User Guide

Ultimate Pooling

Pooling made easy

Trivial Interactive

Version 1.0.0

Ultimate Pooling is a fast and easy to use pooling system that can dramatically improve the performance of your game be reducing the amount of calls to 'Instantiate' and 'Destroy. Ultimate Pooling is designed to be as simple to use as possible while maintaining maximum performance, and as a result pooling can be fully integrated into your game using 2 static methods. It couldn't be simpler!

Features

- Very easy to integrate into your game
- Massive performance increases over traditional methods
- Support for pooling Game Objects and Components
- Support for pooling assets in 'Resources' folder
- Create pools at runtime or in the editor
- Receive events on pooled objects when they are spawned or despawned
- Fast 'batchSpawn' and 'batchDespawn' methods when a number of objects need to be spawned
- Fully commented C# source code

Getting Started

This section will quickly take you through the process of adding basic pooling support using Ultimate Pooling. It is very simple to add pooling support using our simple interface and requires no previous knowledge about pooling to implement

There are 3 quick and easy steps you must complete in in order to add pooling support to your game:

1. Import the namespace: In order to avoid naming clashes, all scripts associated with Ultimate Pooling are organised under the 'UltimatePooling' namespace. This means that before you can access the Ultimate Pooling API in your scripts you will need to import the namespace as shown below:

```
7
8   // Make sure we can access the Ultimate Pooling API
9   using UltimatePooling;
10
```

2. Remove calls to 'Instantiate': In order to add pooling into your game you will need to replace any calls to 'Instantiate' with calls to 'spawn' which can be access via the static 'UltimatePool' class. The static spawn method is a direct replacement for 'Instantiate' with a few more overloads used specifically for pooling. The spawn method will attempt to re-use a pooled object where possible as opposed to creating a new object. The usage is shown below:

```
13  public class Spawner : MonoBehaviour
14  {
15      public GameObject prefab;
16
17      public void createEnemy()
18      {
19          UltimatePool.spawn(prefab);
20      }
21  }
```

Figure 1

The spawn method has a number of overloads which are shown below to provide identical behaviour as the 'Instantiate' method. Note that components can also be spawned.

```
16      public static GameObject spawn(GameObject prefab)
17
18      public static GameObject spawn(GameObject prefab, Vector3 pos, Quaternion rot)
19
20      public static Object spawn(Component prefab)
21
22      public static Object spawn(Component prefab, Vector3 pos, Quaternion rot)
23
24      public static T spawn<T>(Component prefab) where T : Object
25
26      public static T spawn<T>(Component prefab, Vector3 pos, Quaternion rot) where T : Object
27
28      public static GameObject spawn(string prefabName)
29
30      public static GameObject spawn(string prefabName, Vector3 pos, Quaternion rot)
```

Figure 2

3. Remove calls to ‘Destroy’: In order for the ‘spawn’ method to work correctly, it needs to be paired with a matching ‘Despawn’ call which informs Ultimate Pooling that an object is no longer required in the scene and can be pooled for re-use at a later time. As with ‘spawn’, the ‘despawn’ method is a direct replacement for ‘Destroy’ including the overloads that destroy an object after a specified amount of time. The usage is shown below:

```
public class Spawner : MonoBehaviour
{
    public void destroyEnemy(GameObject enemy)
    {
        UltimatePool.despawn(enemy);
    }
}
```

Figure 3

That’s it! You have now successfully integrated pooling into your game. You can verify that pooling is working by looking out for ‘PoolGroup’ objects being created when the game is run. You will notice that there is a group of child objects which are ready to be spawned when needed (provided that default settings are used).

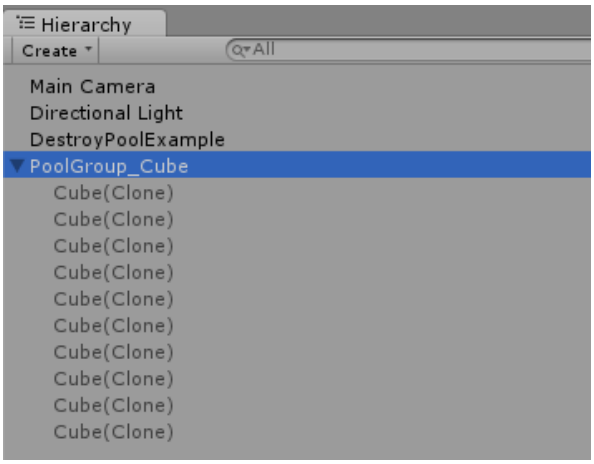


Figure 4

What now?

