Easy Pool Manager

Table des matières

Easy Pool Manager

Introduction

You create and destroy many object during your games, but it has a strong impact on performance. Many of them, may simply be reused.
Easy Pool Manager allows you to implement quickly and easily an instance pooling solution, in order to use the maximum of the already created objects.

Concept

EPM groups the various objects that you use regularly in pools. The instatiation & destruction of an object will be managed by EMP, so you will just to replace methods "Instantiate" and "Destroy" by Spawn & Despawn methods provided by EPM.

All these pools is managed by a Pool Manager. EPM has 2 methods for creating all its elements (PoolManager, Pool, Objects) :

- **Dynamic**

No advance preparation is required, all components PoolManager, Pool, pooled object are created dynamically.Only the replacement of methods "Instantiate" & "Destroy" must be made within your script. At the first use of the method Spawn or SpawnStart PoolManager will be created automatically, and a pool for this object if none exists.

If this method allows rapid migration, it is not avoided the first instantiation during playmode.

- **Static (Look at PoolManager Component)**

This mode requires the prior creation of PoolManager and pools in your scene. However, no dynamic creation will be necessary in play mode if your pools are set correctly. On first use of the Spawn or SpawnStart method, as the pool is already existing, the poolmanger will use this object, eliminating the dynamic creation.

The replacement of methods "Instantiate" & "Destroy" must be made within your script.

- EPM can handle the two methods simultaneously.

Migration & use in scripts

Migration & use in scripts

Migration

Using Easy Pool Manager requires changing your existing scripts. We must replace the "**Instantiate**" methods by PoolManager.Spawn and "**Destroy**" by PoolManager.Despawn.These methods have the same mandatory parameters as the original methods of Unity.

Unity messages orders

In the case of the conventional use of Unity, here is the order in which messages are received.

A creation following a call to "**Instantiate"**
1 - Awake
2 - Enable
3 - Start

These three messages will be received by the object for each call to the method Instantiate.

A destruction following a call to "**Destroy"**
1 - Disable
2 - Destroy

These two messages will be received by the object for each call to the method Destroy.

Easy Pool Manager messages orders

Using Easy Pool Manager brings some change in the reception of messages and their frequency. Indeed with Easy Pool Manager objects are already present and are never destroyed.

So there are two cases, when an object pool is used for the first time, and when the object has already been used.

First use
1 - Awake
2 - Enable
3 - Spawn
4 - Start

Spawn The message is added compared to a conventional use

After first use
1 - Enable
2 - Spawn

As the object is not destroyed, only Enable & Spawn messages are received.

You must be careful to "code" that you put in the Awake & Start messages

The message "**OnDestroy**" is not sent anymore, it is replaced by "**OnDespawn**", so you must move the code in the new method "OnDespawn"

Garbage Collector

Garbage Collector

Easy Pool Manager has Gabarbage Collector option, This allows you to permanently destroy the instances that are no longer used for some time.

This feature can be used in two different ways :

* In automatic, by enabling the option in the inspector of PoolManager
* Via a script to launch it on demand

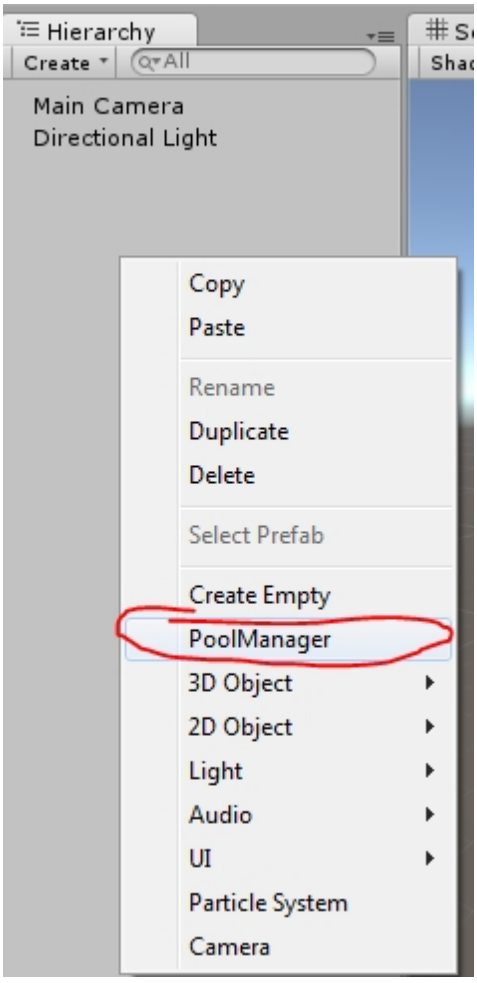
Both methods can be used in conjunction.

