Task 25 Spike: Measuring Performance & Optimisations

CORE

Context

Before any optimisations can be undertaken, the performance characteristics of the application must be understood. Good optimisation processes require solid measurement techniques to collect data that can then be accurately compared and analysed, and used to inform code changes. Once measurements have been completed, informed optimisations can be made.

Knowledge/Skill Gap:

Developers need to be able to collect and analyse software performance data. Once measurements have been made, the developer can identify and execute improvements to code.

Goals

Use the code provided on the unit website for this spike. The code uses SDL, and creates a number of boxes with random size, position and velocity. The code is designed to run a number of different "tests", each using a standard game loop that updates (moves) each box, checks for collisions, and draws all the rectangles to the screen. Each test uses a different collision detection approach, and is timed.

Measure the performance of the various collision-checking methods provided. You can use the timing code provided or any other profiling tool you choose to collect meaningful data, and then create a report that presents a comparison of the different collision detection approaches.

Your results must be collected without the overhead of rendering time, and for a sufficient duration to create stable results.

Your report must include:

1. A description of each collision test (method) approach (and the differences between them).
    a. The method you used to collect your data and your reasons for choosing it
    b. The raw data you collect to support your results.
    c. A summary table of the results

Using the results of your measurements, modify the provided code to use the most collision detection method found to be the most optimal. When this is done, add your own modifications to further improve performance. Using your performance measurement method, demonstrate that your modification result in performance improvements (however minor they might be) over the most optimal path found.

Include the results of your performance testing and the rationale for your modifications in your Spike Report.

Expected Output

Repository

1. Code
2. Spike Report

Canvas

1. Spike Report

Notes

- Download, compile and run the code provided on the unit website. You will need to create a new project that has the appropriate references/paths set to your SDL libraries.

- Read through the code carefully and understand the structure. The code uses function pointers to make running different tests easy. You may need to read up on this first.
- Identify the different collision functions that are available in the code.
- There is an explicit delay in the game loop of the default code. You should remove this so that the code runs as fast as possible.
- When collecting collision data, you also want to remove the overhead of rendering. (There is a
- simple Boolean flag to do this.)
- Make sure you run the tests for a more than just a few seconds, and repeat the tests in order to get a good sample size for your results.
- Run the tests, collect the data, analysis and present in your report. A pretty chart is highly recommended. Don't forget to include the summary table.
- The timing code provided is the simplest way to get results, but using a more sophisticated profiling tool is recommended, both to improve your results and to improve your familiarity with professional tools
- You do not have to achieve enormous performance improvements (although there certainly are substantial gains to be made), you just have to achieve a measurable improvement.
- If the improvements you make do not result in consistently better results, consider why this might be and note the reason down in your spike report. (Hint: compiler settings?)