DV2549 — Assignment 1
Memory allocation strategies

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

1 Description

Memory allocation is a primary task in any piece of software, and Game Engines (GE) are not an exception. The rate at which game objects of any kind have to be created in a GE will differ among different genres, nevertheless in many situations a game will face the need to create large amount of dynamic objects for the gameplay.

As we know, memory allocation offered by the OS services (if there is any) is a costly operation that has to be used wisely when developing real-time interactive applications. The two main factors that will affect the performance are the time spent during the system calls (switching context to the OS) and the memory fragmentation due to the OS using a general "one-size-fits-all" strategy for all memory requirements.

In this assignment, you will implement a custom memory allocator module which should allow for pool and stack based allocation strategies [1]. To be able to have a baseline for comparisons, the proposed design and implementation should also allow switching between your custom memory management and the regular allocation service provided by the OS (Linux, Windows or OSX).

2 Detailed description

2.1 Implementation

For this assignment, each group can choose a different scenario, that combines their particular needs with the usage of pool and stack based allocators. It can be a simple and small application that does not render anything, or it can be an existing 3D project that can be modified for the assignment.

The memory manager has to support multiple threads doing memory requests to it. You can also assume that more than one memory manager can exist in runtime, to support for different types of requirements.

We will focus on two different scenarios. For the first one, your program has to create a large amount of small objects that can be released in a different order than the one used for creation. Their lifespan can surpass the time it usually takes to process a single frame in a game, and they have to be of the same or similar size, to avoid internal fragmentation of the memory.

The second usage scenario is a frame based allocation strategy, in which many objects are created with the only purpose of supporting all the tasks during a single frame. After every frame, these objects are not useful anymore and should be removed. The motivation behind this use case is that it is not easy to be sure about the stack size for every possible platform, so we want to store all local variables using our own stack within this memory manager.

In order to obtain useful results (and performance gain measurements), you might need to vary the number of requests and also the size of the requests in both scenarios.

3 Submission and grading

The assignment will be done in groups of 4-5 students, and has to be submitted in electronic form (PDF and source code). In your report, you should provide at least the following:

- A description of the problem and the solution proposed.
- Any considerations and assumptions not mentioned here in the assignment.
- A UML design for the memory manager and other components that make use of it.
- Graphs comparing the execution time for the different scenarios chosen.
- A conclusion section explaining the results obtained and analysing in a broader sense the use of custom allocators for memory management in game projects. For example explaining which situations in the context of a game would be of most benefit from using such techniques.

If you submit a Visual Studio project, please clean the project first to lower the size of the directory.

The grading for this assignment is U or G. To reach the G grading the work has to fulfill all the requirements explained above.

References

[1] Gregory, J. “Game Engine Architecture”, 5.2. Memory Management.

[2] C++ FAQ. http://www.parashift.com/c++-faq-lite/memory-pools.html