PoolManager Component

If you want to avoid dynamic creations, you must create your PoolManager& pools in design mode.

How to create

1- Right click int he hierarchy
2- Select PoolManger
3- The poolManager is created and automaticaly selected.

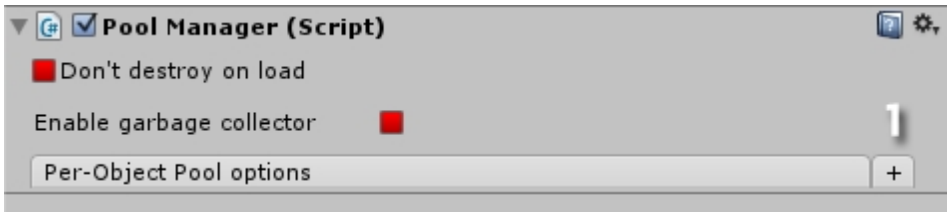


How to add a pool

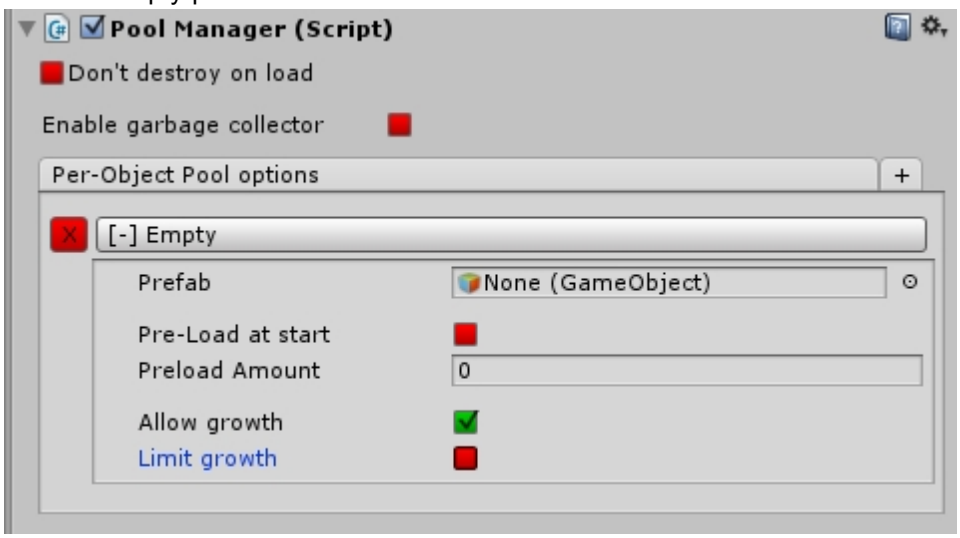
How to add a pool

0- Create the PoolManager (look here)

1- Click on "+" (1)



