

zCache Object Pooling

©COPYRIGHT DIGITAL SOFTWARE/JOHAN 'ZARKOW' MUNKESTAM

Version: 1.2.0 (Unity 6.3 LTS, 6000.3 and newer; Built-in, URP and HDRP)

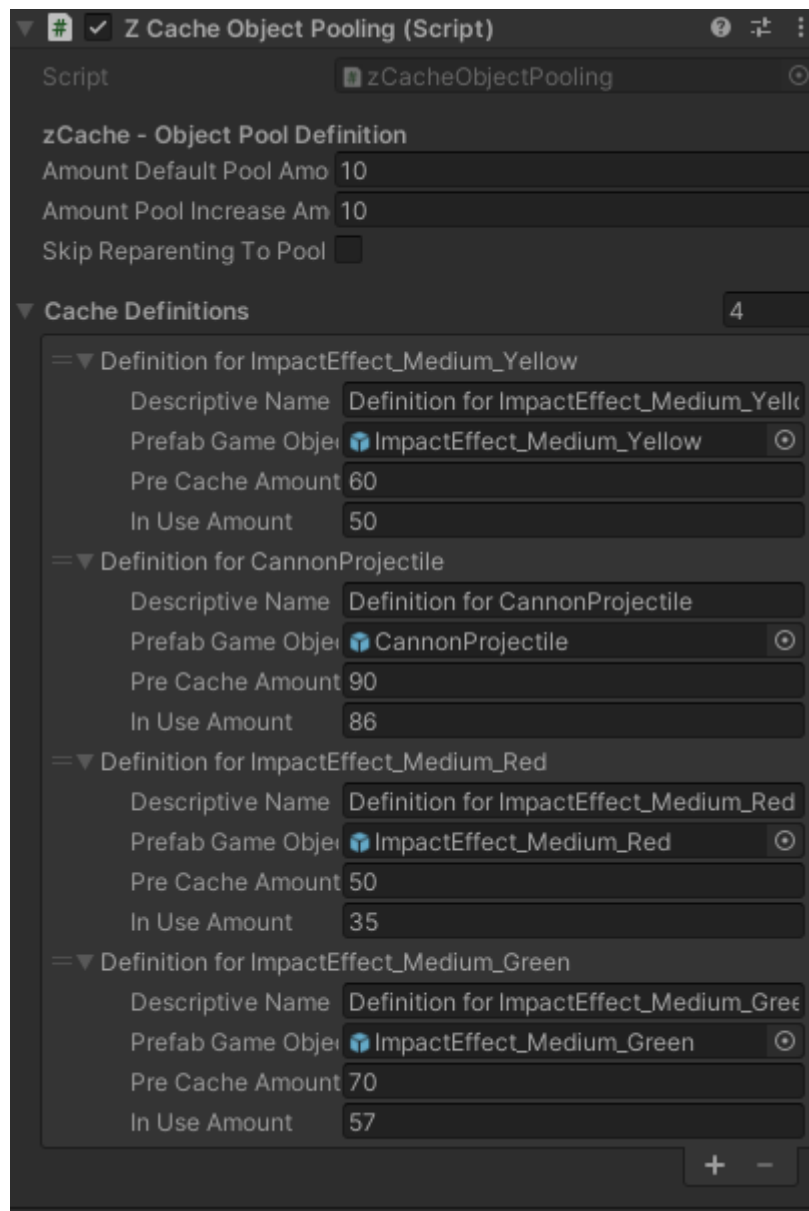
Introduction

Any project that uses projectiles, spawns monsters/opponents during play, shows impact effects or explosions, OR any other scenario where the instantiation of GameObjects occur a lot and they are thrown away later, will benefit from re-using these – this reduces the loading of data from disk and reduces the load on the GC to clean things up. Both of these actions are a common reason for micro-stutter and is important to reduce in these times where 144Hz screens and so on is starting to be fairly common amongst dedicated gamers.

