Object Pooling Concepts

Object Pooling Concepts

What Is Object Pooling?

Object Pooling is a ‘creational’ design pattern that can be used to help optimise your games performance.

It involves ‘pre-instantiating’ a number of game objects (‘the object pool’). This is often done at the start of the game, and then the game objects in the pool can be used throughout the game.

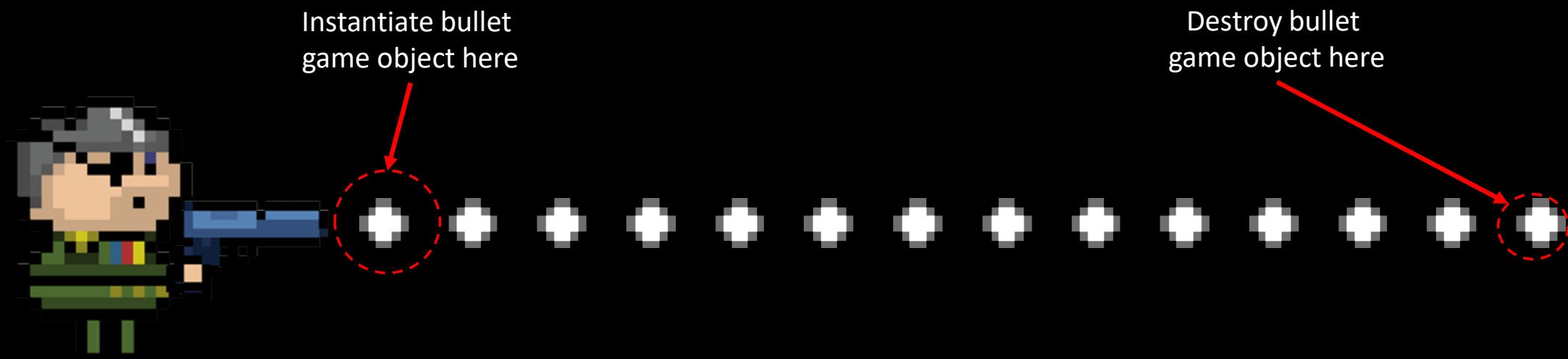
So during the game, rather than creating and destroying game objects when they are needed, the required game object is taken from the object pool and ‘activated’ for use, and then ‘deactivated’ when it’s no longer required.

The game objects in the object pool are continually recycled and reused, rather than creating and destroying new game objects.

This approach has several benefits, especially for games like dungeon shooters, where thousands of short lived game objects like bullets are used in normal gameplay.

Object Pooling Concepts

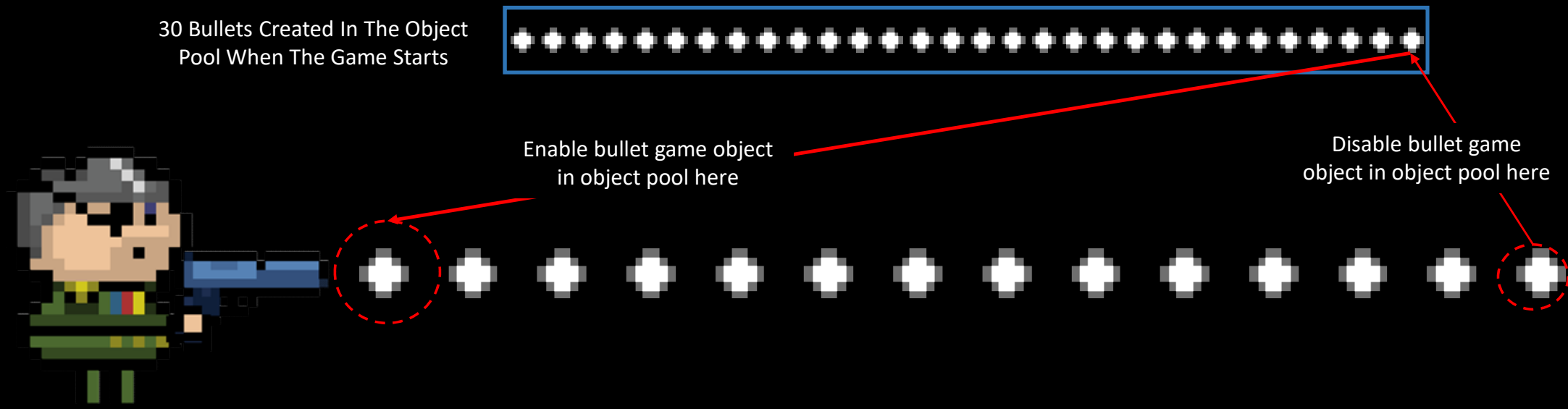
NOT Using An Object Pool - Creating and Destroying Game Objects



In this example we are instantiating bullets as they are fired, and then destroying them when they hit their target or go off screen. In a dungeon shooter this will result in tens of thousands of bullet game objects being created and destroyed.

Object Pooling Concepts

Using An Object Pool - Activating And Deactivating Bullets



In this example we are instantiating bullets in an Object Pool as the game starts. When a bullet needs to be fired a disabled bullet is selected from the Object Pool. It's then enabled and fired, and when it hits the target or goes off screen it's disabled and returned to the Object Pool so that it can be reused.

So rather than tens of thousands of bullet game objects being created and destroyed, only a small number of pooled objects (30 in this example) are created when the game starts and are then reused.

Object Pooling Concepts

So How Can Object Pooling Improve Performance?

Creating and destroying 10,000s of game objects uses more CPU time to create and destroy the objects.

- Enabling and disabling a small number of pre-instantiated game objects in the Object Pool takes less CPU processing time.

Every time a game object is destroyed it's allocated memory is released using a system process called the 'Garbage Collector'. If the Garbage Collector is processing a lot of destroyed objects in a single frame this can result in CPU 'spikes' which can cause frame rates to drop substantially, and the game to 'stutter'.

- With the Object Pool a defined number of game objects are created at the start of the game and memory for them is allocated once.
- Subsequent enabling / disabling of these objects does not trigger any new memory allocations or Garbage Collection memory deallocations.
- This avoids 'spikes' and frame rate drops, since these objects are not deleted until the game finishes.

Some games such as dungeon shooters by their nature involve creating 10,000s of bullets, visual effects, and sound effects, and are susceptible to performance issues due to the large number of game objects that may be required.

- Object Pooling is a good strategy where you have a lot of short lived game objects as part of the game, and where any performance dips will be very noticeable and affect gameplay.

Object Pooling Concepts

What Are The Downsides To Object Pooling?

You have to make sure objects from the Pool are reinitialized if required before using them, else you may have some 'dirty' values left in their member variables or components.

You are reserving memory 'upfront' when you create the pool, which may lead to a larger memory 'footprint' compared to just creating the game objects when you need them.

You usually decide upfront how many of each object type you want in the Object Pool – this may be too many or too few (with additional coding you can increase or decrease the Object Pool during gameplay if required).

Implementing an Object Pool typically requires more coding than just creating the objects when you need them.

Object Pooling Concepts

So Are Object Pools Suitable For Dungeon Gunner?