`
Randomizes the order of array elements.

`T RandomElement<T> (T[] array)`
Returns a random element from an array.

`T RandomElement<T> (T[] array, int[] weights)`
Returns a random element from an array with weighted selection.

`T RandomElement<T> (T[] array, AnimationCurve weightCurve)`
Returns a random element from an array with weighted selection.

`T[] RandomElements<T> (T[] array, int count)`
Returns an array (length of count) of randomly selected elements.

`T[] RandomElements<T> (T[] array, int[] weights, int count)`
Returns an array (length of count) of randomly selected elements with weighted selection.

`T[] RandomElements<T> (T[] array, AnimationCurve weightCurve, int count)`
Returns an array (length of count) of randomly selected elements with weighted selection.

`T[] RandomElementsUnique<T> (T[] array, int count)`
Returns an array (length of count) of randomly selected, non-repeating elements.

List Methods

`void Shuffle<T> (List<T> list)`
Randomizes the order of list elements.

`T RandomElement<T> (List<T> list)`
Returns a random element from a list.

`T RandomElement<T> (List<T> list, int[] weights)`
Returns a random element from a list with weighted selection.

`T RandomElement<T> (List<T> list, AnimationCurve weightCurve)`
Returns a random element from a list with weighted selection.

`List<T> RandomElements<T> (List<T> list, int count)`
Returns a list (length of count) of randomly selected elements.

`List<T> RandomElements<T>(List<T> list, int[] weights, int count)`
Returns a list (length of count) of randomly selected elements with weighted selection.

`List<T> RandomElements<T> (List<T> list, AnimationCurve weightCurve, int count)`
Returns a list (length of count) of randomly selected elements with weighted selection.

`List<T> RandomElementsUnique<T> (List<T> list, int count)`
Returns a list (length of count) of randomly selected, non-repeating elements.

Vectors

`Vector3 DirectionVector3`
Returns random normalized Vector3 direction.

`Vector2 DirectionVector2`
Returns random normalized Vector2 direction.

`Vector3 InsideUnitSphere`
Returns a random point in a sphere with a radius of 1.

`Vector2 InsideUnitCircle`
Returns a random point in a circle with a radius of 1.

`Vector3 InsideUnitCube`
Returns a random point in a cube, with a max width and height of 2, and with the origin at the center.

Vector2 InsideUnitSquare
Returns a random point in a square, with a max width and height of 2, and with the origin at the center.

Vector3 InsideCube (float xSize, float ySize, float zSize, Vector3 direction)
Returns a random point in a cube, with a given x, y, and z size, as well as a direction. The origin is as the center of the cube.

Vector3 InsideCube (float xSize, float ySize, float zSize)
Returns a random point in a cube, with a given x, y, and z size. The origin is as the center of the cube.

Vector3 InsideCone (float height, float radius, Vector3 direction)
Returns a random point within a cone of the specified dimensions aligned along a direction. The origin is at the base of the cone.

Vector3 InsideCone (float height, float radius)
Returns a random point within a cone of the specified dimensions. The origin is at the base of the cone.

Vector3 InsideCylinder (float height, float radius, Vector3 direction)
Returns a random point within a cylinder of the specified dimensions aligned along a direction. The origin is at the center of the cylinder.

Vector3 InsideCylinder (float height, float radius)
Returns a random point within a cylinder of the specified dimensions. The origin is at the center of the cylinder.

Vector2 InsideCone2D (float length, float height)
Returns a random point within a cone of the specified dimensions. The origin is at the point of the cone.

Vector2 InsideCone2D (float length, float height, float rotationDegrees)
Returns a random point within a cone of the specified dimensions aligned along a rotation in degrees. The origin is at the point of the cone.

Vector3 InsideCapsule (float height, float radius, Vector3 direction)
Returns a random point within a capsule of the specified dimensions aligned along a direction. The origin is at the center of the capsule.

Vector3 InsideCapsule (float height, float radius)
Returns a random point within a capsule of the specified dimensions. The origin is at the center of the capsule.

Vector2 InsideCapsule2D (float height, float width, float rotationDegrees)
Returns a random point within a 2D capsule of the specified dimensions aligned along a rotation in degrees. The origin is at the center of the capsule.

