Simple Object Pooler

Asset By: João Milone

Contact:
crossblackstudios@gmail.com

1) QuickStart:

Asset includes following folders and content:

1. **Demo:**

   1.1. Extra Scripts:
   Includes a couple of scripts to make the demo scenes. Maybe you'll find something useful to use with your projects :D

   1.2. Materials:
   Some materials for the demo scenes.

   1.3. Models:
   3D models for the scenes.

   1.4. Prefabs:
   Prefabs used in demo scenes, such as bullets, enemies and others.

   1.5. Scenes:
   Demo scenes.

   1.6. Textures:
   Textures for materials.

2. **Documentation:**
   This document.

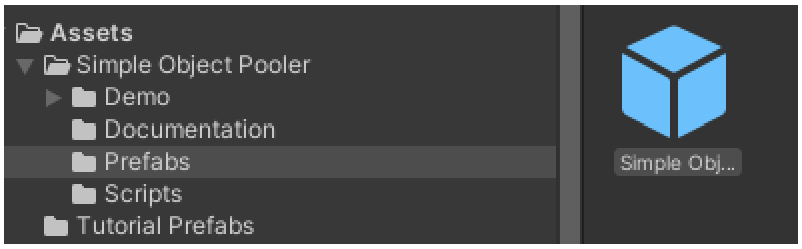
3. **Prefabs:**
   Includes Simple Object Pooler prefab to make it easier to implement in your projects

4. **Scripts:**
   Holds the *ObjectPooler.cs* script along with a *ObjectDeactivator.cs* script.

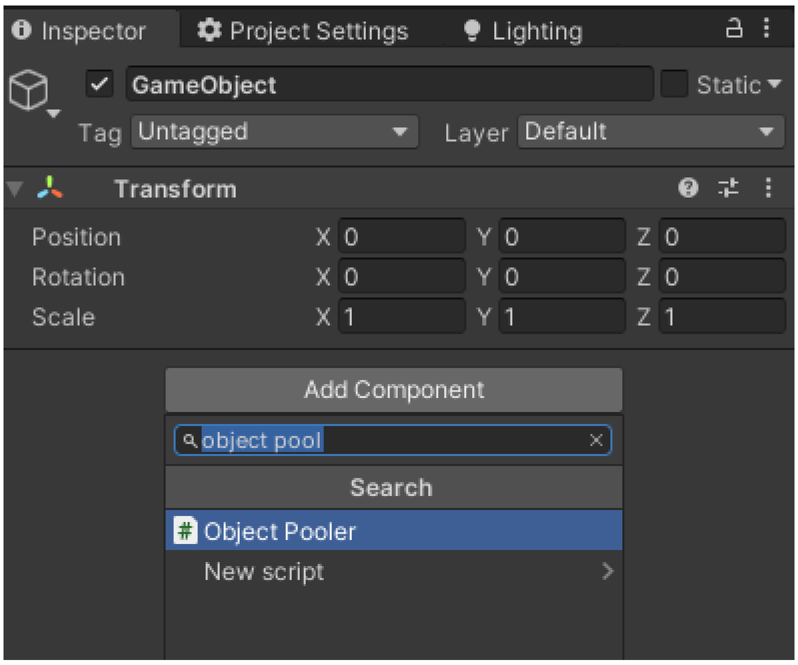
2) How to setup

2.1) Creating your Object Pooler

Select *Simple Object Pooler* in the prefabs folder and drag it into your scene.



Obs: If by any means you wish to create your Pooler Manager from scratch you may do so by simply attaching “*Object Pooler.cs*” to your desired object.