Once you have completed this section then you have fully integrated pooling and don’t need to worry about the performance of ‘Instantiate’ or ‘Destroy’ any more. You can simply stop reading and enjoy pooling. On the other hand, w have only scratched the surface of Ultimate Pooling and there is much more you can do, such as defining custom pools, handling spawned and despawned events and much more. Continue reading to explore some of the more advanced features!

Creating Pools

Ultimate Pooling provides 2 methods which allow you can create a pool group for a prefab object. The first method is the easiest and will be used by the majority of people and involves creating an object pool via the Unity Editor. The second method is for the more advanced use and involves creating pools dynamically at runtime via scripting. Both methods are explained in detail below:

Create Pool in Editor

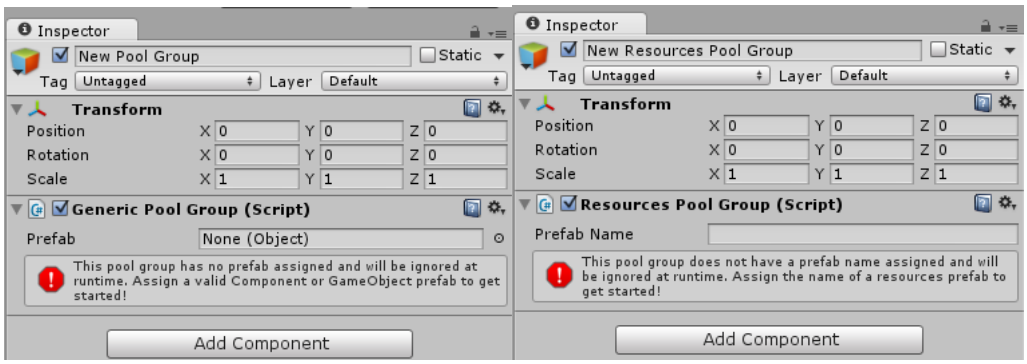
There are a number of way I which you can create a PoolGroup object but the easiest by far is to use the provided menu function which construct a game object with the PoolGroup component attached. You can find these menu items at:

Menu bar - 'GameObject/UltimatePooling/'

Or

Right click hierarchy window - 'UltimatePooling/'

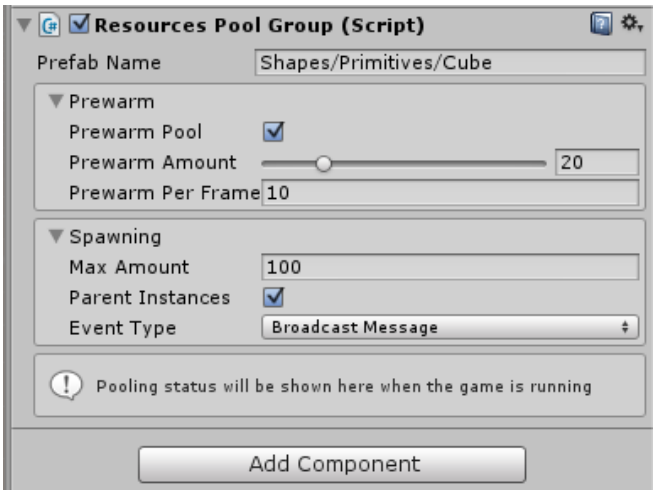
There are 2 items located under this menu which are 'Create Pool' and 'Create Resources Pool'. The first item will create a generic pool object that can accept prefab game objects and components as the spawning type. The second item is used when you want to create a pool for an object that exists within the resources folder, and accepts a string value that represents the path used to load the object from the resources folder.



Once you have created a PoolGroup object you should see one of the above components in the Inspector window. Both of these components require that you assign a prefab value before you can edit its pooling settings. For the GenericPoolGroup component you can simply drag a prefab object into the prefab slot as you would do with any other component. The ResourcesPoolGroup component requires that you type the name of the prefab into the prefab name field.

