Contents

Recycling

Pooling isn't all fun and games. To be able to recycle and re-use an object, whether it's a GameObject or a regular C# object, you need to remember to reset its values when you're done with it.

Resetting State

Imagine what would happen if you forgot to reset the values on an object, when returning it to the pool. The next time your code takes that object from the pool to use it, it has completely different values to the original object! This may appear as objects having the wrong rotation, being in the wrong state, or other unusual glitches.

For this reason it's very important that you correctly reset any changed values back to the original value. To help with this, all instances of an object in a pool will be given a special component when they're created: `PoolableGameObject` or `PoolableComponent`.

Notifications

The `PoolableGameObject` component is a MonoBehaviour script that's automatically added to all instances of a GameObject in a `GameObjectPool`. Its primary purpose is to provide notifications to your own components about when the object is being taken from the pool, and when it's being placed back into it.

The notifications are sent according to the value of the `NotificationMode` property. This is a property that is present on both the `PoolableGameObject` component, as well as the `GameObjectPool` itself. You should generally only set the value on the pool, and not on the component, as the component will be set up automatically.

Notification Mode	Description
Interface	A custom interface that is applied to any components that need to respond to the notification. Components attached to the pooled object should implement the IPoolable interface if they wish to perform actions when the object is acquired from, or released to, the pool.
SendMessage	The built-in Unity messaging system that sends notifications using the SendMessage(String) method. Components attached to the pooled object should implement the OnAcquire method to receive a notification when the object is acquired from the pool, and the OnRelease method to receive a notification when the object is released back to the pool.
UnityMessagingInterface	The built-in Unity messaging system that sends notifications using the ExecuteEvents class, using an interface applied to any components that need to respond to the notification.

See more about `PoolableGameObject` on the API Reference page.

Interface Mode

Using the Interface notification mode is usually the best choice. To use this mode, you should ensure all components on the source object (the object or prefab being pooled) implement the IPoolable interface. The interface defines two methods, `Acquire` and `Release`, which are invoked by the pool when the instance is acquired from, or released back to, the pool.

SendMessage Mode

The SendMessage notification mode uses the built-in Unity messaging system, and should be familiar to most users. There's no need for an interface, but there is a small performance hit using this mode. To use SendMessage, you should ensure all components on the source object implement the two methods: `Acquire` and `Release`, which are invoked by the pool when the instance is acquired from, or released back to, the pool.

UnityMessagingInterface Mode

This mode is very similar to the Interface mode. You'll need to implement the IPoolable interface and implement the two methods: `Acquire` and `Release`. Rather than the methods being invoked by Pure Pool directly, they'll be invoked using Unity's ExecuteEvents class. Unless there's a particular reason for you to need this, we recommend using the Interface mode instead.

Implementing Acquire and Release

Our recommendation is to use the Acquire method as a replacement for the MonoBehaviour's Awake and Start methods, and use the Release method to reset values. It's important to remember that a pooled object is created once and then recycled, so its Awake and Start methods will only run one time. Any code you would usually put in the Awake or Start methods may make more sense in the Acquire method.

If you need to pool an object that has components you can't modify, you should create a new MonoBehaviour and implement one of the notification modes above. You can then use the new component to reset the one you can't change. To make your life easier, Pure Pool includes components to reset AudioSources, ParticleSystems, Rigidbodies, and user-created MonoBehaviours:

Component	Description
PoolableAudioSource	A component that allows pooling of multiple associated AudioSource components. The audio sources will be stopped from playing when the object is released back to the pool, and their position will be set back to 0 when the object is acquired from the pool.
PoolableParticleSystem	A component that allows pooling of multiple associated ParticleSystem components.
PoolableRigidbody	A component that allows pooling of multiple associated Rigidbody components. The `velocity` and `angularVelocity` are set to zero when the object is released back to the pool.
PoolableMonoBehaviour	A component that allows pooling of multiple associated MonoBehaviour components. Converts all MonoBehaviours to JSON representations and stores them internally, using `JsonUtility.ToJson`. When the object is released back to the pool, the JSON representations of each MonoBehaviour are restored, using `JsonUtility.FromJsonOverwrite`. This component can make it easy to reset your MonoBehaviours, but it can only reset fields that are serialised. If this component does not meet your needs, you should implement one of the notification modes above to properly reset your MonoBehaviour.