FPS Messungen im GameMaker

Description

Qualitätskontrolle ist gerade bei Computerspielen sehr wichtig. Es gibt kaum etwas nervigeres, als Spiele, die abstürzen oder aufgrund von Fehlern nicht spielbar oder schaffbar sind. Doch auch die Geschwindigkeit spielt eine wichtige Rolle.

Die FPS, also Frames per second bzw. zu Deutsch Bilder pro Sekunde sind ein wichtiges Maß dafür, ob ein Spiel flüssig läuft, oder nicht. Gerade bei PCs gibt es da große Unterschiede. Die Messung der Geschwindigkeit wird dabei im Hobby-Bereich gerne weggelassen bzw. nur auf ein oder zwei Systemen durchgeführt. Das Motto lautet oft:

> Wenn es auf meinem alten Laptop gut läuft, läuft es überall.

Für viele kleine Projekte reicht das tatsächlich aus. Aber sobald man mehrere Versionen für diverse Systeme hat, kritische Features einbaut oder einfach nur optimieren will, sind konsequente FPS-Messungen unerlässlich.

fps_real vs fps

Der GameMaker bietet dazu zwei schreibgeschützte Variablen an: *fps* und *fps_real*. Der Unterschied ist, vereinfacht gesagt, dass *fps* die Frames innerhalb des Gamespeeds angibt, *fps_real* gibt die tatsächliche FPS an, die meist weit darüber liegt.

Beispiel: Unser Gamespeed liegt bei 60 FPS. Dann wird *fps* nie einen Wert über 60 anzeigen. *fps_real* hingegen kann aber auch 2000 anzeigen.

Für eine ordentliche Qualitätssicherung ist also *fps_real* entscheidend.

Richtig messen

Im einfachsten Fall geben wir im Draw- oder Draw-GUI-Event folgendes ein:

```
draw_text(16, 16, string(fps_real));
```

Dann wird uns immer die aktuelle, reale FPS angezeigt. Doch damit fangen wir eigentlich nicht so viel an. Die Zahl ändert sich bei jedem Step, also 60mal pro Sekunde. Außerdem wäre es schön, wenn wir am Ende mindestens einen Durchschnittswert hätten.

Wir gehen also wie folgt vor: Es wird eine Variable mit dem Namen *_fps* erstellt. Das ist der Wert, den wir später am Bildschirm ausgeben. Da wir die Werte speichern wollen, kreieren wir noch das Array *allFPS*. Am Ende geben wir drei Werte aus: Den Durchschnitt, den kleinsten und den größten Wert.

Diese Werte kann man, wenn man will, auch in eine Datei speichern. Das ist hilfreich, wenn mehrere Personen testen sollen. Für eine Vergleichbarkeit wäre eine Benchmark ganz gut, man könnte es auch Teststrecke nennen. Das bedeutet, dass man in der Praxis einen immer gleichen Ablauf hat, der über eine bestimmte Zeit läuft und am Ende die Werte ausgegeben werden. Wir machen das ganz simpel mit einer Textbox.

FPS-Praxis

Meistens nutzt man hierfür ein eigenes Objekt. In meinen Spielen habe ich ein Startobjekt, welches ich meist **obj_init** oder **obj_gameInit** nenne. Da werden alle Konstanten, globale Variablen und Einstellungen geladen, gesetzt und ggf. die „Umgebung” geprüft (Windows oder Browser etc.). Dieses Objekt ist persistent und beinhaltet auch Debug-Funktionen, wie etwa die Ausgabe von *fps_real*-Werten. Darauf wollen wir uns jetzt beschränken. Wir werden messen, Messwerte im Array speichern und am Ende die drei Werte ausgeben.

Event Create

```
_fps = fps_real;
allFPS = [];
fps_average = 0;
timer = 0;
alarm[0] = room_speed;
```

Wir initialisieren die drei bereits beschriebenen Variablen. Dann starten wir *alarm[0]*. Hier werden die Messungen vorgenommen und am Ende einer bestimmten Zeit ausgegeben. Doch zunächst zeigen wir den *_fps-Wert* auf dem Bildschirm an.

Event Draw GUI

```
draw_set_color(c_white);
draw_text(16, 16, string(_fps));
```

Nun geht es an die Datenerfassung und Auswertung.

Event Alarm[0]

```
_fps = round(fps_real);

array_push(allFPS, _fps);

if (timer == 5)
{
 for (var i=0; i<array_length(allFPS); i++)
 {
  fps_average += allFPS[i];
 }

 fps_average = round(fps_average/array_length(allFPS));

 array_sort(allFPS, function(elm1, elm2)
 {
        return elm1 - elm2;
 });

 var low = allFPS[0];
 var high = allFPS[array_length(allFPS)-1];

 show_message("Durchschnitt: „ + string(fps_average) + „\nniedrigster Wert: „
 game_end();
}


alarm[0] = room_speed/10;
```

In Zeile 1 übergeben wir den gerundeten *fps_real*-Wert an *_fps*. Gerundet, weil Kommawerte bei FPS keinen wirklichen Sinn ergeben.

Mit *array_push* fügen wir den neuen Wert hinzu. Alternativ würde auch *array_insert(allFPS, array_length(allFPS), _fps);* gehen, aber *array_push* ist der bessere Weg.

if (timer == 5) ist ein Platzhalter für das Ende der Messung. Man könnte einen Counter laufen lassen, die Position eines Objekts abfragen, was auch immer.

Wenn die Messung beendet ist, starten wir eine for-Schleife. Hier zählen wir alle Werte zusammen und speichern diese in *fps_average* ab. Danach wird der Durchschnitt berechnet (Summe durch Anzahl der Werte).

Sobald das erledigt ist, sortieren wir das Array aufsteigend. Weitere Informationen dazu gibt es hier.

Der Kleinste Wert ist danach auf Platz 1 (bzw. 0) und der höchste Wert ist am Ende. Diese speichern wir in *low* und *high*. Wichtig: Der letzte Wert ist *array_length(allFPS)-1*, da die Zählung bei 0 beginnt.

Das Ganze geben wir per *show_message()* aus. Hier könnten wir auch eine Funktion aufrufen, die alles in eine Datei speichert. Am Ende von Alarm[0] rufen wir den Alarm wieder auf. Das machen wir

10 Mal pro Sekunde.

Fazit

Die Frage ist erlaubt, warum man nicht gleich die Debug-Funktionen von GM nutzt. Bei der FPS-Messung hat das einen guten Grund: Debug ist fast immer langsamer als normale Kompilierung. Außerdem kann man so bspw. verschiedene Compiler testen. So konnte ich feststellen, dass ein Effekt für ein zukünftiges Tutorial per YYC rund viermal so schnell läuft als per VC.

In HTML5 macht übrigens *fps_real* zu *fps* keinen Unterschied. Da zeigt er immer Gamespeed an.

Das gezeigte Beispiel ist natürlich sehr rudimentär und lässt sich beliebig erweitern, etwa durch eine Anzeige einer FPS-Kurve. Wer sich das sparen will, gibt die Werte als CSV-Datei aus, ließt sich in einer Tabellenkalkulation ein und erstellt sich hier die Grafiken. Neben dem FPS-Wert lassen sich noch weitere Daten speichern. Etwa bestimmte Positionen, die Zeit, Raum usw. So kann man Einbrüche von Werten besser nachvollziehen.

Date Created
2. Juni 2022
Author
sven