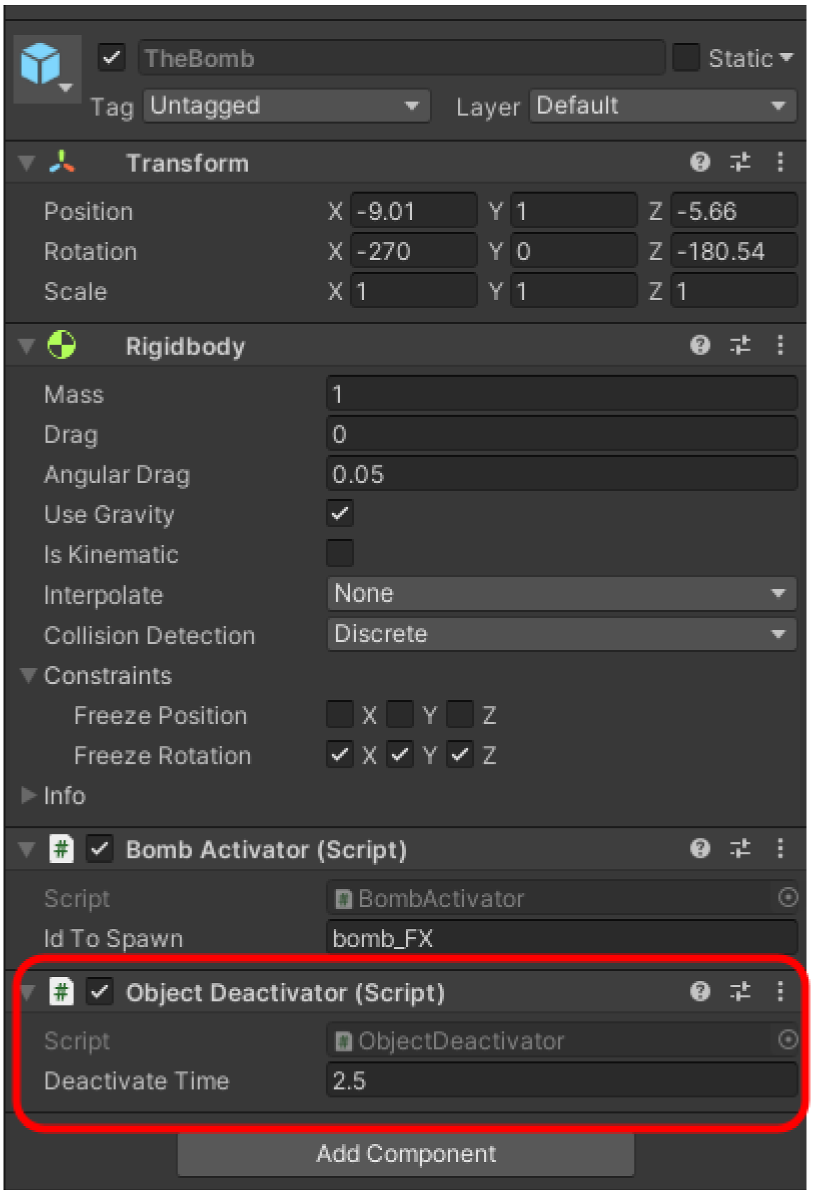


2.2) Setting up your prefabs

You're using an ObjectPooler so you don't want to destroy the objects in your pool, as you reutilize them for better performance. If by any means you wish to destroy objects in a pool to free up some memory there are functions provided for that.

But anyways, we provide a quick and simple Object*Deactivator* script to use in some cases. For example bullets. You probably don't want them flying for ever, you'll want to deactivate them after some time. So you can attach this script to these kinds of objects.

This script is optional as you can always create your own conditions to deactivate the objects. For example if your pooling enemies, you probably want to deactivate them on a health system when they die. Instead of destroying them.



TheBomb ✓ | Static
Tag Untagged | Layer Default

Transform
Position X -9.01 Y 1 Z -5.66
Rotation X -270 Y 0 Z -180.54
Scale X 1 Y 1 Z 1