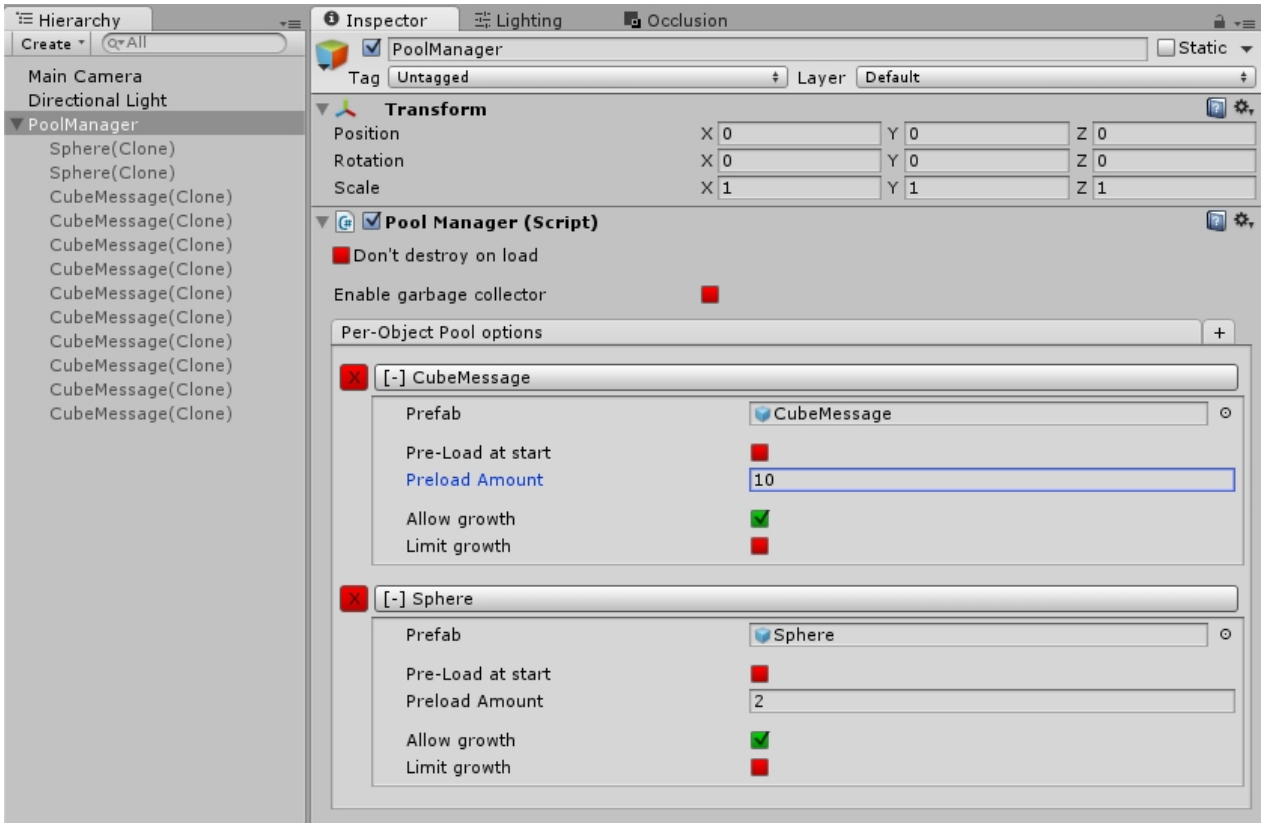
2- A new empty pool is created



3- Drag'n drop a gameobject or a prefab in **Prefab** fields
4- Set **Preload Amount**

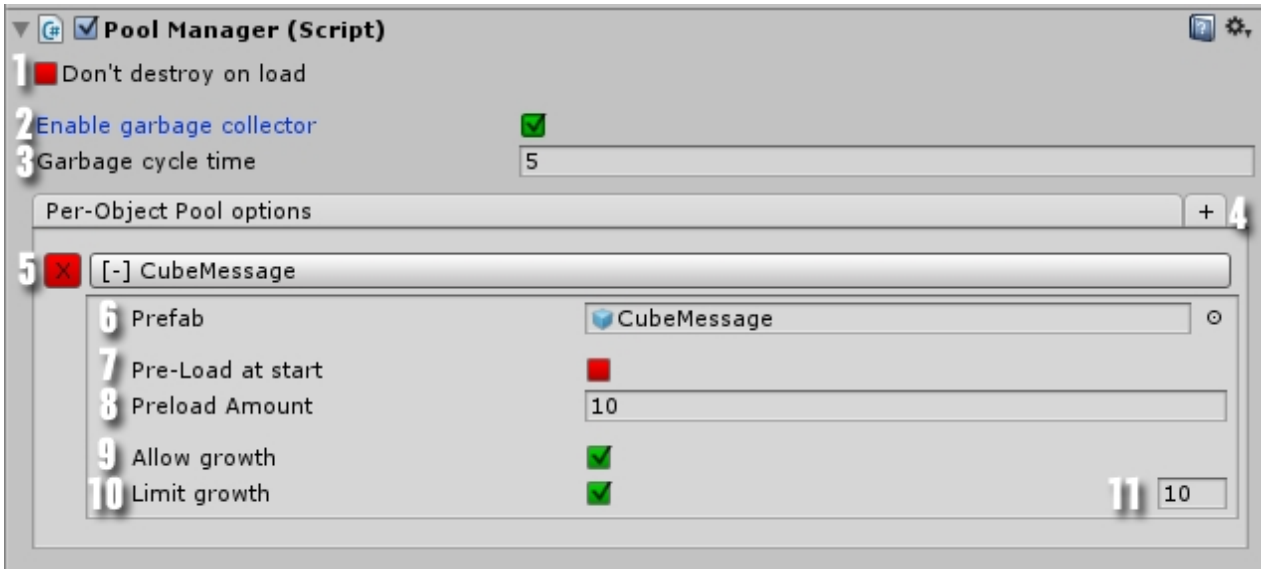
Note :
You can create as many pool as you want. However, you can not create two pools for the same object.