Vector2 InsideCapsule2D (float height, float radius)
Returns a random point within a 2D capsule of the specified dimensions. The origin is at the center of the capsule.

Vector2 InsideConvextPolygon2D (Vector2[] vertices, float rotationDegrees)
Returns a random point within a 2D array of vertices - e.g. PolygonCollider2D. Has an extra parameter for rotation.

Vector2 InsideConvextPolygon2D (Vector2[] vertices)
Returns a random point within a 2D array of vertices - e.g. PolygonCollider2D.

Vector2 InsideBox2D (float width, float height, float rotationDegrees)
Returns a point inside of a 2D box, given a width, height, and rotation. The origin is at the center of the box.

Vector2 InsideBox2D (float width, float height)
Returns a point inside of a 2D box, given a width and height. The origin is at the center of the box.

Vector3 InsideBounds (Bounds bounds)
Returns a random point within a bounding box.

Vector3 InsideCameraFrustum (Camera cam)
Returns a random point within a given camera's view frustum.

Colors

Color ColorRGB ()
Returns a random color with an alpha of 1.

Color ColorRGBA ()
Returns a random color.

Color ColorFromGradient (Gradient gradient)
Returns a random color from a gradient.

Color ColorFromTexture (Texture2D texture)
Returns the color of a random pixel from a texture.

string ColorHex ()
Returns a random hex color, ignoring the alpha.

string ColorHexAlpha ()
Returns a random hex color, including the alpha.

Color ColorHSV ()
Returns a random HSV color.

Color ColorHSV (float hueMin, float hueMax)
Returns a random HSV color.

Color ColorHSV (float hueMin, float hueMax, float saturationMin, float saturationMax)
Returns a random HSV color.

Color ColorHSV (float hueMin, float hueMax, float saturationMin, float saturationMax, float valueMin, float valueMax)
Returns a random HSV color.

Color ColorHSV (float hueMin, float hueMax, float saturationMin, float saturationMax, float valueMin, float valueMax, float alphaMin, float alphaMax)
Returns a random HSV color.

Rotation

float RandomDegrees
Returns a random number between 0.0 and 360.0.

float RandomRadians
Returns a number between 0.0 and 6.28318530718 (pi * 2).

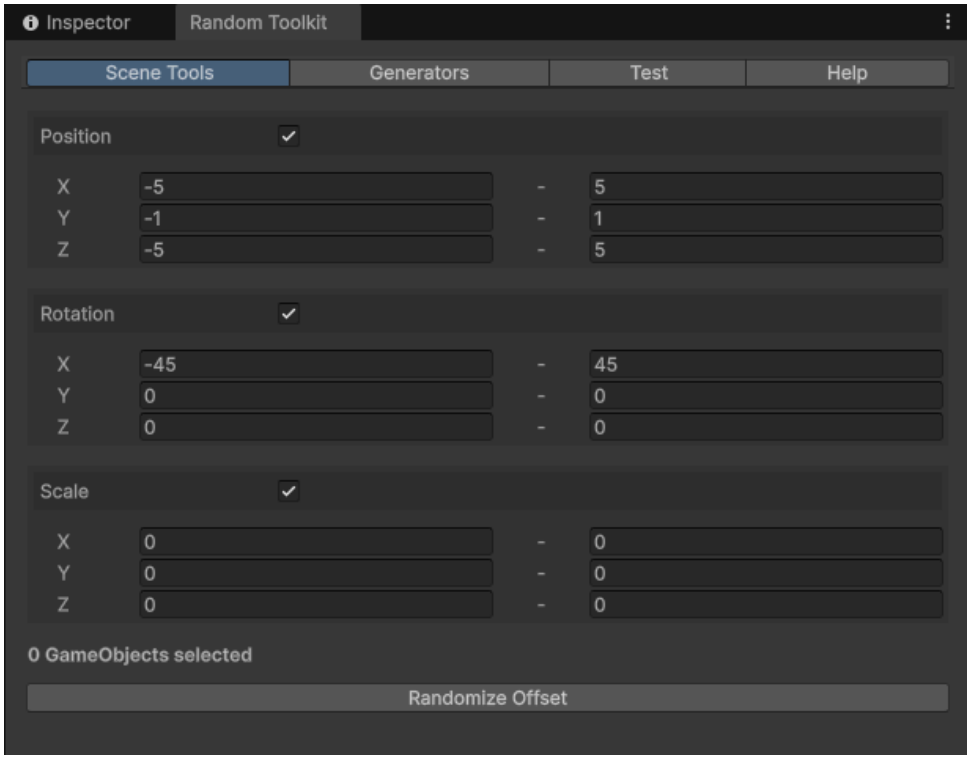
Quaternion RandomUniformQuaternion
Returns a random Quaternion with uniform distribution.

