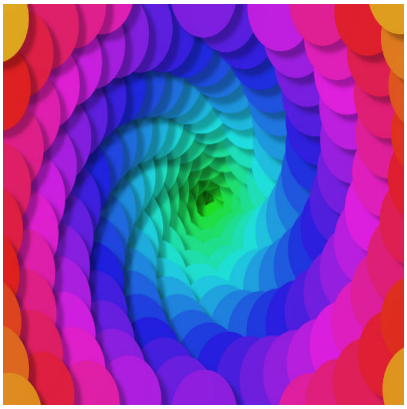
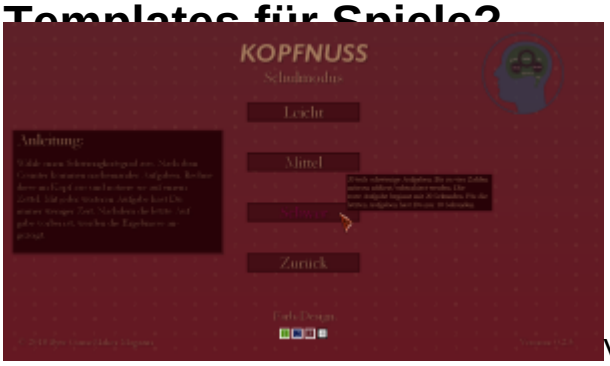


Farb-Themes für eigene Spiele

Description

In den meisten Fällen ist das Farbdesigns eines Spiels ziemlich früh in Code gemeißelt. Spätere Änderungen oder gar verschiedene Designs, die man dem Spieler anbietet, sind nur noch schwer umsetzbar. Das muss aber nicht so sein.

Templates für Spiele?



Vor allem bei Webseiten ist es üblich, dass man mit Templates arbeitet. Design und Inhalt sind strikt voneinander getrennt und so kann das Aussehen der Seite jederzeit angepasst oder völlig geändert werden. In Spielen kennt man das eher weniger, dabei gibt es einige gute Gründe, die dafür sprechen.

Die Trennung von Design und dem eigentlichen Gamecode ist bei Spielen selbst dann sinnvoll, wenn man nicht mehrere Designs anbietet, ähnlich wie bei der Trennung von Sprache und Code. Wer im Nachhinein Schriften, Farben und andere Dinge ändern will, kann dies im schlimmsten Fall an tausend

Es gibt aber auch andere Gründe, dies zu tun. Als „Dienst

am Spieler«, sozusagen. Bei einem Spiel für Kinder, die einen männlichen oder einen weiblichen Charakter wählen können, wären zwei verschiedene Farbdesigns durchaus sinnvoll. Oder wenn man mehrere lokale Profile erstellen möchte und jedes Profil seine eigenen Farben wählen kann. Theoretisch könnte man soweit gehen und den Spieler selbst die wichtigsten Farben wählen lassen. Das hier gezeigte Prinzip macht dies ebenfalls möglich und wäre lediglich eine Erweiterung.

Manchmal sind praktische Gründe entscheidend. Bei meinem aktuellen kleinen Projekt Kopfnuss stellte sich mir folgende Frage: Wie verhält sich die Darstellung bei unterschiedlichen Projektoren und Lichtverhältnissen in Klassenzimmern? Also begann ich, alle diesbezüglich relevanten Daten auszulagern, so das der Spieler (oder Lehrer) per Tastendruck umschalten kann.

Das Prinzip

Die Idee ist ähnlich simpel wie bei den Sprachdateien: Wir lagern alles in Variablen aus und lesen sie ein. In meinem Spiel gibt es für jedes Theme ein Skript mit den jeweils gleichen globalen Variablen. Wird das Theme umgeschaltet, werden lediglich die Variablen überschrieben und die Farben zur Laufzeit ausgetauscht.

Das bedeutet: Ich habe nur ein paar (in meinem Fall 19) Variablen, die ich für jedes Theme definiere. Im Spielcode werden diese Variablen benutzt. Sollte sich eine neue ergeben, kann man das jederzeit ergänzen.

