Read Me

Thank you for downloading Lean Pool!

If you have any questions, feel free to email me at: carlos.wilkes@gmail.com

Step 1 - Replace your Instantiate calls

Find your code that heavily relies on:

```
Instantiate( … );
```

And replace it with:

```
Lean.Pool.LeanPool.Spawn( ... );
```

For example: `var enemy = Lean.Pool.LeanPool.Spawn(enemyPrefab);`

Step 2 - Replace your Destroy calls

Find your code that heavily relies on:

```
Destroy( … );
```

And replace it with:

```
Lean.Pool.LeanPool.Despawn( ... );
```

For example: `Lean.Pool.LeanPool.Despawn(enemy);`

Step 3 - Done!

Wasn't that easy?

Why do I have to keep typing 'Lean.Pool.' before everything?

To improve organization, all Lean Pool classes are inside the Lean.Pool namespace.

If you don't like typing Lean.Pool. each time, then you can add the following code to the top of your script: using Lean.Pool;

You can now just call LeanPool.Spawn(...) etc

How do I alter the settings and stuff?

If you use the steps above then Lean Pool will automatically create itself when you first call Lean.Pool.LeanPool.Spawn, so to adjust the settings ahead of time you need to manually create the pool. To do this, right click in your **Hierarchy** window, and select **Lean / Pool**. You should now see a Lean Pool GameObject in your scene that's automatically selected.

Next, drag and drop your prefab into the **Prefab** slot, and this will associate this pool with this prefab. You can now adjust the pool settings like the Preload, Capacity, etc.

Help, my Rigidbody velocity isn't being reset!!

By default, Lean Pool doesn't modify any components when spawning or despawning. This means when a Rigidbody is despawned it will retain its old velocity, and when spawned again it will shoot off with its old velocity. To fix this, the velocity must reset when it spawned, which can be done by adding the LeanPooledRigidbody component that comes with Lean Pool.

How do I create custom spawning/despawning behaviour for the _____ component?

All you need to do is make a new component and define the OnSpawn and/or OnDespawn methods. Lean Pool will automatically send a message to these methods when spawning and despawning your prefabs, allowing you to handle correctly resetting their state.

NOTE: Despawned prefabs are also deactivated, so you could handle such code in OnEnable and OnDisable.

I don’t want to use messages, they’re slow!

By default. Lean Pool sends the OnSpawn and OnDespawn messages to prefab clones to handle custom pooling, but it also supports Unity Events for greater performance. To use this, simply change your pool’s **Notification** setting to **Poolable Event**, add the **LeanPoolable** component to your prefab, and link the OnSpawn and OnDespawn events to the methods you need to call to properly spawn and despawn your prefabs.

This system is much faster because the reference to the LeanPoolable component is only retrieved once per prefab clone, and so there is no searching through the prefab hierarchy as in SendMessage or BroadcastMessage.

How do I increase the performance even more?

You may notice that Lean.Pool.LeanPool.Spawn and Lean.Pool.LeanPool.Despawn have a little bit of extra overhead. This is because they’re static methods that search a dictionary to find the pool associated with the prefab clone before performing the actual spawn or despawn code.

If this is an issue for you then you need to store a reference to the Lean.Pool.LeanGameObjectPool itself, and then you can call: myPool.Spawn and myPool.Despawn directly.

Want More Assets?

If you like this asset then please check out my website here: https://sites.google.com/site/carloswilkes/