Example



Inspector

Inspector



1 - Make the PoolManager and content not be destroyed automatically when loading a new scene

2 - Enables the garbage collector

3 - Time between each launch ok the garbage collector

4 - To add new pool

5 - To destroy a pool

6 - The reference object for the pool creation.

7 - The content of the pool will be created on startup, if not directly in design mode

8 - The number of pre-loaded instance

9 - Whether to allow the extension of the pool according to the request

10 - 11 If the extension is allowed, sets a limit.

DespawnObject Component

Enabling /Disabling instead of instantiating/destroying poses no problem in the case of simple gameobject, but it complicates the process when the gameobject has effects such as:
- Particles system
- Sound
- Trail
- Rigidbody

Indeed, these components may need to be reset after use, or gameobject must wait for one of these components to be truly destroyed.

Rather than you let yourself manage this aspect via Spawn & despawn messages, EPM is proivided with component allowing you to simply perform these tasks "DespawnObject". You can of course create additional scripts that use the Spawn & Despawn event in addition to this component

How it works?

This component must be placed on the GameObject managed by EPM, and it will do the rest at the time of spawn and despawn.

At **Spawn** depending on the components found
- Disable kinematic for RigidBody.
- Play the particles system.
- Play sound.
- Init trail.

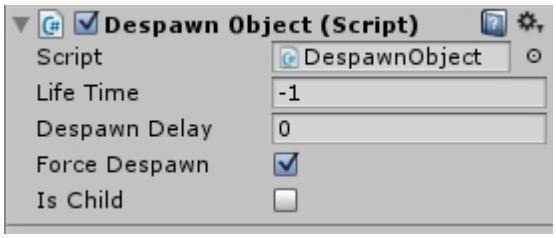
At **Despawn** depending on the components found
- Enable kinematic for rigidbody to reset physic
- Stop particule system => the gameobject will be available for another spawn only when no particles will be more to the screen.
- Stop sound if is on loop, otherwise the gameobject will be available for another spawn only when the sound is finished.
- Reset Trail.

But as you may have read the effects can delay disabling of gameobject. Next use cases this will not matter, while in other cases the delay could be harmful (eg a pojectile, he would continue his run in wait for the particle effect wears off)

How to use it?

Add the script Despawn Objetc (EasyPoolManager/Plugins/DespawnObject.cs) on you prefab, or use the button Add component on you GameObject inspector (Easy Pool Manager => DespawnObject.cs)

Inspector



Life Time : Define the life time in seconde. -1 = infinity. Usefull for Exmplosion, bullet for example.
Despawn Delay : Delay in second before its deactivation.

Force Despawn & Is Child :
These two values are used to define the behavior of its deactivation gameobject.

1 - The GameObject must be disabled, and its effects. In this case deactivation is instant
Force Despawn = **TRUE**
Is Child = **FALSE**

2 - The GameObject must be disabled, but the effects should complete successfully (over particle, wait for the sound etc ...)
Force Despawn = **FALSE**
Is Child = **FALSE**

3 - The GameObject must be disabled, but the effects should be completed correctly, however the main GameObject should no longer be active until the complete deactivation of its effects.

This one requires a little preparation GameObject. The scripts to be disabled instantly be placed on the root, while the effects will be placed on child gameobject. The component must be place in the root and in the child.

On Root
Force Despawn = **FALSE**
Is Child = **FALSE**

On Child with effect should complete successfully.
Force Despawn = **FALSE**
Is Child = **TRUE**

Example

PoolManager API

Description

This class allows you to manipulate the pools and create object instances or destroy them.

Static functions for Pool management

CreatePool : Creates a pool for gameobject, , with a number of prefetch object.

GetPool : Returns in array all instances of a poll, for a given gameobject.

GetObjectCountInPool : Returns the number of gameobject contained in a pool.