Getting Started

- 1) Open the 'DemoScene' scene under
 'Assets\DigitalSoftware\zCacheObjectPooling\DemoScene\'
- 2) Press Play

How to define the cache of zCache Object Pooling



As can be seen in the DemoScene sample known effects and/or projectiles can be pre-listed in the zCache Object Pool prefab with an amount set – this will make sure that there is a populated pool at the start of the scene.

But even if one does not pre-designate an effect but later ask the zCache for it, a pool will be created at run-time and every instance will be kept tracked of. This makes the solution very flexible and can allow you to determine per scene what elements to pre-cache and still manage any scenario where other effect or projectiles appear without too much of a performance cost.

Also note that if one do not want to configure the cache by hand, one can copy the

current setup during *Play* mode by clicking on the three dots and select *Copy Component*. After when going back to design mode in the editor, select the three dots and then choose *Paste Component Values*. You have now set up the pre-cache close to what you see it being used as at run-time, without having to figure it out yourself.

Amount Default Pool Amount – The amount of elements that will be created when a run-time added pool is added. For specific amount for pre-defined pool entries, see Definition section below.

Amount Pool Increase Amount – The amount of elements that will be created every time we run out of available elements in the pool. If the effects/monster/projectile is complex, it is recommended to use a lower the value to minimize run-time impact. A better option is to then set a higher Pre Cache Amount in the specific pool entry.

Skip Reparenting To Pool – This option is for scenarios where there is a lot of effects and projectiles in use and the performance of the device (read: mobile phone) is low, and doing moves in the scene tree is enough of an impact to be noticeable. Any disabled/requeued effect/projectile is kept at the root of the scene tree, and not hidden under the Pool entry. For normal usage, leave this unchecked, for a cleaner scene view in the Editor.

Explanation of Definition entry

Descriptive Name – Optional descriptive name of the pool entry. Will be set to 'Definition for [prefabName]' for any entry that is added automatically at run-time.

Prefab Game Object – The referenced prefab element that populates this pool entry.

Pre Cache Amount – The amount of instances we create on start of the Scene. Will also be adjusted at run-time to display the current amount of elements in the pool.

In Use Amount – Only for displaying the number of elements that is currently in active use. While there is a prefab for displaying zCache Object Pooling statistics in the game view, this entry is for observing the value in the editor.

How to make use of the zCache Object Pooling

To start using the Object Pooling one merely needs to request out an element that may or may not exist in a pool entry and pre cached – if it isn't cached already, attempts will be made to create a pool entry and pre cache a default amount of elements.

For instance, in the script file for the cannon barrel '*AutoShootingGun.cs*' we find:

```
zCacheObjectPooling.ActivateFromPool(_projectilePrefab, _transform.position,
_transform.rotation);
```

This makes sure an instance of the projectile defined is put into the scene and activated, at a specific position and having a specific rotation.

Any element added into the zCache Object Pooling has an additional script added to it, '*zCacheTracking.cs*', that holds the current state of the element and holds reference to what pool it came from, to allow for fast requeue without needing to do any lookup.

For projectiles it is recommended to keep track of the trackingReference locally per projectile, but this could also be done in a requesting parent depending on design-reasons. In the script '*Projectile.cs*' for the projectile in the DemoScene we find:

```
void Start()
{
    _transform = this.transform;
    _cacheTracking = this.GetComponent<zCacheTracking>();
}
```

The local cache of the tracking reference here allows the projectile to easily, locally, to alert the zCache itself that it is ready to re-enter the pool of available entries. Often projectiles are set off into the world of the scene and logic for handling hit vs enemies or environment is done within the class itself, so it makes sense to cache and pass through the tracking reference at this place.

