Simple Performance Test

Current version: 1.1

Important links

- Simple Performance Test GameMaker Marketplace page: https://marketplace.yoyogames.com/assets/449/simple-performance-test
- Current version of this document: https://docs.google.com/document/d/1UArkOkVrcddrTX8uHoAvU3ZWhTOa9tcAYzUF6l3e0VQ/edit?usp=sharing
- author twitter: @csanyk
- author's main site: https://csanyk.com

License

Simple Performance Test is subject to the GameMaker Marketplace EULA

- https://marketplace.yoyogames.com/eula

Additionally, as the author and copyright holder of the work, I grant the following rights to all users, beyond those granted in the EULA:

1. You have the right to share the source code of this project.
2. I *encourage* you to share your test projects that you build with this project, for purposes of peer review.

Simple Performance Test Overview

Purpose and Philosophy

Simple Performance Test is a project stub that I created in order to make it super easy to set up a comparison test between two snippets of GML code.

It's very true that performance is not the only measure of code quality. Ease of understanding, maintanability, and verifiability of correctness are all also very important. Often you'll hear seasoned programmers say that performance optimization is not critical in code, and that premature optimization is the bane of inexperienced coders. This is true.

But when performance is crucial, it is good to have some way of testing one way of doing something against another, to see whether there is an advantage, and how much of an advantage there is.

The benefit of Simple Performance Test is to settle arguments about performance with a verifiable tool that can yield measurable, repeatable results.

How it works

The test runs each snippet 10,000,000 times, and records the time it took to run each snippet in microseconds (1,000,000-ths of a second). Then the results of the test are displayed on the screen.

After each loop has run its course, the timings of each are displayed, showing which is the faster (lower run time = better).

By running the test code blocks many millions of times, minor fluctuations and anomalies can be averaged out, and minor differences in execution time can be magnified.

Understanding the outcomes and limitations of the SPT test

Accuracy, precision, margin of error

It's been long time since I've been in a science classroom, so if anyone reading this would like to offer a better or more concise explanation, I would welcome your comments.

At first, it may seem that this test is not exceedingly precise. If you run the tests repeatedly, they do not give the exact same numbers each time.

In order to test how precise the test might be, I thought to try running the exact same code in Test1 and Test2. My theory was, since the code is exactly the same, it should take exactly the same amount of time to execute, so the result should be identical.

However, to my surprise I learned that this was not the case! If you run the tests empty, simply executing the empty `repeat{}` loop for 10,000,000 iterations two times, and compare the results, you'll see the execution time for each test will NOT be identical, nor will they be exactly the same each time. In fact, the total execution times for Test1 and Test2 can vary by as much as a tenth of a second. This seems wildly inaccurate, then.

However, if we remember that the test loops run 10,000,000 iterations, and what we're really concerned with is the time the code takes to run one iteration, we can obtain the average execution time for a single iteration by taking the total run time and dividing it by 10,000,000.

I currently believe that most of the difference between one run of the same test code and another is attributable more to factors outside of GameMaker itself -- the operating system that the project is running on may be busy running other tasks in the background, and thus the amount of available CPU time can vary from the execution of Test1 and Test2 enough to cause them to take a different amount of time to run each time, even though the code running within the loop may be exactly the same.

Accordingly, in theory it may be possible to improve the accuracy of the test results by running the test project on a stripped-down Windows configuration with as many unessential processes and services shut off as possible, and/or by assigning the test project its own CPU core to run on by itself. But specifics on how to do such things are beyond the scope of tihs documentation.

But the two loops, running identical code, *will* be close to each other, to a degree of accuracy. In a fashion, you may "calibrate" your test by running identical code in Test1 and Test2 several times, and note the range of variance in the average iteration time. To the extent that the numbers agree for Test1 and Test2, we can consider the test to be accurate to within that degree.

For example, on my hardware, one run of the empty loop test, using the Windows build (non-YYC) yielded these results:

```
Test1: 1433763 microseconds (avg 0.14377630 microseconds per iteration)
Test2: 1449522 microseconds (avg 0.14495220 microseconds per iteration)
```

In total, Test1 and Test2 are off by 15,759 microseconds, or 0.015759 seconds. Despite being off by over 1/100th of a second, the average times per iteration agree to a precision of 0.01 microseconds, or 1/100,000,000th of a second. Therefore, we can ignore anything in the average per iteration times after third decimal as "margin of error".)

The margin of error will not be the same for any test setup. Depending on the hardware, the build target, and the actual code being tested, it may vary. For example, running the same empty test with the Windows (YYC) build target, yields reuslts of anywhere from 1-3 microseconds.

```
Test1: 2 microseconds (avg 0.00000020 microseconds per iteration)
Test2: 3 microseconds (avg 0.00000030 microseconds per iteration)
```

```
In this test, the accuracy is to within 0.0000001 microsecond, which is very
precise indeed.
```

Over 10,000,000 iterations of an empty `repeat{}` loop, there will be considerable variation, as much as a .1 second difference in execution time per test run. However, if you calculate the *average* times for the test loops to complete one iteration, we get a result which is precise enough to be useful.

