Easy Pool Kit - Easiest Pool Manager for objects and Game Objects

What's it?

Easy Pool Kit is a **powerful** but **easy to use** tool for pooling objects in Unity.
It provides simple and lightweight managers to pool objects both for C# objects, Unity game objects and recyclable objects(recommended).
You can start pooling your objects in **just one line of code**!
You can also customize the pool configuration easily for **better performance** and **precise memory management,** such as particle pool or sound pool.

From this package, you'll get:

- A really **simple to use** and **high performance** tool for pooling objects.
- Pool management for C# objects, Unity Game Objects and **Recyclable Objects**.
- **Full C# source code** witch is **easy to use** and **easy to extended**.
- Powerful **debug toolkit** to help you **visualizing** the pool usage information.
- Compatible to **all Platforms** Unity support, of cause can be used on mobile.
- Easy to understand **demos** provided.

How to use it?

Import the package and that's it!

Let's have a look at the 4 demos:

1. Demo1_SimpleGameObjectPool

This demo shows how to simply spawn a new pooled game object just in one line of code:

```
GameObject newObj = SimpleGOPoolKit.Instance.SimpleSpawn(objectTemplate);
```

"objectTemplate" is the template Game Object or Prefab you want to copy from.

and despawn it with code:

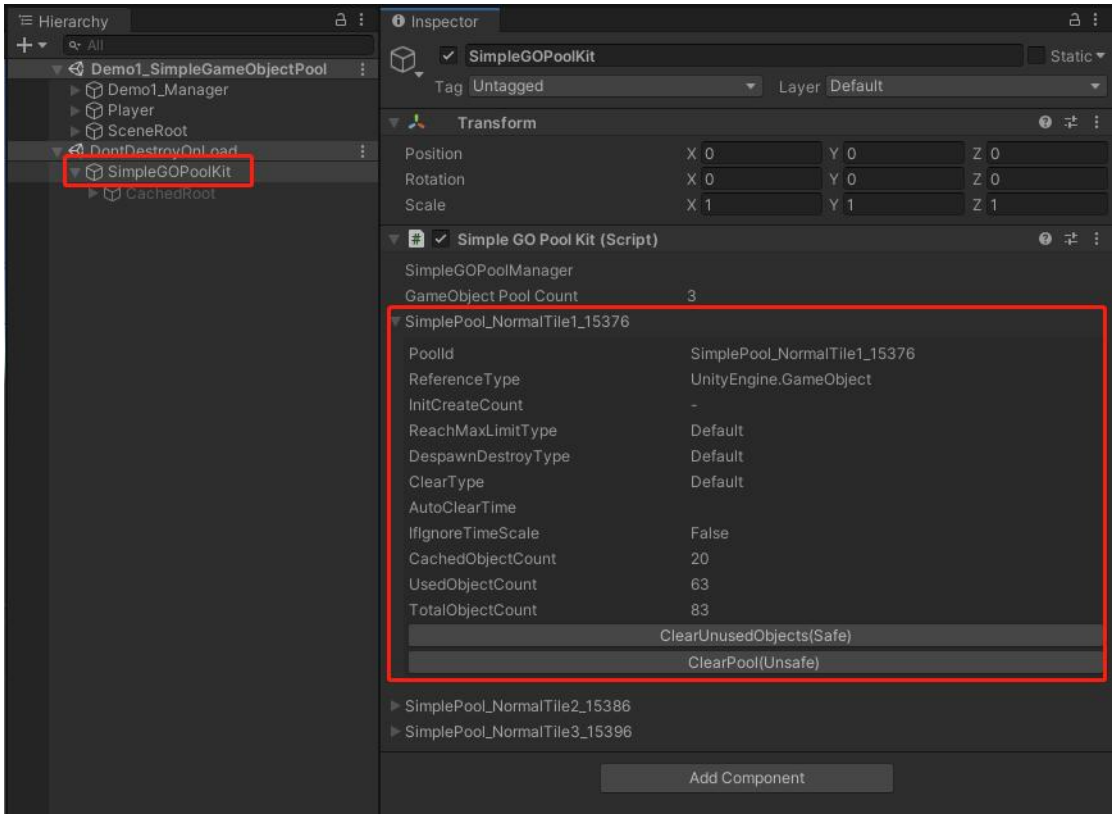
```
SimpleGOPoolKit.Instance.Despawn(pooledObj);
```

When running the demo1 scene, you can select "SimpleGOPoolKit" in Hierarchy and watch in the Inspector. It indicates how many simple Game Object Pools exists with their own configure and status.
You can click the button "**Clear Unused Objects**" to destroy all objects not used in pool now, it is a **safe** operation as the used pooled objects are still alive. You can also do it in one line of code:

```
SimpleGOPoolKit.Instance.ClearAllUnusedObjects();
```

You can click the button **"Clear Pool"** to destroy all objects used or cached from the pool. It's an unsafe operation as the used pooled objects will be destroyed as will. It can be coded as:

```
SimpleGOPoolKit.Instance.ClearAllPools();
```



SimpleGOPoolKit is always used to spawn some simple game object without any status, it can be pooled and recycled simply without extra work to reset its status or any other inner code.