Common Settings

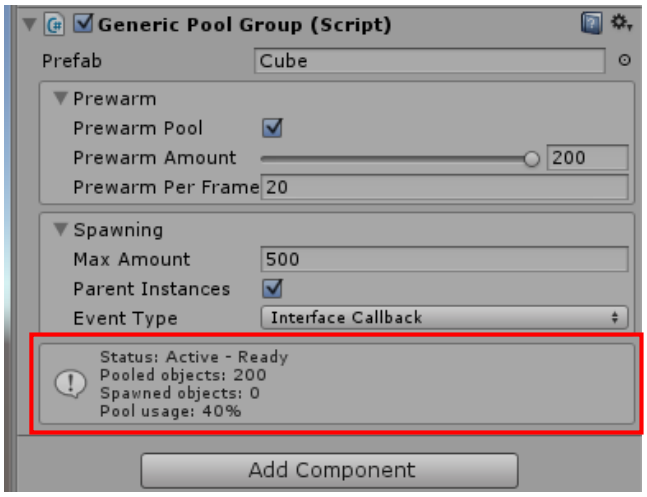
Once a valid prefab has been assigned to the component you will be able to access the settings for the pool which are the same regardless of the pool type. These settings are explained below:



- **Prewarm Pool:** When enabled, the pool will attempt to create a pre-set number of objects when the game starts. These objects will be immediately pooled and means that the pool will be populated when 'spawn' is called.
- **Prewarm Amount:** Visibility depends on 'Prewarm Pool'. Specifies the amount of objects that should be generated during the prewarm. This value will depend entirely on how often the prefab is spawned and the average lifetime of the object. In general the more a prefab is used, the greater this value should be to avoid calls to 'Instantiate' when the pool becomes empty. Note that this value is clamped between 0 and 'max Amount'.
- **Prewarm Per Frame:** Visibility depends on 'Prewarm Pool'. Specifies the amount of objects that are allowed to spawn in a single frame. This allows the initial 'Instantiate' calls to be spread across frames to reduce lag spikes and loading time. If you want to prewarm 50 objects and you allow 5 object per frame to be created then the pool will be warmed after 10 frames.
- **Max Amount:** The maximum number of objects that this pool can hold. The lower this value, the lower the memory footprint of the pool but you may risk more calls to 'Instantiate' if the pool does not contain any pooled objects. If the number of pooled objects is greater than this value then one object will be permanently destroyed per frame until this is not the case. In general, this number should roughly correlate to the maximum potential objects of the prefab type in the scene at any time. For example: If you know that you will never have more than 50 bullets in the scene at any time then it would make sense to set this value to 50 also.
- **Parent Instances:** When enabled, all objects spawned by this pool will be child objects of the pool game object. This allows the hierarchy to remain un-cluttered and keeps objects organised based on their prefab type. Note that when a pool is destroyed you have the option to keep these child objects alive or destroy them along with the pool.
- **Event Type:** The preferred method for receiving spawn and despawn events on pooled objects. For more information take a look at the Spawn Events Section.

Debugging Aid

When the game is running and a pool object is selected, you will be able to see detailed information about the pool status such as the number of objects currently pooled and the usage amount of the pool. This can be very useful when debugging and tweaking the pools values to ensure that maximum performance is achieved.



Create Pool via Scripting

There are some occasions where you might want to create an object pool dynamically and modify its initial settings through scripts. This can be achieved using the 'PoolManager' class which can be accessed through the static 'UltimatePool' class. For more information about this class you can take a look at the scripting reference included with Ultimate Pooling.

Creating a pool at runtime can be achieved using the following code:

```
25      private void Start()
26      {
27          // Create a new pool for the prefab object.
28          // Note that if there is already a pool for this prefab then this method will simply returne a
29          PoolGroup pool = UltimatePool.Pools.createPool(prefab);
30
31          // Initialize the prewarm values
32          pool.prewarmPool = true;
33          pool.prewarmAmount = 200;
34          pool.prewarmPerFrame = 20;
35
36          // Initialize the spawning values
37          pool.maxAmount = 500;
38          pool.parentInstances = true;
39          pool.eventType = PoolEventType.InterfaceCallback;
40      }
```

For more information regarding scripting with Ultimate Pooling take a look at the included scripting reference which documents the complete API.

Spawn Events

Once you have pooling setup in your game you may find that you need to know when an object is spawned or despawned since you are no longer receiving 'Start' or 'OnDestroy' calls. This is a common requirement and can allow you to reset all variables related to an object so that it can be considered a newly spawned object.