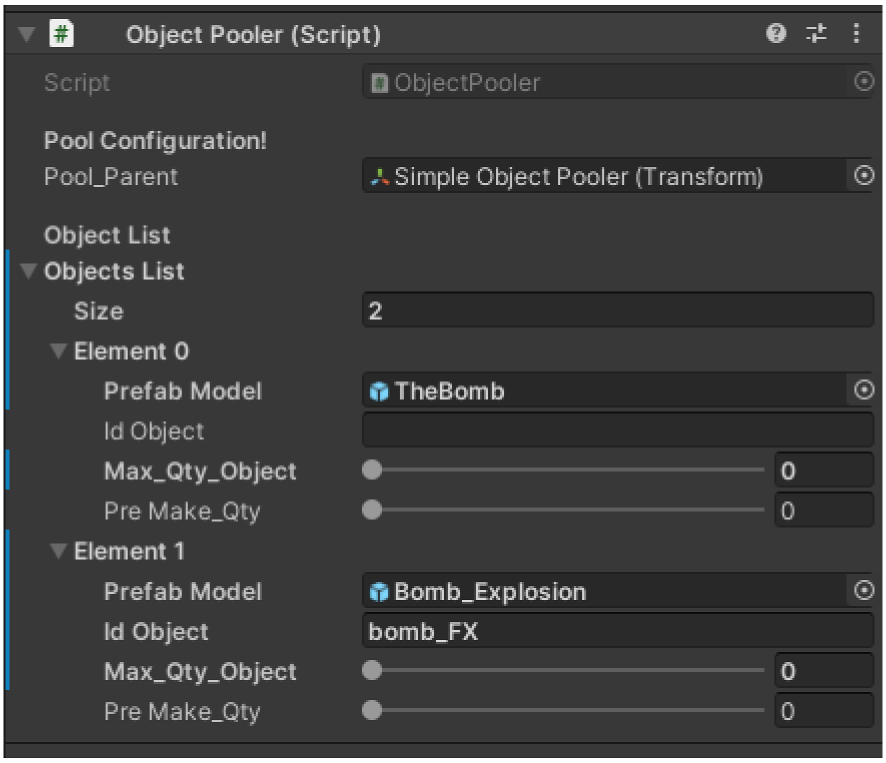
Rigidbody
Mass 1
Drag 0
Angular Drag 0.05
Use Gravity ✓
Is Kinematic ☐
Interpolate None
Collision Detection Discrete
Constraints
Freeze Position ☐ X ☐ Y ☐ Z
Freeze Rotation ✓ X ✓ Y ✓ Z
Info

Bomb Activator (Script) ✓
Script BombActivator
Id To Spawn bomb_FX

Object Deactivator (Script) ✓
Script ObjectDeactivator
Deactivate Time 2.5

2.3) Setting up your Object Pooler

This asset was made to make it easy and simple to set up your pool of objects, but still be very effective and complete with handy functions!



Object Pooler (Script)

Script	ObjectPooler
Pool Configuration!	
Pool_Parent	Simple Object Pooler (Transform)
Object List	
Objects List	
Size	2
Element 0	
Prefab Model	TheBomb
Id Object	
Max_Qty_Object	0
Pre Make_Qty	0
Element 1	
Prefab Model	Bomb_Explosion
Id Object	bomb_FX
Max_Qty_Object	0
Pre Make_Qty	0

Pool Parent: Assign a transform to be the parent of the pool. In this way objects that will be instantiated won't be all over the place in your hierarchy. The transform doesn't need to be the *Simple Object Pooler* itself, but I like it like this so it keeps everything easy to find. **Note** that this is optional if you leave this blank it still works, but objects will be instantiated with no parent.

Objects List Size: How many objects you want to create a pool to. In this demo case we only want a bomb and an explosion effect for it.

Prefab Model: This is a reference to a prefab you want to create a pool to. Just drag it from your Projects section to here.

Id Object: This is the id of the object, this means that whatever you write here is going to reference the object in the pool. It is the reference you're going to write in your code to request this object. **Note:** You can also leave this blank, if you do, the name of the object itself is going to be it's id. For example the id of our bomb will be "TheBomb". This is useful if you're still used to serialize GameObjects and you can still easily request from ObjectPooler like this:

ObjectPooler.RequestObject(*yourReferenceGameObject*.name, position);

Note 2: Serializing a string costs less to memory than serializing GameObjects.

Max_Qty_Object: The max amount of objects of that type. If you leave this at 0 the object pooler will instantiate objects with no limit as they're needed this is also good for testing and seeing how many objects you need for your case. Note that this object pooler works on demand. That means that objects will be instantiated if there isn't one available. If the limit is reached when an object is needed, the *ObjectPooler* will use the farthest object used in that pool. So keep that in mind if you want a limit to a pool. **Note:** By default the max range is set to 100, if you want more just increase the range in *ObjectPooler.cs.*