2. Demo2_ObjectReferencePool

This demo shows the usage of ObjectPoolKit which can spawn any **pure C# objects** such as your customized EventArgs. The API is nearly the same as SimpleGOPoolKit:

Spawn C# object:

```
var textColorChangeArg = ObjectPoolKit.Spawn<ChangeColorEventArgs>();
```

Despawn C# object:

```
ObjectPoolKit.Despawn(textColorChangeArg);
```

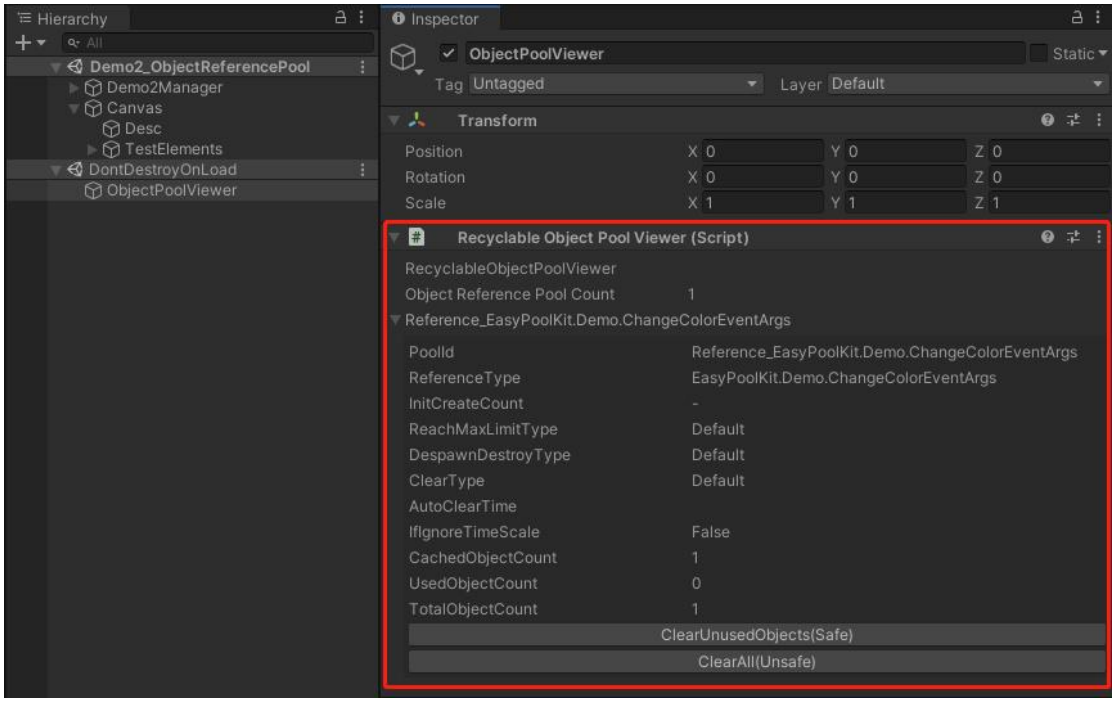
Clear unused pooled objects:

```
ObjectPoolKit.ClearAllUnusedObjects();
```

Clear all pooled and used objects:

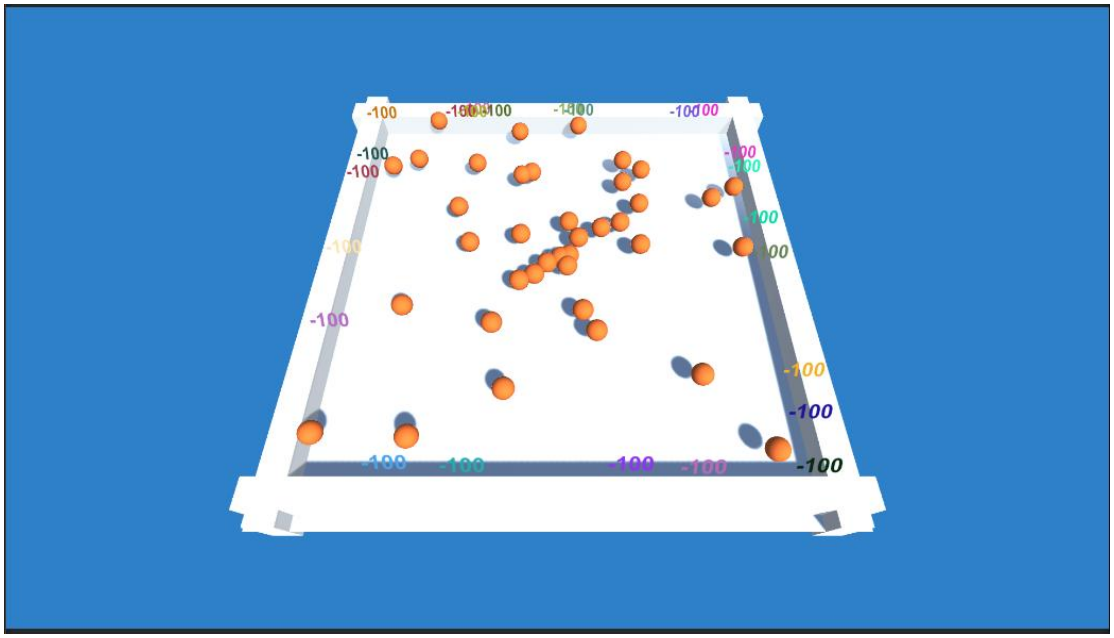
```
ObjectPoolKit.ClearAllPools();
```

The ObjectPoolKit can be used in any C# project not only in Unity. If you want to watch the debug info in Inspector Window, just drag the [EasyPoolKit/Prefabs/ObjectPoolViewer.prefab] into the scene.



3. Demo3_RecyclableGameObjectPool

This demo shows how to create customized objects and pools to get a better performance.



Every ball in this demo has a component **[DemoRecyclableBall]** which is inherited the base class **[RecyclableMonobehaviour]**. It overrides the method **"OnObjectDespawn()"** to reset the status variables when given back to pool manager.

The hurt text pool is customized by the configuration of hurtTextConfig:

```
var hurtTextConfig = new RecyclablePoolConfig
{
    ObjectType = RecycleObjectType.RecyclableGameObject,
    ReferenceType = typeof(DemoHitText),
    PoolId = "HurtTextPool",
    InitCreateCount = 20,
    ReachMaxLimitType = PoolReachMaxLimitType.RecycleOldest,
    MaxSpawnCount = 30,
    DespawnDestroyType = PoolDespawnDestroyType.DestroyToLimit,
    MaxDespawnCount = 25,
    ClearType = PoolClearType.ClearToLimit,
    AutoClearTime = 0.5f,
    IfIgnoreTimeScale = false,
};
RecyclableGOPoolKit.Instance.RegisterPrefab(HurtTemplate, hurtTextConfig);
```

- ObjectType has 3 types:

(1) **Object** is pure C# object,
(2) **RecyclableGameObject** is game object with component inherited RecyclableMonoBehaviour,
(3) **GameObject** is simple game object without any variable to reset when pooled or recycled.

- InitCreateCount:

If you set InitCreateCount, the pool manager will create the same size of objects when first create the pool.

- ReachMaxLimitType has 3 types:

(1) **Default** means there's no limit of the pool size, you can get as many as objects from pool.
(2) **Reject Null** should be used with **MaxSpawnCount**. If used object's count is larger than the MaxSpawnCount, the pool manager will **return null** when spawn method is called.
(3) **Recycle Oldest** should also be used with **MaxSpawnCount**. When used object's count is larger than MaxSpawnCount, the pool manager will **force recycle the oldest object** and return it as the new spawned object.

- DespawnDestoryType has 2 types:

(1) **Default** means when a used object is given back to the pool manager, it will not be destroyed.
(2) **DestoryToLimit** should be used with **MaxDespawnCount**, when a used object is given back to the pool and the pooled object's count is equal or larger than MaxDespawnCount, the object will be destroyed.

- ClearType has 2 types:

(1) **Default** means when pool manager calls "ClearAll()", it will destroy all objects in the pool.
(2) **ClearToLimit** means when pool manager calls "ClearAll()", it will destroy objects to the count of **InitCreateCount**.

- AutoClearTime

If you want to make the objects spawned from pool auto despawn in time, set the AutoClearTime

in seconds else set it as null.