Editor Window

Random Toolkit features an editor window for you to use (**Window > Random Toolkit**).

The main aspect is **Scene Tools**, which allows you to apply a random Transform offset to all selected GameObjects in the scene.

- Enable position/rotation/scale, then set the min and max for each axis.
- Press **Randomize Offset** to apply the random offset.



There are three other pages in the editor window:

- **Generators** - Stats of the 4 generators featured in this toolkit.
- **Test** - Select an RNG and test out its capabilities.
- **Help** - Useful links.

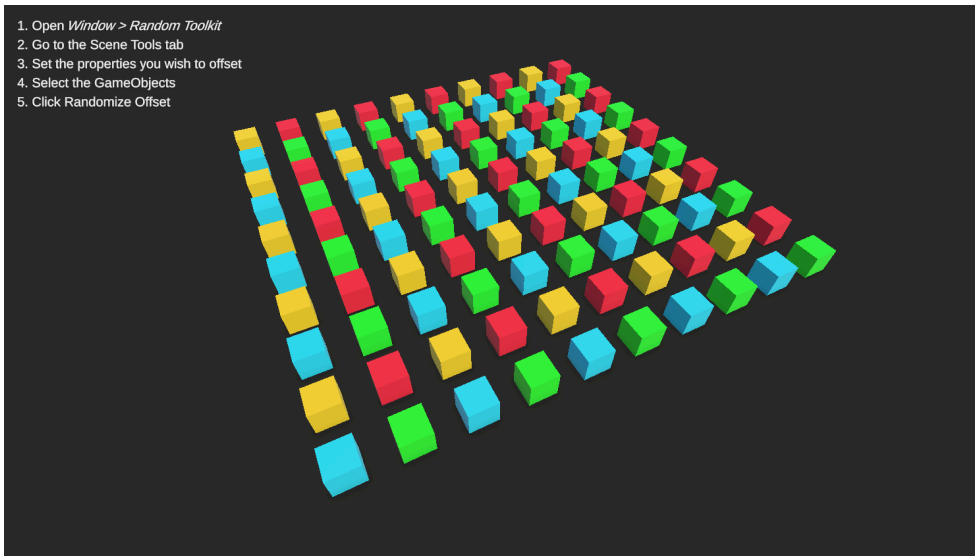
Demos

Since Random Toolkit focuses mainly on code, I've included some demo scenes so you can get a visual idea of what's going on.

Random Position Offset

Located: RandomToolkit/Demos/Random Position Offset/RT_RandomPositionOffset.unity

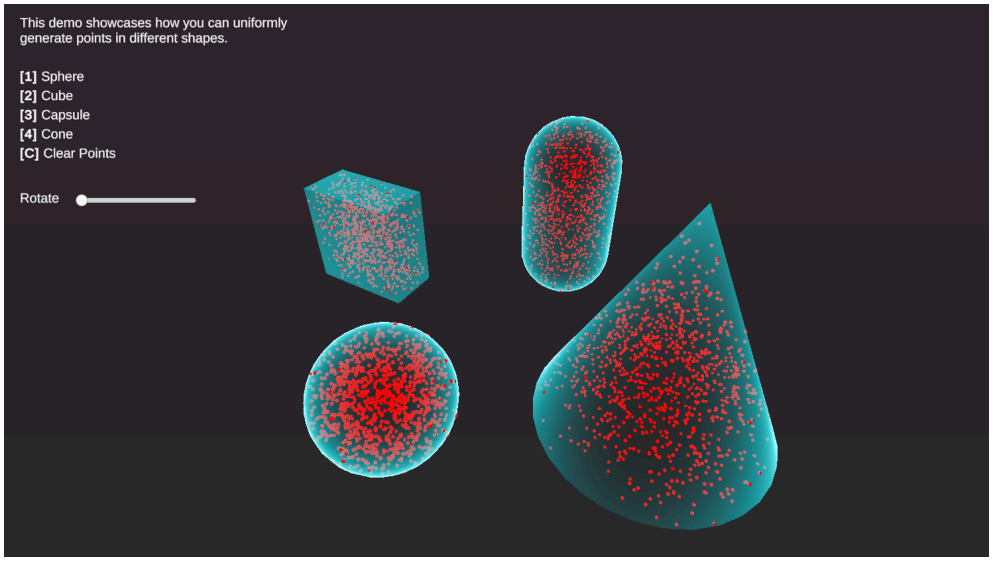
This demo showcases the Random Toolkit editor window capabilities. Simply follow the on-screen instructions to randomly offset the cubes.