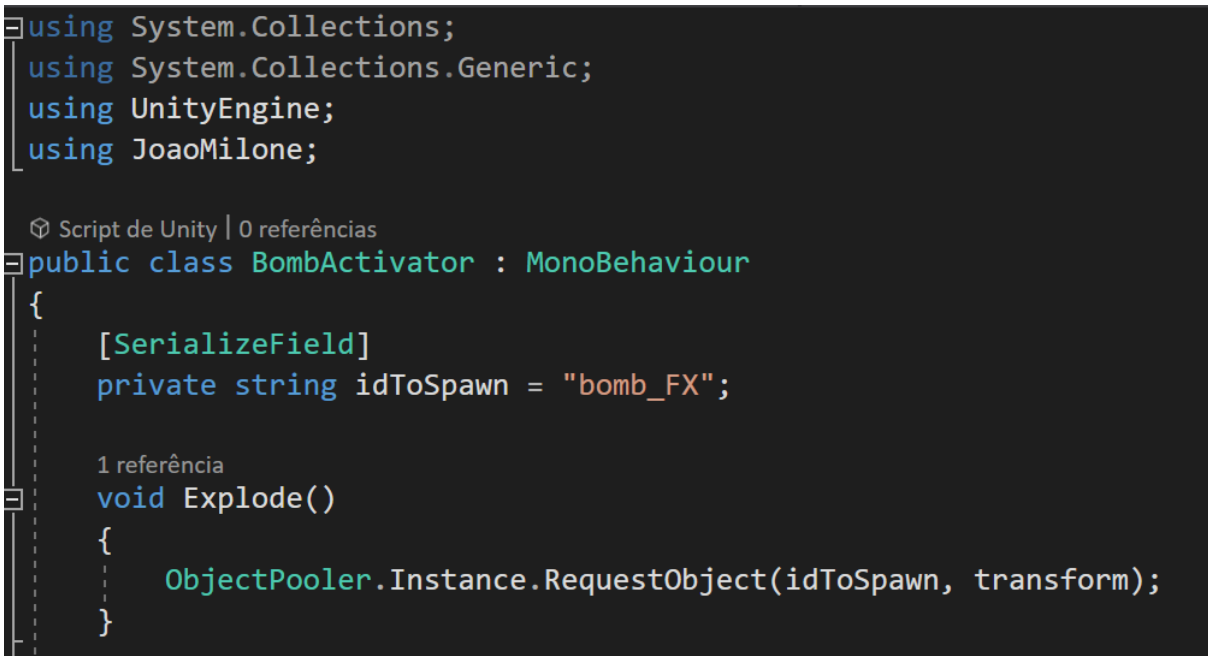
preMake_Qty: If you wish to have some objects pre created in the pool. You can set this to a desired value. **Note:** The number of pre-made objects is capped by ***Max_Qty_Object***, the code does take care of this issue just keep that in mind while setting your pool :D **Note 2:** By default the max range is set to 20, if you want more just increase the range in *ObjectPooler.cs.*

2.3) Requesting a object

First thing you'll need to import my namespace to desired script

using JoaoMilone;

After that you can simply call an object by calling the *Instance* created by *ObjectPooler* and request an object, by passing a desired id, and a position.

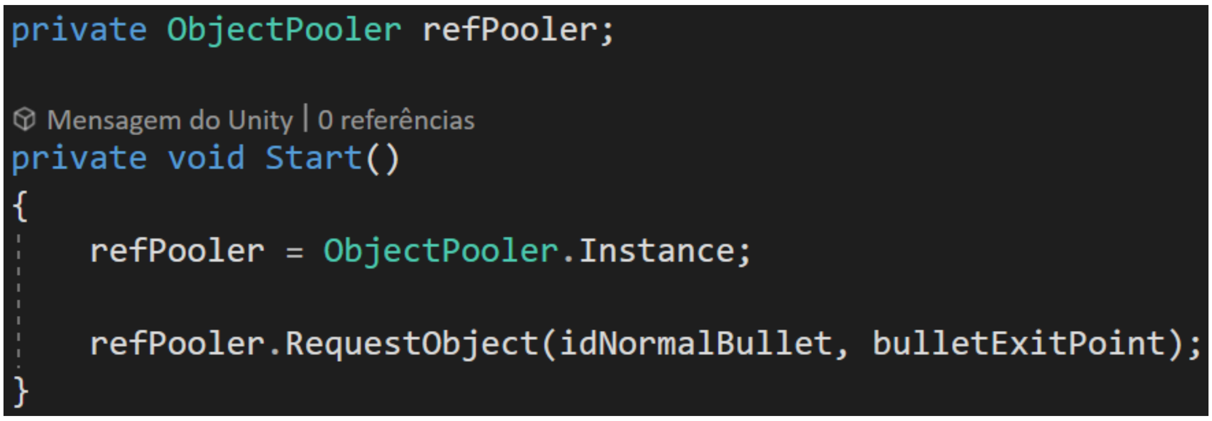


```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using JoaoMilone;

Script de Unity | 0 referências
public class BombActivator : MonoBehaviour
{
    [SerializeField]
    private string idToSpawn = "bomb_FX";

    1 referência
    void Explode()
    {
        ObjectPooler.Instance.RequestObject(idToSpawn, transform);
    }
```

