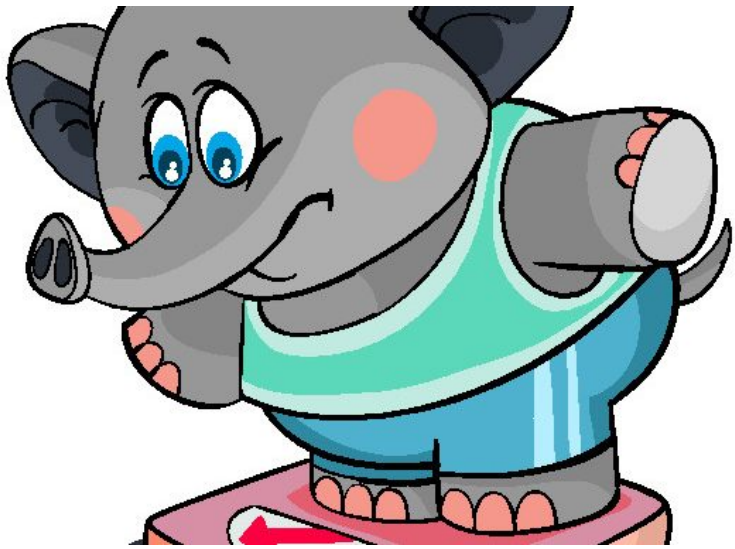
Unity: Memory profiling

July 1, 2018 forwolk Uncategorized 0



For a long time, if you wanted to profile memory footprint of your game in Unity, your only options were the Unity built-in profiler or platform tools (like Xcode telemetry for example). While platform tools were giving you the correct information about application memory consumption, they weren't able to give you any details about what amount of memory is used by different entities of your game. On the other hand Unity's built-in profiler provided you with more detailed information about memory footprint of your games Assets but lacked the precision of the platform tools. To address those issues a couple of years ago Unity released a new experimental memory profiler.

With this new tool, you were able to take a snapshot of the memory used by your app if you compiled it using IL2CPP. It gave you many insights such as what amount of memory, which Texture, Mesh or other Asset used and who was referencing them. New profiler made it easy to track memory leaks.

Unfortunately, to this day new memory profiler is quite experimental. In some cases, it crashes when taking a memory snapshot, in some when analyzing it. During the last Unite, Unity

SEARCH …

RECENT POSTS

Memory alignment fun

Unity: Memory profiling

IL2CPP optimisations

Dictionary: Best practices

RECENT COMMENTS

ARCHIVES

July 2018

June 2018

May 2018

CATEGORIES

Uncategorized

META

Log in

Entries RSS

Comments RSS

WordPress.org

announced that they are going to invest resources in this tool and fortunately we will see a very cool and stable profiler upon **Unity 2018.3** release.

Until that happens we still need to profile our games. Some of us may also want to create CI tests to check for memory consumption of different assets as well as make sure that there are no duplicates assets loaded into memory. For those reasons, I decided to make a post and some small examples on how you can utilize low-level Memory profiling API exposed by Unity.

We start with MemorySnapshot class. This one allows you to capture the snapshot of Unity's heap in the form of PackedMemorySnapshot. Snapshot class houses information about managed and native Unity objects as well as relationships between them. Using this information we can calculate how much memory is being used by our assets, search for duplicates or accident circular references in our code.

Getting the size of the native objects from the snapshot is pretty straightforward. Their size is stored together with the other data in PackedNativeUnityEngineObject. Getting the size of the managed object is a bit trickier.

What is available to you is PackedGCHandle class which stores an address to the managed object's pointer. By reading the pointer, you retrieve the actual location of the object in a heap. At this point, you need to find the type of a managed object you are dealing with. If this is not an *Array* or a *String* you can just use size field from the respective TypeDescription class. If its a *String* you have to parse its size properly. If it's an *Array*, you have to calculate the number of elements in it, taking rank (1, 2, n-dimensional) into account and then multiply resulting *Array* size with the size of the type it houses.

The relationship between objects can also be easily analyzed. To find out who is referencing who, check out Connection class.

I've reused some code from Unity's Memory profiler repository to do a quick implementation of my own analyzer. It prints out the size of native and managed types to the file. It also checks for the duplicate resources loaded into memory. You can check the implementation here.

« **PREVIOUS**
IL2CPP optimisations

NEXT »
Memory alignment fun