Contents

Pooling C# Objects

In addition to pooling GameObjects in Unity, Pure Pool can also be used to pool regular C# objects, including your own custom classes.

GenericObjectPool<T>

A generic implementation of an object pool, that allows for recycling and reuse of objects of type *T*. The GenericObjectPool class cannot be serialised by Unity and will become a null reference if using Unity's live recompilation feature. For the ability to serialise the object pool, the SerialisableObjectPool class is highly recommended.

SerialisableObjectPool<T>

A serialisable, generic implementation of an object pool, that allows for recycling and reuse of objects of type *T*.

By virtue of being serialisable, SerialisableObjectPool can survive an assembly reload caused by live recompilation inside of the Unity editor. However, to ensure Unity is able to serialise fields containing pools, you should subclass SerialisableObjectPool<T> by creating a new, non-generic, class derived from it.

SerialisableObjectPool achieves this by serialising the number of instances of the object that were contained in the pool, and then recreating them after deserialisation. In other cases, it's possible to let Unity serialise the objects contained in the pool, and simply add them back into the pool after deserialisation.

To use the SerialisableObjectPool<T> class, derive a new, non-generic, class from it and override the `GetObjectFactory` method. This method is responsible for providing an object factory that can create new instances of the desired object. Initialise a new instance of the derived class using the constructor, and then set the properties to appropriate values. Once all properties have been set, invoke the `Initialise` method. A pool cannot be used without being initialised in this way.

SerialisableObjectPool Example

Derive a new class from SerialisableObjectPool, and specify the type of object you want to pool as the generic type parameter:

```
public class MyCustomClassPool : SerialisableObjectPool<MyCustomClass>
{

        /// <summary>
        /// Gets a function used to create new instances of the pooled
type.
        /// By default, this method uses the public parameterless
constructor of type <typeparamref name="T"/>.
        /// This method should be overridden in a subclass if
different behaviour is required.
        /// </summary>
        /// <returns>A function that can be used to create new
instances of the pooled type.</returns>
        /// <exception cref="InvalidOperationException">No public
parameterless constructor could be found on type <typeparamref
name="T"/>.</exception>
        protected virtual Func<T> GetObjectFactory() {
                // Use the public parameterless constructor for type T
as the object factory method.
                ConstructorInfo defaultConstructor =
typeof(T).GetConstructor(Type.EmptyTypes);
                if (defaultConstructor == null) throw new
InvalidOperationException($"No public parameterless constructor could
be found on the pooled type ({typeof(T).FullName}). Override the
{nameof(this.GetObjectFactory)} method in a subclass to return a
suitable object factory.");
                return
Expression.Lambda<Func<T>>(Expression.New(defaultConstructor)).Compile
();
        }

}
```

You can then declare a field of this new type in your MonoBehaviour, marking it with the `SerializeField` attribute:

```
public class MyScript : MonoBehaviour {

        [SerializeField]
        private MyCustomClassPool myCustomClassPool;

}
```