If we run the test project several times, the numbers will vary. But once you have some idea of the margin of error involved with a particular test, you'll know how much of a difference the test is capable of detecting.

Code added to the `repeat{}` loop can introduce even more variability in the execution times, so I can't tell you what your *actual* margin of error might be for any particular test. But you can get a feel for it by running your tests multiple times and seeing how the results vary, and look at the range of variation.

The main thing to keep in mind is, whatever the margin of error is, comparing results *within this margin of error* is not meaningful.

But if there is a *dramatic* difference in execution time between Test1 and Test2, say a 2x or 10x difference between the execution time of Test1 and Test2, SPT can show that. If the difference in performance is less than that, it's almost certainly negligible in real-world application of the code.

Testing is only valid if it is repeatable

The important thing is that the test is *repeatable*, so if you re-run it several times, with consistent results, that will tend to reinforce the findings of the test. If someone doubts your findings, you can share the project with them and let them audit the code, and run it to see for themselves. Perhaps they will find a flaw in the way you wrote the tests, or perhaps they will find an even better way to write the code to be even faster. Perhaps most importantly, being able to re-run the tests again and again can be of great benefit because as GameMaker changes over time, findings that were once true may no longer hold, and commonly believed knowledge about tricks to improve performance may not be true any more.

You can also compile the test and run it on several different devices, different platforms, etc. to get a reasonable idea of how performance differs on different hardware or different build targets. Over time, you can re-run the same test project when new versions of GameMaker come out, to see if any changes in performance have been provided by updates to the run time.

Why 10,000,000 test loop iterations?

10,000,000 is somewhat arbitrary. I picked it because it's a very large number, which should help smooth out variations in execution time for a single iteration. For a small code snippet, 10,000,000 iterations only takes just a few seconds, so you don't have to wait too long to get results. So it feels like a useful number.

But I did not choose this number carefully; I just pulled it out of the air. It may be that a smaller number could give reasonably accurate results even more quickly, or a larger number could give better results accuracy at the expense of longer run time to obtain them. If you want to, you can experiment and see whether a different number might be better.

Feedback

Scientific testing is all about peer review. I really, really encourage you to share your tests and their results with the GameMaker developer community. By doing so, we will educate each other as well as correct each other's mistakes.

In that spirit, I give this SPT test framework to the community for free. You may share your test projects that you build from it with anyone. But as for the test framework itself, please direct people who want it to the official Marketplace page. That way they will be able to obtain the latest version, and I will also be able to know how many users have downloaded it, and have some idea how popular it is. SPT will **always** be free.

If you want to *share* a project that you've built with SPT, whether as a Marketplace product, or on your own web site, or a download service like dropbox, I grant the right to do so.

Improving SPT

If you know of any ways to improve the accuracy or precision of the test method, I'm interested in hearing from you. Send me your feedback through the marketplace page.

Using Simple Performance Test

Set up a test

1. Open up the Simple Performance Test project.
2. Edit the object oSPT:
    a. In the Create Event:
        i. Give the test project a title by providing a string value to test_project_title.
        ii. Name the two test cases by providing a string value to the variables `Test1Name` and `Test2Name`
        iii. Set the number of trials the test should run by modifying the value of the variable `test_iterations`
        iv. If you plan to publish the source code of the project, and I *strongly* recommend that you do, provide a url in the `test_source_url` variable for where the test project file can be downloaded.
            1. Be sure to upload your project files to this location! Once the project has been fully set up, export the project as a .gmz, and then upload to the location you plan to share the project from.
    b. In the Step Event
        i. Add the GML code that you wish to test in the repeat loops for Test1 and Test2.
        ii. Be sure to use only the code that you want to factor into the test in these loops. If you need to do any set up, such as declaring variables or performing calculations needed prior to the code that you actually want to test, do those outside of the `repeat{}` loops, and outside of the lines of code that mark `test_start_time` and `test_end_time`.

Run the test

1. Compile and run the project.
2. Press the **Enter** key to **run** the tests.
3. Wait for the tests to run. This can take several seconds or even longer, depending on the code being tested. The game will appear to be frozen because the two test loops are executing 10,000,000 times in a single Step of the game's main loop. At a minimum this will take several seconds for GameMaker to process that step.
4. When the test loops have completed running, the results will be displayed on the screen.
5. To run the tests again, press **Enter**.
6. Press **R** to **restart** the program. The calls the game_restart() method, so should reset the program to its starting state. You can re-run the test again, by pressing Enter, or exit the program.
7. Press **Esc** to **exit** the program.

Release Notes/Road Map

Version	Notes
1.0	Initial release ● Simple Performance Test object
1.0.1	● documentation errata
1.1	● documentation errata ● clickable links in Instructions credits