DV2549 — Assignment 2
Multithreaded Asset Manager

Deadline	end of LP
Submission	Source code and report (pdf) uploaded to ItsLearning
Grading	U/G

1 Description

Most game engines provide the ability to encapsulate game assets in some form of packages or compressed files. These packages can contain multiple assets and a variety of ready to use Engine content inside.

A package can be a simple concatenation of binary files (compressed and not compressed), with some form of descriptor or header listing its contents, or it can use compression and archiving techniques such as ".cab", "tar.gz", "zip".

The way in which a package file is built and accessed is critical for the performance of the application. Which asset goes in each package and how the memory blocks are read from persistent storage will determine how efficient the game is accessing the persistent storage and memory.

It is also common nowadays to exploit the multiprocessing capabilities on modern hardware, therefore it is highly likely that multiple threads of execution will try and access these packages at the same time to load resources.

Another important aspect, is that at any given time during the game, the Working Set Size has to stay below the size of the physical RAM as unexpected file I/O can affect the game experience if not render the game unplayable.

If the OS supports paging it cannot predict in a clever way which pages to keep resident and which ones to swap out, so the application has to avoid resorting to this facility if it is available. For this assignment, we will assume that if we do not use more than a certain amount of RAM the OS will not swap out any memory page to disk.

2 Task

2.1 Design

You will propose a design for a Resource Manager, with capabilities of accessing assets stored in a package or bundle stored in the filesystem.

The design should support the addition of different archiving and compression techniques although for the assignment you only have to support one technique of your choice. In a similar manner, the design has to allow for the addition of different file formats of assets without making major changes.

Finally, you have to support for the use of *global unique identifiers* (GUI) for each asset within a package, and ensure that no asset is loaded more than

once by the resource manager. This implies that all assets will have some type of GUI and a table or text file will contain the mappings between GUIs and concrete assets in a package. The idea is that at run-time only GUIs are known and used in the engine.

2.2 Implementation

If you need to access compressed files, you can use any existing library of your choice. The solution has to be thread-safe, to allow multiple threads to access concurrently the Resource Manager, and it also has to run on 64 and 32 bit architectures.

The solution will support setting memory usage constraints, to test for different scenarios and target platforms. At any given time, the Resource Manager will not be allowed to use more than a fixed number of bytes. Of course all the scenarios for testing should fit within this hard limit.

It is a design decision whether to treat the over allocations as warnings or errors. During development of a product, it is probably more helpful to have warnings, whilst in production there is a real hard limit which is the device where the code runs.

Moreover, when allocating resources, the Resource Manager should be able to replace those resources unused to fulfill the request. If no resources can be freed to fulfill the request the application should stop or issue a warning that the memory limit has been reached and dump a list of all the resources loaded in memory.

For testing purposes, you need to implement a prototype or reuse any existing project that makes use of common assets such as 3D models, textures and audio files.

For this assignment the mapping table from GUIs to concrete assets within packages can be created by hand, in a real scenario this would be a task for a pipeline conditioning tool which can process the assets tree and create a bundle and all the GUIs.

For this testbench, you have to define a scenario to stress a particular situation within a Game such as short lifespan objects (and its associated asssets), background level content streaming, multiresolution techniques, and so on. You only have to work on one scenario of your choice, not limited to the ones listed above.

3 Submission and grading

The assignment will be done in groups (ideally the same groups created for assignment 1), and has to be submitted. You have to write a report, in which you should provide a clear explanation of the solution, enumerate each assumption made about the problem and mention the different decisions that were made in order to obtain the final design.

Along with these comments, a UML design has to be included in the PDF document and a brief description of the classes as well.

The grading for this assignment is U or G. To reach the G grading the work has to fulfill the requirements explained in section 2. The workload between the participants of the group should be distributed in an even way, and reported in the final document.

References

[1] Gregory, J. "Game Engine Architecture", Chapter 6. Resources and the File System.

[2] ZZipLib, ZLib compression library.

[3] Windows File Buffering