Assets

> Read the code in the Examples Folder.

CustomAsset

Unity provides a base class called `ScriptableObject`. Derive from it to create objects that don't need to be attached to game objects. Scriptable objects are most useful for assets which are only meant to store data.

To make them easier to use (maybe), I have created `CustomAsset`. Since it is a `ScriptableObject`, it has all the same attributes described here.

For more detailed examples of all uses, view ***Askowl-Lib/Examples/Editor/TestCustomAsset.cs***.

A script asset is a file in the Unity project used to contain data and functionality. When defining a custom asset, use the CreateMenuAsset Attribute. Selecting the specified item from Assets/Create writes the asset. Move it to a directory named ***Resources*** anywhere in your project.

Fill the public data in the asset for use in methods to provide functionality. The classic example is a custom asset with an array of audio clips. A `Play` method can select a clip to play randomly. In this way, sound effects would be less monotonous.

```
[CreateAssetMenu(menuName = "Examples/Sound Clips", fileName = "Clips", order = 1)]
public class ClipsExample: CustomAsset<Clips> {
  public AudioClip[] clips;

  public void Play() {
    AudioClip clip = clips [Random.Range(0, clips.Length)];
    AudioSource.PlayClipAtPoint(clip, new Vector3 (0, 0, 0));
  }
}
```

From any ***Resources*** directory create an asset by selecting Examples / Sounds. Rename the asset to ***Birds***. Select the asset and add bird sounds to the list. A sample already exists in this package. Running the ***Askowl-Lib*** scene displays a ***Bird Sounds*** button that runs this script asset. The same technique could be used to select a prefab from a list to provide a variety of hazards.

Asset Selector

Many assets are plug-and-play. We looked at sound clips earlier, but what about projectiles, opponents or even the cloths a hero is going to wear. It is easy to make a game more interesting with mix-and-match. A prefab is an asset allowing for dynamic behaviour.

We can rewrite the `Clips` class above to use an `AssetSelector`.

```
[CreateAssetMenu(menuName = "Custom/Sound Clips", fileName = "Clips")]
public class Clips: AssetSelector<AudioClip> {
  public void Play() {
    AudioSource.PlayClipAtPoint(Pick(), new Vector3 (0, 0, 0));
  }
}
```

Create a clip from the menu. Remember to put it in a ***Resources*** folder. Then fill the assets list with the clips to select from.

The default picker chooses a random asset from the list. By overriding `Pick()`. An interesting variant would be a pick that chooses different items until all are exhausted. For a simpler example, cycle through the assets in order.

```
  int idx = 0;

  public override T Pick() {
    return assets [idx++ % assets.Length];
  }
```

Meet `Selector.Cycle()`, one of AssetSelector optional pickers. Pickers can be selected in `OnEnable` or by code that has a reference to the Custom Asset. `Selector.Random()` is the default. `Exhaustive()` is like random but it guarantees not to repeat an item until all the other options are exhausted. See `Selector` for more details.

If you need another way of choosing your item, subclass `AssetSelector` and override the `Pick()` method.

Selector

It is useful to select one item from a list as needed.

```
  Selector<int> selector = new Selector<int> (new int[] { 0, 1, 2, 3, 4
});
```

Create a clip from the menu. Remember to put it in a ***Resources*** folder. Then fill the assets list with the clips to select from.

The default picker chooses a random asset from the list. By overriding `Pick()`. An interesting variant would be a pick that chooses different items until all are exhausted. For a simpler example, cycle through the assets in order.

```
Selector<int> selector = new Selector<int> (new int[] { 0, 1, 2, 3, 4 });

for (int idx = 0; idx < 100; idx++) {
  int at = selector.Pick();
  Use(at);
}
```

The magic is in having different pickers for different requirements.

Choices

If the list of items to choose from changes, update the selector with `Choices`.

```
Selector<int> selector = new Selector<int> (new int[] { 0, 1, 2, 3, 4 });
selector.Choices = new int[] { 5, 6, 7, 8 };
```

Random Picker

`Random` is the default picker. In small lists is may appear to be favouring one or another asset.

Cycle Picker

As in the example above each asset is chosen in turn. To activate cycle picking, set it in `OnEnable`.

```
Selector<int> selector = new Selector<int> (new int[] { 0, 1, 2, 3, 4 });
selector.Cycle();

for (int idx = 0; idx < 100; idx++) {
  int at = selector.Pick();
  if (at != (idx % selector.Choices.Length))
    error("Cycle selector is broken");
}
```

`CycleIndex` returns the current index in the cycle. Use it if you want to react to a full cycle;

```csharp
int start = selector.CycleIndex;
do {
  Use(selector.Pick();
} while (selector.CycleIndex != start);
```

Exhaustive Picker

The pick is random, but it guarantees to choose an asset only once until all assets are exhausted.

```csharp
selector.Exhaustive();
```

There is are NUnit Editor tests in ***Examples/Scripts*** that demonstrate all the pickers.

Quotes

`Quotes` is a C# class that if given a list of lines or a `TextAsset` will return a line randomly using the `Pick` interface. A quote is formatted as a ***body of the quote (attribution)*** where the attribution is optional. RTF is acceptable in the quote.

```csharp
Quotes QuotesA = new Quotes();
Quotes QuotesB = new Quotes("name-of-a-TextAsset");
Quotes QuotesC = new Quotes(new string[]{
    "The trouble with having an open mind, of course, is that people will insist on coming along and trying to put things in it (Terry Pratchett)",
    "Never say, 'oops'. Always say, 'Ah, interesting'.",
    "Hints on how to play <b>the game</b> are rarely attributed."
    "Success does not consist in never making mistakes but in never making the same one a second time. (George Bernard Shaw)");

string aQuote = QuotesC.Pick();
```