GetActiveObjectCountInPool : Returns the number of active gameobject contained in a pool.

GetPoolCount : Returns the number of Pool contained in the poolmanger.

GetPooledObjects : Returns in array the references gameobject for each pool.

DestroyPool : Destroye a pool and all the gamobject it contains

GarbageCollector : Destroy all object unused since a time.

Static functions for spawn/despawn management

Spawn : Spawn a gameobject and return its reference.

SpawnStart : Spawn a gameobject and return its reference, without activated.

SpawnEnd : Activates gameobject spawned with a method SpawnStart.

Despawn : Despawn a game object.

Messages

OnSpawn : Callback when the object is spawned and activated.

OnDespawn : Callback when the object will despawn.

PoolManager.CreatePool

public static void **CreatePool**(GameObject obj, int amount)

Parameters

obj : The reference gameobject
amount : The number of instance created.

Description

Creates a pool for gameobject, with a number of prefetch object. If the PoolManager doesn't exit, it is automatically created.

```
public class CreatePool : MonoBehaviour {

	public GameObject myRef;
	public int amount;

	void Awake(){
		PoolManager.CreatePool( myRef, amount);

		InvokeRepeating( "CreateObj",0f,0.8f);
	}

	void CreateObj(){
		PoolManager.Spawn( myRef, transform.position, Random.rotation);
	}
}
```

PoolManager.GetPool

public static GameObject[] **GetPool**(GameObject obj)

Parameters
obj : The reference gameobject of the pool.

Returns
An array of gameobject if the pool exists, otherwise an empty array.

Description
Returns in array all instances of a pool, for a given gameobject.

If the PoolManager doesn't exit, it is automatically created.

```
using UnityEngine;
using System.Collections;

public class GetPool : MonoBehaviour {

	public GameObject myRef;
	public int amount;

	void Awake(){
		PoolManager.CreatePool( myRef, amount);
	}


	void Start(){

		//Gives a random color to each object contained in the pool
		GameObject[] objs = PoolManager.GetPool( myRef );
		foreach( GameObject obj in objs){
			obj.GetComponent<Renderer>().material.color = new
Color( Random.Range(0.0f,1.0f),  Random.Range(0.0f,1.0f), Random.Range(0.0f,1.0f));
		}


		InvokeRepeating( "CreateObj",0f,0.8f);
	}

	void CreateObj(){
		PoolManager.Spawn( myRef, transform.position, Random.rotation);
	}
}
```

PoolManager.GetObjectCountInPool

public static int **GetObjectCountInPool**(GameObject obj)

Parameters
obj : The reference gameobject of the pool.

Returns

int the number of gameobject contained in the pool, otherwise -1 if the pool doesn't exist

Description
Returns the number of gameobject contained in a pool.

If the PoolManager doesn't exit, it is automatically created.

```
using UnityEngine;
using System.Collections;

public class GetObjectCountInPool : MonoBehaviour {

	public GameObject myRef;
	public int amount;

	void Awake(){
		PoolManager.CreatePool( myRef, amount);

		InvokeRepeating( "CreateObj",0f,0.8f);
	}

	void Update(){
		Debug.Log( PoolManager.GetObjectCountInPool( myRef ));
	}

	void CreateObj(){
		PoolManager.Spawn( myRef, transform.position, Random.rotation);
	}
}
```

PoolManager.GetActiveObjectCountInPool

public static int **GetActiveObjectCountInPool**(GameObject obj)

Parameters
obj : The reference gameobject of the pool.

Returns
int the number of active gameobject contained in the pool, otherwise -1 if the pool doesn't exist

Description
Returns the number of active gameobject contained in a pool.

If the PoolManager doesn't exit, it is automatically created.

```
using UnityEngine;
using System.Collections;

public class GetActiveObjectCountInPool : MonoBehaviour {

	public GameObject myRef;
	public int amount;
```

```
	void Awake(){
		PoolManager.CreatePool( myRef, amount);

		InvokeRepeating( "CreateObj",0f,0.8f);
	}

	void Update(){
		Debug.Log( PoolManager.GetActiveObjectCountInPool(myRef) + " / " +
PoolManager.GetObjectCountInPool( myRef ));
	}

	void CreateObj(){
		PoolManager.Spawn( myRef, transform.position, Random.rotation);
	}
}
```

PoolManager.GetPoolCount

public static int **GetPoolCount**()

Returns

int the number of Pool contained in the poolmanger.

