**Object pool**

Nombre:

Object pool

Clasificación del patrón:

Creacional

Intención:

Inicializar objetos sin destruirlos posteriormente con la finalidad de reciclarlos rápidamente, disminuyendo el tiempo de ejecución de los programas.

Motivación:

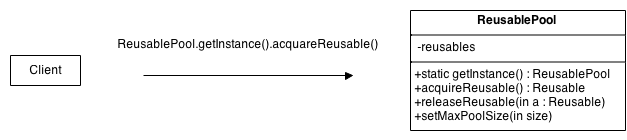
Al observar que los programas tenían que crear y posteriormente destruir los objetos que generalmente eran muy utilizados, se hizo necesario crear un patrón de diseño para agilizar y optimizar los programas.

Aplicabilidad:

Se debe usar cuando:

* Un objeto es requerido muchas veces.
* Cuando se tiene muchos procesos al mismo tiempo.

Estructura:



Participantes:

* Reusable - Las instancias de las clases en este rol colaboran con otros objetos durante un tiempo limitado, por lo que ya no son necesarias para esa colaboración.
* Client - Las instancias de clases en este rol utilizan objetos reutilizables.
* ReusablePool - Las instancias de las clases en este rol administran objetos reutilizables para que los utilicen los objetos del cliente.

Colaboraciones:

* Un objeto cliente accede a ReusablePool y llama al método acquireReusable, este método se utiliza para llamar a un ReusableObject.
* Un ReusablePoolObject mantiene una colección de ReusableObjects y se utiliza para contener a un conjunto de ReusableObject que no están en uso.

Ventajas:

* Control completo sobre la creación
* Aumento significaivo del rendimiento.

Desventajas:

* Se dificulta su implementación en ambientes multihilo
* La clase ReusablePool está diseñada para ser una clase singleton.

Implementación:

Los objetos del cliente pasan un Reusableobjeto al método de un ReusablePoolobjeto releaseReusable cuando terminan con el objeto. El releaseReusablemétodo devuelve un Reusableobjeto al conjunto de Reusableobjetos que no están en uso.

Si hay algún Reusableobjeto en la agrupación cuando acquireReusablese llama al método, elimina un Reusableobjeto de la agrupación y lo devuelve. Si el grupo está vacío, el acquireReusablemétodo crea un Reusableobjeto si puede. Si el acquireReusablemétodo no puede crear un nuevo Reusableobjeto, espera a Reusableque se devuelva un objeto a la colección.

Código de ejemplo:

using UnityEngine;

using System.Collections;

public class PoolManager : MonoBehaviour {

public bool NewShoot(GameObject shooter, string nameOfType)

{

Transform t = transform.FindChild(nameOfType);

for (int i = 0;i<t.childCount;i++)

{

if(!t.GetChild(i).gameObject.activeSelf)

{

t.GetChild(i).gameObject.SetActive(true);

return true;

}

}

return false;

}

public void NewBullet(string nameOfType)

{

Transform t = transform.FindChild(nameOfType);

GameObject g = Instantiate(t.GetChild(0).gameObject) as GameObject;

g.transform.parent = t;

}

}

Usos conocidos:

Este patrón se utiliza ampliamente en los juegos de las cosas obvias, como las entidades de juegos y efectos visuales, pero también se utiliza para estructuras de datos menos visibles, como la reproducción de sonidos en la actualidad

Patrones relacionados:

* Singleton
* Fly weight

Bibliografía:

# Patrón de diseño de grupo **de objetos [página web]**. Ubicación: ​ https://sourcemaking.com/design\_patterns/object\_pool​.

### Unity3D Scripting: Object Pool [Página web]. Sin fecha. Ubicación: http://www.hagamosvideojuegos.com/2014/03/unity3d-scripting-object-pool.html.

### Object Pool (patron de diseño) [Página web]. Sin fecha. Ubicación: <http://www.hagamosvideojuegos.com/2014/03/unity3d-scripting-object-pool.html>.

### Object Pool [Página web]. Sin fecha. Ubicación: http://progavaumb.blogspot.com/2015/04/object-pool.html.