There are a couple of ways that you can listen for these events and each method depends on how you want to receive events. When you create an object pool you will have the option to change the 'eventType' value. There are 2 possible values which are Broadcast and Interface. The Broadcast method uses the built in 'BroadcastMessage' method to 'spawn' and 'despawn' methods. The second method requires that you implement an interface whose methods will be called directly. Note that the second method offers much greater performance.

1. **Broadcast Message:** For this method you will simply need to add two method to your mono behaviour script just as you would do for 'Start' or 'Update' and when the parent game object is spawned or despawned these methods will be triggered. Note that you can have as many scripts as you require with these methods attached to the same object, and these scripts can be at any level in the objects hierarchy structure. The following script shows how to add these methods:

```
public class PooledObject : MonoBehaviour
{
    public void OnSpawned(PoolGroup pool)
    {
        // Initialize script variables
    }

    public void OnDespawned(PoolGroup pool)
    {
        // Cleanup script variables
    }
}
```

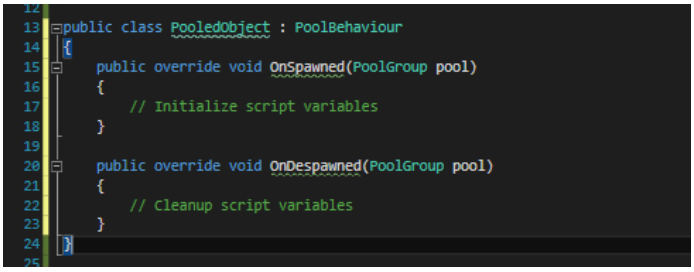
2. **Implement 'IPoolReceiver':** This method requires that you implement a C# interface which contains the spawn and despawn events. This method should be used where extra performance is required as calls to 'BroadcastMessage' can be painfully slow. To implement the interface simply inherit from 'IPoolReciever' in your mono behaviour script as shown below:

```
public class PooledObject : MonoBehaviour, IPoolReceiver
{
    public void OnSpawned(PoolGroup pool)
    {
        // Initialize script variables
    }

    public void OnDespawned(PoolGroup pool)
    {
        // Cleanup script variables
    }
}
```

You will notice that the method names used are the same regardless of the event type used.

3. **Inherit 'PoolBehaviour':** This method supports using Broadcast or Interface event types since the 'PoolBehaviour' script implements the 'IPoolReceiver' interface. Simply inherit from the 'PoolBehaviour' script instead of mono behaviour and you are then able to override the spawn or despawn events. Note that this script inherits from Mono Behaviour so you will still have access to all of Unity's methods.



```
public class PooledObject : PoolBehaviour
{
    public override void OnSpawned(PoolGroup pool)
    {
        // Initialize script variables
    }

    public override void OnDespawned(PoolGroup pool)
    {
        // Cleanup script variables
    }
}
```

It is worth noting that you do not necessarily need to include both methods in your script if you don't need to. If you only require the 'OnSpawned' event to be called then you can simply leave out the 'OnDespawned' if you are using the Broadcast method.

Report a Bug

At Trivial Interactive we test our assets thoroughly to ensure that they are fit for purpose and ready for use in games but it is often inevitable that a bug may sneak into a release version and only expose its self under a strict set of conditions.

If you feel you have exposed a bug within the asset and want to get it fixed then please let us know and we will do our best to resolve it. We would ask that you describe the scenario in which the bug occurs along with instructions on how to reproduce the bug so that we have the best possible chance of resolving the issue and releasing a patch update.

http://trivialinteractive.co.uk/bug-report/

Request a feature

If you feel that Ultimate Spawner should contain a feature that is not currently incorporated then you can request to have it added into the next release. If there is enough demand for a specific feature then we will do our best to add it into a future version. Please note, there are some features that are not possible to implement due some limitations of Unity but a workaround may be possible.

http://trivialinteractive.co.uk/feature-request/

Contact Us

Feel free to contact us if you are having trouble with the asset and need assistance. Contact can either be made by the contact options on the asset store or buy the link below.

Please attempt to describe the problem as best you can so we can fully understand the issue you are facing and help you come to a resolution. Help us to help you :-)

http://trivialinteractive.co.uk/contact-us/