Note: If you want you can create a reference to the instance of *ObjectPooler* to keep a cleaner code. Obs: Instance of *ObjectPooler* is created at *Awake()*

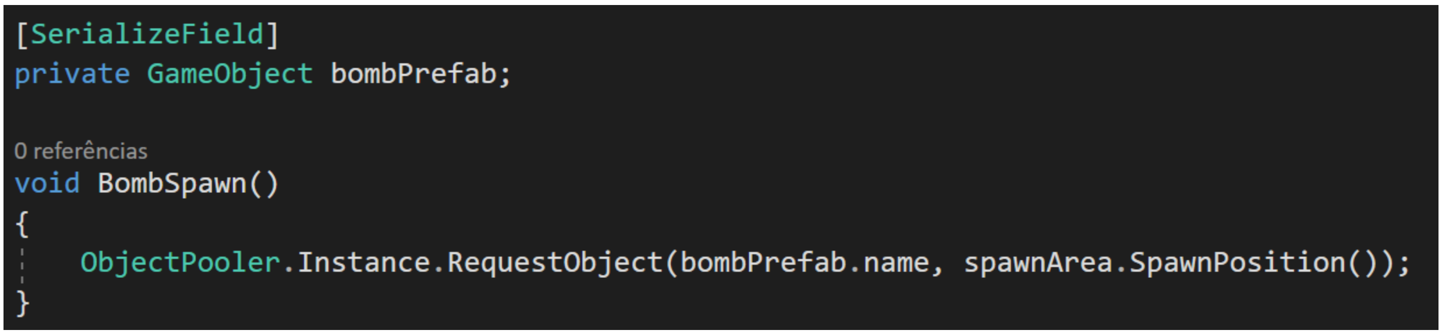


```
private ObjectPooler refPooler;

Mensagem do Unity | 0 referências
private void Start()
{
    refPooler = ObjectPooler.Instance;

    refPooler.RequestObject(idNormalBullet, bulletExitPoint);
}
```

Note 2: This is how you would call an object in the pool that had its gameObject.name as it's id.



```
[SerializeField]
private GameObject bombPrefab;

0 referências
void BombSpawn()
{
    ObjectPooler.Instance.RequestObject(bombPrefab.name, spawnArea.SpawnPosition());
}
```

2.4) Public functions in *ObjectPooler*

Note: All requests return the GameObject just like an Instantiate function.

GameObject RequestObject(string id, Transform tf):
Requests an object from the pool and assigns the transform reference to object.

GameObject RequestObject(string id, Vector3 position):
Requests an object from the pool and assigns a position to be activated! The rotation of the object will be the same as prefab.

GameObject RequestObject(string id, Vector3 position, Quaternion idt):
Requests an object from the pool and assigns a position and rotation.

GameObject RequestObjectWithLocalPosition(string id, Vector3 position):
Requests an object from the pool and assigns a local position to be activated! The rotation of the object will be the same as prefab.

int TotalObjectsInPool():
This counts how many objects are in the pool. (Activated or not).

int TotalActiveObjects():
This can be expensive to the CPU with large pools. Be careful in using this inside Updates().

int CountObjectWithID(string id):
This counts how many objects exist with a certain id.

int CountActivatedObjectWithID(string id):
This counts how many objects of certain id are active.

void ClearPoolWithID(string id):
This deletes every object with a certain id with an intent to clear some memory. Objects with this id can still be spawned again if you want.

void ClearEntirePool():
This will DELETE every object in the pool. Objects can still be spawned again if you want.

Hope you make great games with this! Thank you!

Obs: This document may have been updated check this link for current version:
https://docs.google.com/document/d/1fFBaRc-xrPOEmI23uBmKZPXoSN3G7vRygWnutA-jmxo/edit?usp=sharing