Assets

> Read the code in the Examples Folder.

Components

`Components` is a static helper class with functions to create and find Components.

Components.Find(name)

Search the current scene for components of type `T`, then return the one with the name supplied. For a call with no name, we use the name of type T.

If there are no matching objects in the scene, `Find` will try to load a resource of the supplied type and name. The name can be any path inside a ***Resources*** directory.

Components.Create(gameObject, name)

Will create a component of type T inside the provided game object. The instance of T given the name supplied or the name of T if it is null.

Components.Create(name)

The overload that does not supply a gameObject will create a new one and name the same as the component. The new gameObject is attached to the root of the current hierarchy.

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<ClipsExample> {
  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.

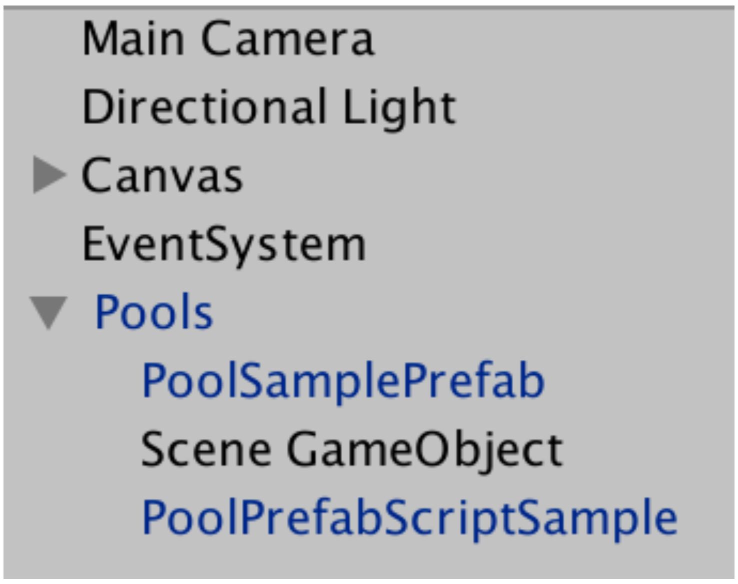
If you need another way of choosing your item, subclass `AssetSelector` and override the `Pick()` method.

Pooling

Unity3D games can run on lightweight platforms such as phones, tablets and consoles. Virtual or augmented reality games are immersive, and the least stutter in frame-rate is evident and annoying. Two of the most prominent culprits are instantiating many complex objects and garbage collection.

`Instantiate()` and `Destroy()` aren't evil, but using them on game objects that are regularly needed is an overhead that is better overcome. The solution is to use a pool of these objects. It minimises expensive instantiation, and the garbage collector does not have to reclaim the memory for every usage.

A GameObject becomes a pool of it has the `Pool` script attached. Any child object becomes candidates for pooling. Alternatively, you can drag the ***Askowl/Assets/Prefabs/Pools*** prefab into the hierarchy. You can have as many pools as you wish and they may reside in any scene. The names have to be unique.


Main Camera
Directional Light
Canvas
EventSystem
Pools
PoolSamplePrefab
Scene GameObject
PoolPrefabScriptSample

In this example, three GameObjects are pooling aware. ***Scene GameObject*** has been created within the scene, while the other two copies of the same prefab with differing values in editor-available fields. They represent two different characters or effects that differ only in detail.

To retrieve a clone from the pool, use `Acquire()`. A new GameObject is cloned from the master if the pool is empty.

To release an object back to the pool, just disable it.

```
myClone.gameObject.SetActive(false);
```

Never call `Destroy()` unless you don't want to reused the GameObject. It cannot be returned to the pool after `Destroy`.

Acquire GameObject by Name

Calling `Acquire` with the name of the GameObject will retrieve a clone from the pool, instantiating a new one if necessary. `Pool.Acquire("Scene GameObject")` does the trick, or returns null if the Pool did not contain an original by that name.

Seeding some of the GameObject information using optional parameters is possible.

Transform parent

An effect will have the target as the parent, while a character may have a spawn point or a team leader. Position and rotation below are relative to that of the parent.

Vector3 position

The location where the clone will spawn relative to the parent. It defaults to (0, 0, 0).

Quaternion rotation

The facing direction, relative to the parent.

bool enable

Defaults to true so that the clone is enabled when it is taken from the pool or created.

bool poolOnDisable

PoolOnDisable also defaults to true. Using `SetActive(false) to disable a component will cause it to return to the pool. For situations where you want to enable and disable and as part of game processing, set`poolOnDisable`to false. Use`Pool.Return(clone)` to release the GameObject to the pool for reuse.

```
    for (int i = 0; i < 21; i++) {
      prefab1[i] =
        Pool.Acquire<PoolPrefabScriptSample>("PoolSamplePrefab",
                                             parent: FindObjectOfType<Canvas>().transform,
                                             position: new Vector3(x: i * 60, y: i * 60));
    }
```

Acquire

The generic form of `Acquire` is a shortcut to get a component.

```
PoolPrefabScriptSample script = Pool.Asquire<PoolPrefabScriptSample>();
// is the same as
script = Pool.Asquire<PoolPrefabScriptSample>("PoolPrefabScriptSample");
// is the same as
GameObject clone = Acquire(typeof(T).Name);
script = (clone == null) ? null : clone.GetComponent<T>();
```

The same optional parameters are available as the non-generic game object - with the addition of name so that you can return a prefab with a different name to the MonoBehaviour inside.

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.

```
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();
```

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;

```
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.

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

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

Singleton

This Singleton class is a convenience copy of the code from the From http://wiki.unity3d.com/index.php?title=Singleton with some of the fluff removed. Use it as the super-class instead of `MonoBehaviour`.

```
public class SingletonSample : Singleton<SingletonSample> {
  public int value = 0;
}
```

Retrieve a reference using the static Instance field.

```
SingletonSample sample = SingletonSample.Instance;
```