When the projectile hit something we have the following code:

```
// we hit something, kill projectile, show explosion
KillProjectile();

// if there is any effect linked, trigger one listed
zCacheObjectPooling.ActivateFromPool((( _impactEffects.Length >
0)?_impactEffects[Random.Range(0, _impactEffects.Length)]:null),
    hitInfo.point, Quaternion.Euler(hitInfo.normal.x,
hitInfo.normal.y, hitInfo.normal.z));
```

We find in the code a request to kill the projectile (itself):

```
private void KillProjectile()
{
    _dead = true;

    // note this projectile done, time to reQueue it to the pool
    zCacheObjectPooling.ReQueueToPool(_cacheTracking);
}
```

We don't Destroy() the item, as we want to keep it in the pool available for later use. Instead we alert the zCache Object Pooling that this element is ready to be ReQueued.

As could also be found above, there was another call to `zCacheObjectPooling.ActivateFromPool()` – this one is for spawning an effect that should trigger upon this projectile hitting some hard surface. As the projectile in the DemoScene has an array of prefabs to select from, the code above randomly picks an effect. This can be used, but perhaps more common is to detect what kind of element that was hit (material type or so on) and request the effect based on that.

The effect being spawned in the DemoScene does things differently than the projectile – where the profile may travel some distance and interact with the world, most effects have limited impact on the world and/or should only be shown for a brief time. For these scenarios there is a script included that can be dropped onto any effects to remove the need to handle the ReQueue order in code:

'*ObjectPooling_ReQueueAfterTTL.cs*' – this script will handle the time-keeping for you and alert the pool when this element is ready to re-enter into it. It is advisable to set the TimeToLive some second beyond the longest possible life it will have, so if the prefab has particle emitters with variable lifetimes, pick the life-time of the particle emitter and add the highest life time value of any sub-particle, and then there is no risk of prematurely disabling the effect – causing any visual glitches of explosions disappearing mid-animation etc.

Pool an existing object type in five lines

If your code already calls `Instantiate()` and `Destroy()` on a prefab, switching it to the pool takes five lines and no configuration. A pool entry is created at run-time the first time a prefab is requested; pre-caching it in the Definition list described above is optional.

1. Drag Prefabs\zCacheObjectPooling.prefab into the scene, once per scene.
2. using Assets.DigitalSoftware.zCacheObjectPooling.Scripts.Core;
3. var tracking = zCacheObjectPooling.ActivateFromPool(prefab, position, rotation); // instead of Instantiate(); the object is active at that position
4. Use tracking.CachedGameObject exactly as you used the instance before.
5. zCacheObjectPooling.ReQueueToPool(tracking); // instead of Destroy(); or add ObjectPooling_ReQueueAfterTTL to the prefab and skip this line for short-lived effects

`ActivateFromPool` takes an optional fourth argument, a parent Transform, for objects that should move with something else. `ReQueueToPool` also accepts the `GameObject` itself if you did not keep the tracking reference; keeping it is faster.

Statistics Prefab

The prefab `UI_zCacheObjectPoolingStatistics` is a drop-in component that allows you to display the current statistics of current pool entries, their prefabs, the tracked amount in the cache and how many elements that are currently in active use.

==[Stats]==

Descriptive Name: Definition for ImpactEffect_Medium_Yellow
Prefab Name: ImpactEffect_Medium_Yellow
Pool Stats: 47/50

Descriptive Name: Definition for CannonProjectile
Prefab Name: CannonProjectile
Pool Stats: 83/90

Descriptive Name: Definition for ImpactEffect_Medium_Red
Prefab Name: ImpactEffect_Medium_Red
Pool Stats: 51/60

Descriptive Name: Definition for ImpactEffect_Medium_Green
Prefab Name: ImpactEffect_Medium_Green
Pool Stats: 47/60

It is advisable to control the activity of this UI layer with a Debug mode for your game, to only show it when it is of interest.

By default, the data is located at the low centre-position of the primary display.

Further support

Head over to official support forums at <https://community.digitalsoftware.se/> or the Discord at <https://discord.gg/fyqtxZt>