NOX ECSX 2: Test Case Plan

February 6, 2017

Contents

1 Purpose

The test case plan is meant to outline in a rough manner how the test cases are supposed to accomplish, how they are performed and how we measure the performance of said tests. In general in these test cases we want to try as much as possible to make the performance measurements represent the specific case we are testing. This means that we want to avoid using components that deals with code that we do not have full control over, like the Box2D and OpenGL code. For each test want to use both different compilers and profilers to make sure the readings are accurate.

2 Testing tools

Insert text about importance of different use of compilers and profilers

2.1 Compilers

Different compilers to use? Also use different versions of each compiler?

- Visual C++, visual studio compiler
- Clang
- GCC

2.2 Profilers

Different profilers to use? PMS knows much more here, let him do it

- VSTS Profiler, visual studio profiler

3 Test cases

3.1 Compilation time

The compilation time should be kept as low as possible when building the engine.

How the test is performed

There are a few ways to measure this.

- *Complete compilation*
  There are occurrences where you would recompile your entire project, but they are rare. However, they happen often enough that it is worth testing, but is not the test we value the most.

- *Partial compilation*
  A bit harder to test, but the most frequent type of compilation is when you use the engine for developing your own game. This results in small parts of the engine being recompiled after each change. The most common occurrence of this would be when adding additional components to the engine.

How the test is measured

Tested by timing the time it takes to compile the engine. Should also be measured using different compilers to ensure the results are consistent.

3.2 Large number of different components

This case will test the possibility of having many differently functioning components at once in the engine.

How the test is performed

The test is performed by first creating 200 or more dummy components that does some simple task. We then place out several actors with the components in an otherwise empty world. We need to create a script which makes all off these components, as it would be a waste of time for us to so by hand.

How the test is measured

The test case is successful if the possibility of having that many components run properly at the same time over multiple actors at the same or greater performance of the NOX-Engine.

3.3 Memory usage

Compare the usage of ram between the NOX-Engine and our system. We want to avoid increasing the amount of memory used. This is important because of the development towards the mobile platforms, which is lacking in the memory aspect compared to the PC platform.

How the test is performed

The test is performed by creating a set amount of actors, and then measuring how much memory is currently in use when the spawning is complete. We want to test a few different cases.

- *Empty actors*
  Check for actors without any components to compare the overhead of the actors alone.

- *Actors with empty components*
  Actors with empty components to check the overhead of the components.
- *Deep scenegraph*
  Place actors with children multiple levels deep in the scenegraph to display how much overhead the graph provides in very deep graphs.

It is also important to note that an increase in memory usage could be justified by an increase in other aspects off performance, for example faster update loops.

How the test is measured

The measurement is checked by how much memory is in use when certain amounts of actors are created.

3.4 Multi-threading support

Show that the use of multi-threading in our implementation are actually giving a significant benefit in appropriate scenarios.

How the test is performed

There setup for this is to create dummy components with waits inside their update functions, giving a consistent, but not a realistic scenario where multi-threading would improve performance. Depending on how the multi threaded system works, we could do a lot of other specific tests to show the potential of the system.

For example the opt in idea for the multi threading could show off how effective it can be when used properly, and how ineffective it is when ignored.

How the test is measured

The test case is successful if there is a significant boost in performance. There will be comparisons between the NOX-Engines single threaded architecture, ECS-NOX 2 run with a single thread, and ECS-NOX 2 with multiple threads.

3.5 Thread safe logging system

Show that the logging system has the possibility to function in a multi-threaded environment without any issues.

How the test is performed

The test is conducted by creating a few components that run on different threads, and let them spam the logging system over a set period.

How the test is measured

Successful if there aren't any crashes or errors over a substantial amount of time. This does not guarantee that it could not crash, but is a good indicator that everything is working at an acceptable level.

3.6 Fast spawning

The creation of new actors and their components should be done as fast as possible. This is an issue in the NOX-Engine, where creating a lot of new actors or components at the same time causes frame rate issues.

How the test is performed

Make it so that a significant enough number of actors are spawned so that it has an impact on the frame rate. There will be several different versions of this case.

- *Empty actors*
- *Actors with empty components*
- *Actors with components*

How the test is measured

We then measure how long it takes for all of the actors to be spawned, along with various other metrics from several profilers.