Object Pool Pattern

Muhammad Zain Raza(19K-0135), Muhammad Ahmed Raza(19K-0134), Ali Rehman(19K-1279)

Definition

The object pool pattern is a creational design pattern that uses a set of initialized objects kept ready to use – a "pool" – rather than allocating and destroying them on demand. A client of the pool will request an object from the pool and perform operations on the returned object. When the client has finished, it returns the object to the pool rather than destroying it; this can be done manually or automatically.

Intent

Object pooling can offer a significant performance boost; it is most effective in situations where the cost of initializing a class instance is high, the rate of instantiation of a class is high, and the number of instantiations in use at any one time is low.

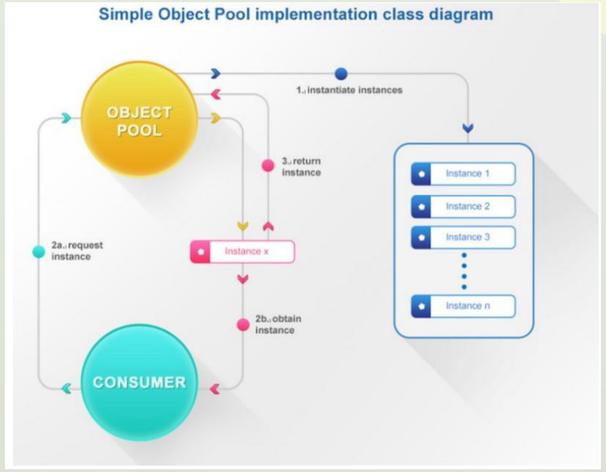
In game development, it comes up quite often. Even fairly simple objects that are visible (such as projectiles, boxes, etc) often have a non-trivial amount of instantiation involved, and characters even more so.

In the not-uncommon situation that you have a huge number of a given object/creature being created over the course of a level, but only a limited amount present at one time (such as waves of enemies in a vertical shooter), then pooling them is generally much better for performance.

Problem

Object pools (otherwise known as resource pools) are used to manage the object caching. A client with access to a Object pool can avoid creating a new Objects by simply asking the pool for one that has already been instantiated instead. Generally the pool will be a growing pool, i.e. the pool itself will create new objects if the pool is empty, or we can have a pool, which restricts the number of objects created. It is desirable to keep all Reusable objects that are not currently in use in the same object pool so that they can be managed by one coherent policy. To achieve this, the Reusable Pool class is designed to be a singleton class

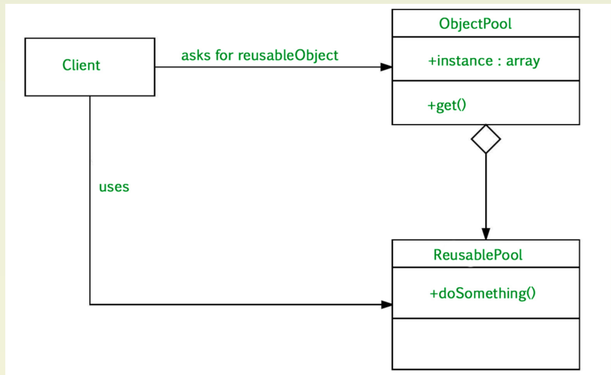
Structure



Consequences

Object Pool involves resetting the state of objects when they are returned to the pool, to avoid unexpected behavior when they are reused. This can be especially important when dealing with resources such as database connections, as failure to reset their state can result in issues with data integrity. It is also important to ensure that concurrent access to objects in the pool does not cause problems, as this can lead to unstable behavior in the program.

UML Diagram



UML Diagram Participants:

Client: This is the class that uses an object of the PooledObject type.

ReuseablePool: The PooledObject class is the type that is expensive or slow to instantiate, or that has limited availability, so is to be held in the object pool.

ObjectPool: The Pool class is the most important class in the object pool design pattern. ObjectPool maintains a list of available objects and a collection of objects that have already been requested from the pool.

Related Patterns

Data Locality. In this pattern you pack objects of the same type together in memory. It will help the CPU cache to be full as the game iterates over those objects, which is what the Data Locality patterns is about.

Prototype: It is a creational design pattern that lets you copy existing objects without making your code dependent on their classes.

Our Implementation

We created a game “Space Shooter” where a spaceship has numerous asteroids coming towards it. The spaceship has to either dodge the asteroids or destroy them using its bullets. When the space ship fires bullets, the bullet objects have to be instantiated at run time which affects the performance of the game. To improve the performance, we used ObjectPool pattern to create a pool of bullets when the game starts. The bullets fired by the spaceship are fetched from the already instantiated pool and returned back to the pool when the bullet is used. Here are some code snippets:

Creating the Bullet Object Pool

```
//Make the Pool
for (int i = 0; i < amountToPool; i++)
{
    //make objects
    GameObject obj = Instantiate(bulletPrefab);
    //disable it
    obj.SetActive(false);
    //add to pool
    poolObjects.Add(obj);
}
```

Using the Bullet Object from the Pool

```
//Linear Search
for (int i = 0; i < poolObjects.Count; i++)
{
    if (!poolObjects[i].activeInHierarchy)
    {
        return poolObjects[i];
    }
}
return null;
```

Tool & Language we used:

Conclusion

The object pool pattern is a technique to improve performance by reducing the cost of creating and destroying objects. It stores a pool of reusable objects and is useful for resources like database connections. Care must be taken to avoid concurrency issues and reset object state.