Konkretes Beispiel

Das grüne Theme sieht so aus:

```
// Theme green
global.activeTheme  = 1;

global.bgCol1   = make_color_rgb(79, 143, 19);
global.bgCol2   = make_color_rgb(127, 219, 40);
global.shadowCol  = c_black;
global.shadowDistance  = 4;
global.shadowAlpha  = 0.25;
global.textCol   = make_color_rgb(2, 47, 25);
global.textColMO  = make_color_rgb(164, 255, 79); // Mouse Over
global.barCol   = make_color_rgb(0, 19, 10);

global.procentCol[0] = make_color_rgb(220, 255, 40);
global.procentCol[1] = make_color_rgb(210, 235, 40);
global.procentCol[2] = make_color_rgb(200, 215, 40);
global.procentCol[3] = make_color_rgb(190, 195, 40);
global.procentCol[4] = make_color_rgb(180, 175, 40);
global.procentCol[5] = make_color_rgb(170, 155, 40);

global.counterCircle = make_color_rgb(38, 70, 9);
global.infoboxCol  = make_color_rgb(163, 231, 99);
global.infoboxBorder = make_color_rgb(79, 143, 19);
```

Wie man sieht, sind hier vorwiegend Farben als RGB-Werte definiert, aber nicht ausschließlich.

global.activeTheme sagt aus, welche Nummer gerade aktiv ist. Die Zahl wird vom Spiel in der INI gespeichert, damit der „Service“ komplett ist. Wäre ja schade, wenn der Spieler die Einstellung jedes Mal vornehmen müsste. Dazu kommen noch Variablen wie *global.shadowDistance* und *global.shadowAlpha*, die den Schatten regeln. Man sieht, dass man das Prinzip beliebig aufblähen kann. Man könnte verschiedenen Schriftarten definieren (was ich nicht getan habe), Abstände und vieles mehr.

Im Spiel kann das dann so eingesetzt werden:

```
draw_rectangle_color(0, 0, room_width, room_height, global.bgCol1, global.bgCo
```

Hier finden wir die zwei Variablen *global.bgCol1* und *global.bgCol2* wieder. Das Ganze ergibt einen Farbverlauf im Hintergrund. Wenn das nur an einer Stelle erscheint, mag der Aufwand nicht als besonders sinnvoll erscheinen, aber selbst in kleinen Spielen hat man am Ende der Produktion zahlreiche Stellen in denn wir Farben definieren. Wie eingangs erwähnt, kann ein zentraler Ort, an dem die Farbwerte als Variablen liegen, sehr nützlich sein.

globale Variablen

Globale Variablen werden in der Programmierung oft sehr kritisch gesehen. Das hat vorwiegend zwei Gründe. Sie können Probleme verursachen, falls man sie versehentlich zweimal definiert und unterschiedlich benutzt. Der zweite Nachteil ist, dass globale Variablen mehr Speicher brauchen als alle anderen. Sie belegen immer den Platz und existieren beispielsweise im GameMaker über das ganze Spiel hinweg, sobald man sie erstellt hat.

Selbst im 21. Jahrhundert gilt es, möglichst sparsam mit Ressourcen umzugehen. Deshalb definieren wir Variablen, die wir nur in einem Event brauchen, mit *var* davor und begrenzen ihre Lebensdauer auf eben dieses Event und nicht das ganze Objekt bzw. dessen Instanz. Im schlimmsten Fall kostet das nicht nur unnötig Speicher, sondern auch Geschwindigkeit.

Das muss man allerdings auch etwas relativieren. Mein aktuelles Projekt Kopfnuss braucht rund 23MB Arbeitsspeicher. Im Menü, wo aufgrund des Hintergrundeffekts am meisten los ist, belastet es meine GPU (eine betagte GeForce GTX 670M) mit 11%. Die CPU wird mit 4 bis 5% belastet. Und das alles trotz dessen, dass ich alle Texte, Farben, Statistiken und Einstellungen als globale Variablen definiert habe. Im Spiel komme ich auf rund 180 globale Variablen, wovon ein paar als Arrays definiert wurden.

Wichtig ist, dass man klar abgrenzt, was man als globale Variable definiert und was nicht. Außerdem müssen diese eindeutig und einmalig bezeichnet werden. Hier hilft zum Glück GMS 2 etwas, indem es erkennt, ob eine Variable bereits im Spiel existiert. Wenn man sich daran hält, sollte es keine Probleme geben.

Date Created
21. Juni 2018
Author
sven