Random Points Inside

Located: RandomToolkit/Demos/Random Points Inside/RT_RandomPointsInside.unity

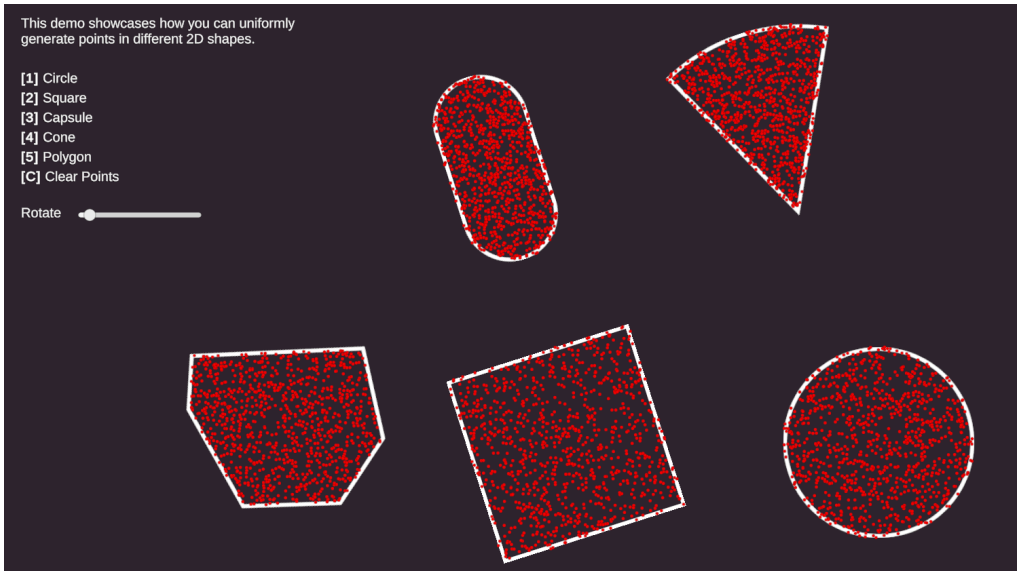
This demo showcases the ability to generate random points in a range of shapes. Press the number keys to spawn a large amount of points at a time, and use the rotation slider to see that these random points can also adjust based on that.



Random Points Inside 2D

Located: RandomToolkit/Demos/Random Points Inside 2D/RT_RandomPointsInside2D.unity

This demo showcases the ability to generate random points in a range of 2D shapes. Press the number keys to spawn a large amount of points at a time, and use the rotation slider to see that these random points can also adjust based on that.



Help

If you have any issues with the toolkit or wish further elaboration on anything, feel free to contact me here: buckleydaniel101@gmail.com

Here are some of my other assets you might like:

Time Traveler

Time Traveler allows you to pause, rewind, and playback time dynamically in your game.

Mini Golf - Complete Game

A complete 3D mini golf game with varying environments and over 100 parts to construct your courses.

3D Wave Shooter

Shoot and kill enemies with unique weapons and effects. Purchase new weapons and upgrades from the shop between rounds, and enjoy the systematic gameplay at hand.

Space Wave Shooter

Fight through waves of enemies, purchasing items and upgrades in-between rounds. This is a complete project with a game ready to be expanded upon and published.