As for the customized pool and object, you can spawn object:

```
DemoRecyclableBall newPlayer =
RecyclableGOPoolKit.Instance.SimpleSpawn<DemoRecyclableBall>(PlayerTemplate);
```

Despawn object:

```
newPlayer.DespawnSelf();
```

or

```
RecyclableGOPoolKit.Instance.Despawn(newPlayer);
```

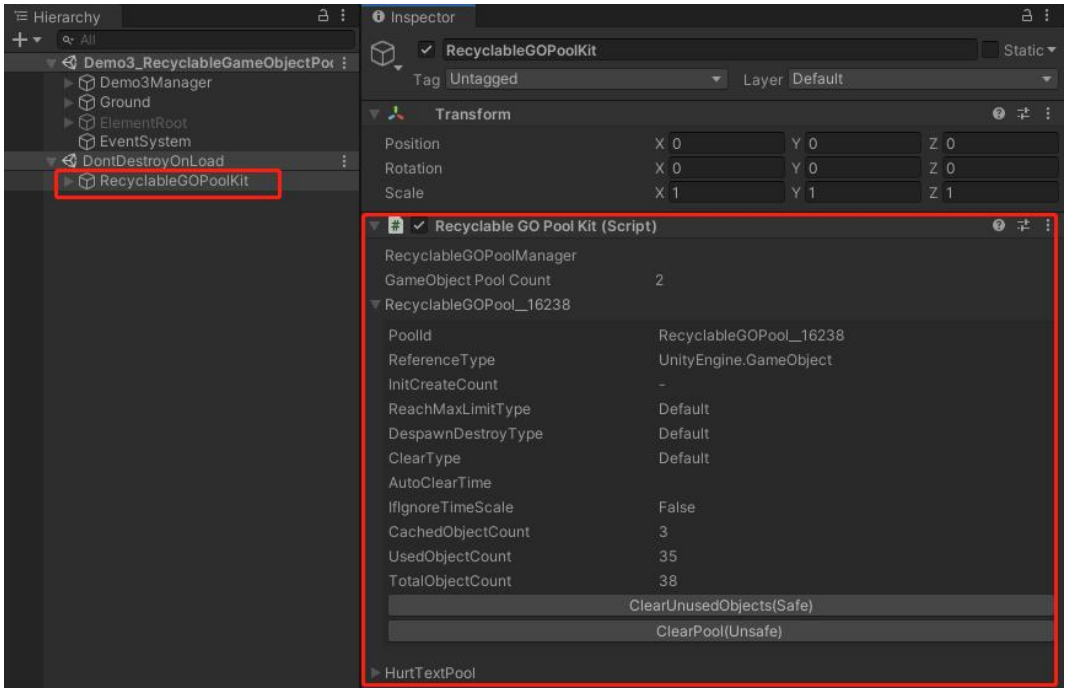
Clear unused pooled objects:

```
RecyclableGOPoolKit.Instance.ClearAllUnusedObjects();
```

Clear all pooled and used objects:

```
RecyclableGOPoolKit.Instance.ClearAllPools();
```

You can select the object of RecyclableGOPoolKit and watch the details of the pool manager:



4. Demo4_Advanced_Sound

This demo shows how to **implement a simple sound pool** by [RecyclableGOPoolKit] and [DemoSoundObject]

Here's the sound pool configuration and the implementation of sound object:

```
//Register sound pool
var soundPoolConfig = new RecyclablePoolConfig
{
    ObjectType = RecycleObjectType.RecyclableGameObject,
    ReferenceType = typeof(DemoSoundObject),
    PoolId = "DemoSoundPool",
    InitCreateCount = 10,
    ReachMaxLimitType = PoolReachMaxLimitType.RecycleOldest,
    MaxSpawnCount = 10,
};
_poolManager.RegisterPrefab(SoundTemplate, soundPoolConfig);
```

```
1 asset usage  2 usages
public class DemoSoundObject : RecyclableMonoBehaviour
{
    public AudioSource Audio;  Changed in 1 asset
    public bool _ifAutoDespawn = false;  Unchanged
    private AudioClip _clip;
    private float? _autoDespawnTime = null;
    private float _playedTime;

    1 usage
    public void Play(AudioClip clip, bool ifAutoDespawn){...}

    Frequently called  0+5 usages
    public override void OnObjectDespawn()
    {
        base.OnObjectDespawn();
        Audio.Stop();

        Audio.clip = null;
        _clip = null;
        _playedTime = 0;
        _ifAutoDespawn = true;
        _autoDespawnTime = null;
    }

    Frequently called  0+2 usages
    public override void OnObjectUpdate(float deltaTime){...}
}
```

There’s no more explanation as it’s really easy to understand.

How to get support?

You can contact me with email: samurai54@outlook.com