Randomize Documentation

Chase Zemanek

Introduction

First of all, thank you for giving Randomize a shot! I am always trying to make Randomize better so if you have any issues or would like to make a suggestion, don't hesitate to reach out!

Overview

Randomize is an unobtrusive system for easily integrating variation into your Unity games. It does this by offering a special type of collection that stores groups of data and providing a way to get random elements from that collection.

Although the title is Randomize and randomization was the initial goal, Randomize also provides ways to get data in other methods. The way data is returned is controlled by one variable so there's no extra setup required and the return method can be changed between retrieving data.

The list of currently supported methods are:
Playlist: Uses a shuffled playlist so that each element is used once before repeating.
Random: Gets data completely randomly.
RandomUnique: Gets data randomly, but will not return the same element twice in a row.
Sequential: Gets data in the order it is stored.
SequentialReverse: Gets data in the reverse order it is stored.

RandomObject

RandomObjects are assets that allow multiple values of the same type to be grouped together in one collection to be accessed randomly. This collection can then be assigned to and used from scripts much in the same way as one would use a normal asset such as an AudioClip or Material.

New RandomObject instances can be created from the Create menu in the project window. Just select Create/Randomize and the type of RandomObject you would like to use. Each RandomObject is given a set type and will only accept data that is of that type.

After creation, we are presented with the RandomObject editor in the inspector window. From here we can set the return behavior, add values to the collection, change their order (only relevent for Sequential and SequentialReverse).

Once our RandomObject has been created and set up, we can incorperate it into our scripts. The following example will set the color of a Material using the RandomColor object.

Create a cube in the scene:

Create a new RandomColor:

Add some colors to the new RandomColor and set its return behavior to RandomUnique:

Now we must add our RandomColor to our code:

```
public class ColorChanger : MonoBehaviour {
     Zemanek.Randomize.RandomColor MyRandomColor;
}
```

We could also include the Randomize namespace at the beginning of the file:

```
using Zemanek.Randomize;
public class ColorChanger : MonoBehaviour {
     RandomColor MyRandomColor;
}
```

Now we want to change the color of the GameObject's material to one from our RandomColor whenever we press the space key:

```
using Zemanek.Randomize;
public class ColorChanger : MonoBehaviour {
     RandomColor MyRandomColor;

     void Update() {
          if(Input.GetKeyDown(KeyCode.Space)) {
               MeshRenderer renderer = GetComponent<MeshRenderer>();
               if(renderer != null) {
                    renderer.Material.Color = MyRandomColor.Next();
               }
          }
     }
}
```

Now just attach our new script to the cube in the scene and set MyRandomColor to be the RandomColor we created and we're good to go.

If you run the scene and press space you can see the color change each time from among those defined in the RandomColor. That's all it takes to incorperate randomness into any project with RandomObjects, just a couple line edits. Randomize was designed to require as little workflow and code changes as possible.

But when something is easy to use, it often comes with some limitations and RandomObjects are no exception. The biggest limitation is that the only types supported by RandomObjects are the types you can create from the menu. This has to do with the way Unity stores data in the background. Sometimes though, such a type limitation is unacceptable! For this we have the RandomData class.

RandomData

RandomData is a generic class that requires a bit more effort to set up, but has no limitations on type.

The easiest way to explain how RandomData works is through code. For this example we're going to add some randomness to a group of strings.

We start with the following code that simply prints the first string in an array when the space key is pressed:

```
public class StringPrint : MonoBehaviour {
	private string[] MyStrings = new string[] {
		"apple", "banana", "cherry"
	};
	if(Input.GetKeyDown(KeyCode.Space)) {
		Debug.Log(MyStrings[0]);
	}
}
```

To introduce randomness, we must first add a RandomData variable:

```
using Zemanek.Randomize;
public class StringPrint : MonoBehaviour {
	private string[] MyStrings = new string[] {
		"apple", "banana", "cherry"
	};
	private RandomData<string> MyRandomStrings = new RandomData<string>(
		MyStrings, RandomBehavior.RandomUnique
	);
	if(Input.GetKeyDown(KeyCode.Space)) {
		Debug.Log(MyStrings[0]);
	}
}
```

Woah, let's break down that constructor:

```
private RandomData<string> MyRandomStrings
```

Declares a new RandomData instance named MyRandomStrings that will use string data.

```
new RandomData<string>
```

Creates a new instance of RandomData that uses strings.

```
(MyStrings, RandomBehavior.RandomUnique)
```

Provides the RandomData instance with a collection of strings to work with and defines its return behavior. In this case, our RandomData instance will work with the string array from earlier and will follow the RandomUnique behavior.

Now that we've added our RandomData instance, we can use it in the same way we did with the RandomColor in the RandomObject example. Here all we're doing is replacing the MyStrings[0] in the Debug.Log function with MyRandomStrings.Next() to call the next random string.

```
using Zemanek.Randomize;
public class StringPrint : MonoBehaviour {
	private string[] MyStrings = new string[] {
		"apple", "banana", "cherry"
	};
	private RandomData<string> MyRandomStrings = new RandomData<string>(
		MyStrings, RandomBehavior.RandomUnique
	);
	if(Input.GetKeyDown(KeyCode.Space)) {
		Debug.Log(MyRandomStrings.Next());
	}
}
```

And just like that we've utilized the power of the RandomData class. In this example we just used a simple string type, but the RandomData can be applied to any type.

The main drawbacks of the RandomData class include the extra set up and the fact that since RandomData instances cannot be saved as assets, they are specific to the script they reside in.

Conclusion

Thanks again for giving Randomize a shot! I really hope it helps you in your game development journey.

Let me know if you have any issues with Randomize or if you think something is missing and I encourage you to check out my website to see if there's anything else I'm working on you could benefit from.

Thanks again!