Pools

Pools is an asset with a couple of classes that will allow you to create and work with pools easily and simply.

Connection

To work with a pool, include the `Redcode.Pools` namespace.

```
using Redcode.Pools
```

Pool

Create a pool from code

Use the `Pool.Create<T>(T component, int count)` method to create a pool. As the first parameter, specify any component on the game object in the scene or prefab, in the second - the maximum number of clones that this pool should create.

```
varpool = Pool.Create(prefab, 10);
```

In the parameter, you can specify 0 so that the pool does not have a limit on the number of clones. Also, you can specify `container` as the third parameter, to which all clones will be added.

```
var container = someObject.transform;
varpool = Pool.Create(prefab, 0, container);
```

Request object

Use the `Get` method to get a free object from the pool. Calling this function will create a clone if there is no free one and if the limit on the maximum number has not been reached. Otherwise, `null` will be returned.

```
Enemy enemy = pool.Get();

if (enemy != null) { ... }
```

Unused pool objects are inactive by default. When an object is requested from the pool, it becomes active.

Return object

When the requested object is no longer needed, it must be returned to the pool. To do this, use the `Take` method.

```
pool.Take(enemy);
```

When an object is returned to the pool, it becomes inactive.

> If the object that is returned does not exist in the pool (that is, it was not created by the pool itself), then it will be added to it and become a child object of the container (about them later) of the pool.

Waiting for a free object

If you want to wait for a free object in the pool, use WaitForFreeObject.

```
yield return pool.WaitForFreeObject();
var enemy = pool.Get();
```

When a free object appears in the pool, your coroutine will continue executing.

Instant creation of pool objects

If you want the pool to pre-create clones, use the `NonLazy` method.

```
var pool = Pool.Create(prefab, 10).NonLazy();
```

> For this method to be effective, the pool must have a limit on the number of clones it can create.

Setting new limits on the number of clones

You can use the Count property to get the maximum number of clones, and the SetCount method to set it. When setting a new quantity, if it is less than the previous one, then those clones located at the end of the list will be removed, even if they were not returned to the pool. If in such a situation it is necessary not to remove clones, but only from the internal list, then specify false as the second parameter in the method.

```
pool.SetCount(5, false); // Clones will not be removed.
```

Installing a new container

You can specify a new clone container by using the `SetContainer` method. The clones will be moved to a new container wherever they are. If you want the clones in the new container to be located locally instead of keeping their position in the global space, set the second parameter to `false`.

```
pool.SetContainer(newContainer, false);
```

Clean up the pool

Use the Clear method to clear the object pool. You can also specify whether clones should be removed from the scene or not.

```
pool.clear();
```

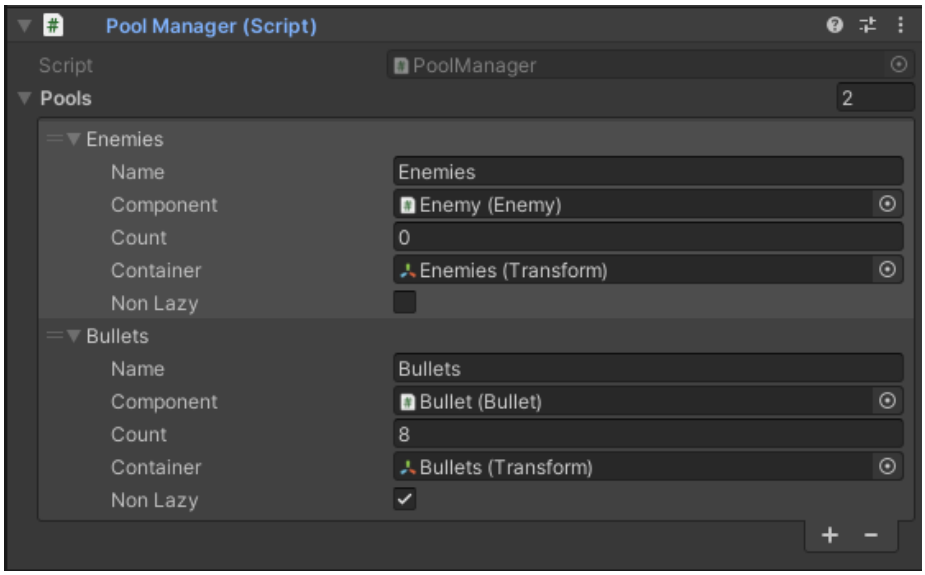
IPoolObject

Sometimes it is necessary that the objects returned from the pool are first processed in a special way, for example, an enemy that was killed and returned to the pool, should look alive on the next request, not dead. To do this, use the IPoolObject interface, implement it in the desired script as it should be processed clone before being returned from the pool.

```
public class Enemy : MonoBehaviour, IPoolObject
{
    public void OnGettingFromPool()
    {
        print("Reset");
    }
}
```

Pool manager

Use the pool manager in the scene to do all the customization you want on the pools and then use them in your game logic. Add PoolManager script to empty game object and configure it.



Each pool in the manager has its own name, it can be used to find the desired pool in the manager. Add a link to the manager to any script you need and use it to get pools or immediately objects from pools, as in the examples below.

```
varpool = manager.GetPool<Enemy>(1); // Get pool at index 1.
var clone = manager.GetFromPool<Enemy>(1) // Request an object from the pool at index 1.
```

```
manager.TakeToPool(1, clone); // Return the clone to the pool at index 1.

// ------------------------------------------------ ---------------------------------

varpool = manager.GetPool<Enemy>(); // Get the first matching Pool<Enemy> from the list.
var clone = manager.GetFromPool<Enemy>(); // Request a clone from the first matching pool Po
manager.TakeToPool(clone); // Return the clone to the first matching pool at index 1.

// ------------------------------------------------ ---------------------------------
varpool = manager.GetPool<Enemy>("Enemy"); // Get Pool<Enemy> by name.
var clone = manager.GetFromPool<Enemy>("Enemy"); // Request a clone from Pool<Enemy>() by na
manager.TakeToPool("Enemy", clone); // Return the clone to the first matching pool with the
```

Good luck!