Task 24 Spike: Performance Measurement

CORE

Context

Performance can be a very important consideration for games, as many games try to create an immersive experience and push the boundaries of technology to calculate and present details. When developers write code, they should be aware of performance issues related to the structure of their code. There are often trade-offs between different considerations.

Knowledge/Skill Gap:

Developers need to be able to collect and analyse software performance data. Sometime this can be done very simply, and it is often a quick way to test and decide on a code design choice. Once measurements have been made, the developer can identify performance issues and make improvements, where needed, supported by data.

Goals

Your overall goal is to create working code, and a simple spike report, that you could share with a colleague so that they would know how to measure code performance directly (in your code, not with IDE or other tools at this stage).

To help you do this, and to get used to using the spike problem-solving approach, we have provided sample code for this task.

Usage:

- The sample Visual Studio solution contains two projects:
  (If you're not using Visual Studio, the cpp files are very simple and should be easily built using your favourite IDE or from the command line)
  - generate_data generates a number of JSON objects resembling social media posts.
  - performance_test counts strings in the generated data that match a hardcoded list

Demonstrate the following performance and measurement concepts (by running the performance_test code):

1. **Measurement**: Show, with numbers and **a chart**, the performance of each provided function.
2. **Scalability**: Show, with numbers and **a chart**, how the performance of each function varies with different data sizes (Use generate_data to create json files of different sizes – you really want to go with a few different orders of magnitude).
3. **Repeatability**: Show, with numbers and **a chart**, how ***repeatability*** will vary with multiple tests in a given run.
4. **Compiler Settings:** Modify your compiler optimisation settings and demonstrate, with numbers and **a chart**, a difference. (Make a note of how to do this so that a team member would be able to reproduce your result.)
5. **Profiling**: Use tracy (https://github.com/wolfpld/tracy) or the Visual Studio performance profiler (Instrumentation mode) to profile the code. Provide a brief description of the results, including screenshots.
   a. If you have another profiler you prefer to use, you may – it just needs to have the same functionality as the suggested ones.

Expected Output

Repository

1. Code
2. Spike Report

Canvas

1. Spike Report

Notes

Note: You must create visual charts for some of these points. We recommend you do this in MS Excel or similar. For this simple task, add the chart images to your spike report (not a separate report document). If you create an accompanying .xlsx file, make sure that is added to your repository.

- Assume you are writing steps to help a colleague to recreate your results. Make sure your notes make enough sense!
- As a review, show your notes to another student. Afterwards, update with the suggestions or omissions so that your notes are valuable. Your repo commit comment would be something like “spike report improvements based on feedback from …” or similar and include who you showed your notes to.
- Learn about std::chrono!