Description

Returns the number of Pool contained in the poolmanger.

If the PoolManager doesn't exit, it is automatically created.

```
using UnityEngine;
using System.Collections;

public class GetPoolCount : MonoBehaviour {

	public GameObject myRef;
	public GameObject myRef2;
	public int amount;

	void Awake(){
		PoolManager.CreatePool( myRef, amount);
		PoolManager.CreatePool( myRef2, amount);

		InvokeRepeating( "CreateObj",0f,0.8f);
	}

	void Update(){
		Debug.Log( PoolManager.GetPoolCount() );
	}
	void CreateObj(){
		PoolManager.Spawn( myRef, transform.position, Random.rotation);

		PoolManager.Spawn( myRef2, transform.position, Random.rotation);
	}
}
```

PoolManager.GetPooledObjects

public static GameObject[] **GetPooledObjects**()

Returns
An array of gameobject some pool exists, otherwise an empty array.

Description
Returns in array the references gameobject for each pool.

If the PoolManager doesn't exit, it is automatically created.

```
using UnityEngine;
using System.Collections;

public class GetPooledObjects : MonoBehaviour {

	public GameObject myRef;
	public GameObject myRef2;
	public int amount;

	void Awake(){
		PoolManager.CreatePool( myRef, amount);
		PoolManager.CreatePool( myRef2, amount);

		InvokeRepeating( "CreateObj",0f,0.8f);
	}

	void Update(){
		string label = "";
		GameObject[] objs = PoolManager.GetPooledObjects();
		foreach( GameObject obj in objs){
			label += obj.name + " - " ;
		}

		Debug.Log(label);
	}


	void CreateObj(){
		PoolManager.Spawn( myRef, transform.position, Random.rotation);

		PoolManager.Spawn( myRef2, transform.position, Random.rotation);
	}
}
```

PoolManager.DestroyPool

public static bool **DestroyPool**(GameObject obj, bool editor=false)

Parameters
obj : The reference gameobject of the pool.

editor : true = DestroyImmediate for inspector

Description
Destroyed a pool and all the gamobject it contains

If the PoolManager doesn't exit, it is automatically created.

```
using UnityEngine;
using System.Collections;

public class DestroyPool : MonoBehaviour {

	public GameObject myRef;
	public int amount;

	void Awake(){
		PoolManager.CreatePool( myRef, amount);

		InvokeRepeating( "CreateObj",0f,0.8f);
		Invoke("DestroyThePool",4);
	}

	void CreateObj(){
		PoolManager.Spawn( myRef, transform.position, Random.rotation);
	}

	void DestroyThePool(){
		CancelInvoke();
		PoolManager.DestroyPool( myRef);
	}
}
```

PoolManager.GarbageCollector

public static void **GarbageCollector**(float inactivity=-1)

Parameters
inactivity **:** Inactivity time since the first activation (by default =-1 It is time to set in the pool)

Description
Destroy all object unused since a time.

If the PoolManager doesn't exit, it is automatically created.

```
using UnityEngine;
using System.Collections;

public class GarbageCollector : MonoBehaviour {

	public GameObject myRef;
	public int amount;

	void Awake(){
```

```
		PoolManager.CreatePool( myRef, amount);

		InvokeRepeating( "CreateObj",0f,0.8f);
	}

	void Update(){

		if (Input.GetKeyDown(KeyCode.Space)){
			PoolManager.GarbageCollector(2);
		}
	}

	void CreateObj(){
		PoolManager.Spawn( myRef, transform.position, Random.rotation);
	}


}
```

PoolManager.Spawn

public static GameObject Spawn(GameObject obj,Vector3 position,Quaternion rotation,int amount = 2, bool allowGrowth=true, int limit=-1)

Parameters

obj : The reference gameobject
position : **P**osition for the new object.
rotation : Rotation for hte new object

Optional parameters

amount : the number of instance created by default if the pool does not exist
allowGrowth : Enable or disable automatic expanding pool
limit : The extension limit, -1 if no limit

Returns

GameObject the gameobject reference from the pool

Description

Spawn a gameobject and return its reference.

The message "**OnSpawnStart**" is broadcasted on the new gameObject (Before activation).
The message "**OnSpawn**" is broadcasted on the new gameObject (After activation).

If the PoolManager doesn't exist, it is automatically created.
If the pool doesn't exist, it is automatically created reltive to the optional parameters

```
using UnityEngine;
using System.Collections;

public class Spawn : MonoBehaviour {

	public GameObject myRef;
	public int amount;

	void Awake(){
```

```
		InvokeRepeating( "CreateObj",0f,0.8f);
	}

	void CreateObj(){
		PoolManager.Spawn( myRef, transform.position, Random.rotation,10);
	}
}
```

PoolManager.SpawnStart

public static GameObject Spawn(GameObject obj,Vector3 position,Quaternion rotation,int amount = 2, bool allowGrowth=true, int limit=-1)

Parameters

obj : The reference gameobject
position : Position for the new object.
rotation : Rotation for hte new object

Optional parameters

amount : the number of instance created by default if the pool does not exist
allowGrowth : Enable or disable automatic expanding pool
limit : The extension limit, -1 if no limit

Returns

GameObject the gameobject reference from the pool

Description

Spawn a gameobject and return its reference without activated. The message "**OnSpawnStart**" is broadcasted on the new gameObject.

This allows you to perform operations on the new object before its activation.

If the PoolManager doesn't exist, it is automatically created.
If the pool doesn't exist, it is automatically created reltive to the optional parameters

```
using UnityEngine;
using System.Collections;

public class SpawnStartEnd : MonoBehaviour {

	public GameObject myRef;

	void Awake(){
		InvokeRepeating( "CreateObj",0f,0.8f);
	}

	void CreateObj(){
		GameObject obj = PoolManager.SpawnStart( myRef, transform.position,
Random.rotation,10);
		obj.transform.localScale = new
Vector3( Random.Range(0.5f,2f),Random.Range(0.5f,2f),Random.Range(0.5f,2f));
		PoolManager.SpawnEnd( obj);
	}
```

```
}
```

PoolManager.SpawnEnd

public static void SpawnEnd(GameObject obj)

Parameters

obj : gameobject to activated

Description

Activates gameobject spawned with a method SpawnStart. The message "**OnSpawn**" is broadcasted on the new gameObject.

This allows you to perform operations on the new object before its activation.

If the PoolManager doesn't exist, it is automatically created.
If the pool doesn't exist, it is automatically created reltive to the optional parameters

```
using UnityEngine;
using System.Collections;

public class SpawnStartEnd : MonoBehaviour {

	public GameObject myRef;

	void Awake(){
		InvokeRepeating( "CreateObj",0f,0.8f);
	}

	void CreateObj(){
		GameObject obj = PoolManager.SpawnStart( myRef, transform.position,
Random.rotation,10);
		obj.transform.localScale = new
Vector3( Random.Range(0.5f,2f),Random.Range(0.5f,2f),Random.Range(0.5f,2f));
		PoolManager.SpawnEnd( obj);
	}
}
```

PoolManager.Despawn

blic static void DespawnObject(GameObject obj)

Parameters

obj : The reference gameobject of the pool.

Description

Despawn a game object. The message "**OnDespawn**" is broadcasted on the new gameObject.

This allows you to perform operations on the new object before its activation.

If the PoolManager doesn't exist, it is automatically created.
If the pool doesn't exist, it is automatically created reltive to the optional parameters

```
using UnityEngine;
using System.Collections;

public class ExampleDespawn : MonoBehaviour {

	public GameObject myRef;

	void Awake(){
		InvokeRepeating( "CreateObj",0f,0.8f);
	}

	void Update(){

		if (Input.GetKeyDown(KeyCode.Space)){
			GameObject[] objs = PoolManager.GetPool( myRef);
			for (int i=0;i<objs.Length;i++){
				if (objs[i].activeInHierarchy){
					PoolManager.DespawnObject( objs[i]);
					return;
				}
			}
		}
	}

	void CreateObj(){
		PoolManager.Spawn( myRef, transform.position, Random.rotation,2,false);
	}

}
```

Messages

PoolManager

OnSpawn : Callback when the object is spawned and activated.

OnDespawn : Callback when the object will despawn.

Description

These messages are sent to the spawned objects.

```
using UnityEngine;
using System.Collections;

// Apply this script on pooled object
public class ExampleMessageMessage : MonoBehaviour {
```

```
	void OnSpawn(){
		Debug.Log("Cube : Spawn");
	}

	void OnDespawn(){
		Debug.Log("Cube : Despawn");
	}
}
```