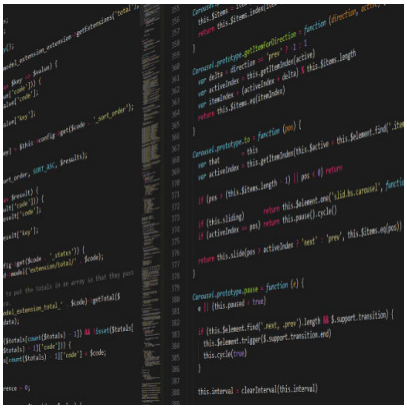


Scripte für Einsteiger

Description

In GameMaker hat man die Möglichkeit, Scripte zu erstellen, um diese in Objekten zu verwenden. Was der Vorteil daran ist und wie man eigene Scripte schreibt, zeigt dieses Einsteiger-Tutorial.

Wenn man sich ein wenig mit GameMaker befasst und Spiele macht, bemerkt man oft, dass man bestimmte Codefolgen immer wieder braucht. Diese sind auf verschiedene Objekte verteilt und wenn man mal etwas ändern will, muss man diese Objekte editieren und hoffen, dass man keine Stelle vergessen hat. Um den entgegen zu wirken, gibt es grundsätzlich zwei Möglichkeiten. Einerseits kann man mit Vererbungen arbeiten. Wie das Funktioniert, wird in einem späteren Tutorial erklärt. Doch Vererbungen sind nur in einigen Fällen die beste Wahl, Scripte sind oft besser. Schon alleine deshalb, weil man diese leichter in andere Objekte portieren kann.

Was ist ein Script?

Wer sich mit anderen Programmiersprachen bereits etwas auskennt, kann sich mit der Antwort zufrieden geben, dass ein Script sich in etwa so verhält wie eine Funktion, beispielsweise in PHP.

Sollte Dir das nichts sagen, kann man das natürlich auch anders erklären. Bei einem Script handelt es sich um für sich abgeschlossenen Code, den man von Außen ansprechen kann. Der Code führt dann Anweisungen aus, kann vorgegebene Variablen verarbeiten und auch einen Wert zurück liefern. Wenn man das Script in einem Objekt aufruft, verhält es sich so, als würde der Code im Objekt stehen. Der Vorteil ist, dass ich diesen Code immer wieder aufrufen kann, sowohl mehrfach in einem Objekt, wie auch aus verschiedenen Objekten heraus, muss aber nur den Code im Script ändern, wenn ich das will.

Wer ein wenig darüber nachdenkt, kommt schnell darauf, dass sich die GameMaker eigenen Befehle eigentlich wie Scripte verhalten und umgekehrt. Wenn wir also im GM mit *instance_destroy()* ein Objekt zerstören, wird im Hintergrund eine Befehlskette ausgeführt, die das Objekt veranlassen, sich mit allen Variablen etc. zu zerstören. Man beachte dabei die Klammer hinter dem Befehl. Darauf gehe ich später noch ein.

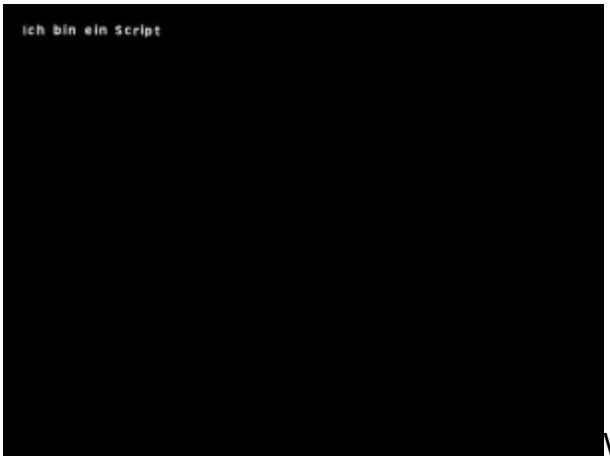
Ein anderer Befehl, wie etwa *instance_create(x, y, object)* erstellt ein Objekt. Hier geben wir noch x, y und den Namen des Objekts an. Wir geben also dem Befehl Parameter mit und genau das kann man auch mit einem Script machen.

Scripte erstellen und anwenden

Um das Prinzip besser verstehen zu können, mache ich mehrere Beispiele. Wir erstellen nun ein sehr simples Script. Dazu klicken wir in der oberen Leiste auf **Create a Script**. Nun öffnet sich ein neues Fenstern, den wir mit Code füllen können. Wir nennen das Script *scr_hallo*. Dann schreiben wir folgenden Code hinein:

```
draw_set_color(c_white);
draw_text(20, 20, „Ich bin ein Script!");
```

Das Fenster können wir schließen. Jetzt brauchen wir einen Raum. Ich habe ihn 640×480 Pixel groß gemacht und einen schwarzen Hintergrund gegeben. Da unsere Schrift weiß wird (*c_white*) ist es sinnvoll, einen dunklen Hintergrund zu verwenden. Dann erstellen wir ein Objekt, geben ihm den Namen *obj_scripttest* und platzieren dieses Objekt irgendwo im Raum. Den Raum können wir schließen und wenden uns dem Objekt zu. Dort erstellen wir das **Draw-Event** und ziehen uns den Execute-Code rein. Hier rufen wir nun das Script auf:



Wenn Du das Programm startest, sollte Dir das angezeigt

werden, was Du auf dem Screenshot sehen kannst. Glückwunsch, Du hast dein erstes Script erstellt!

Nun, es ist nicht sinnvoll, ein Script zu erstellen, um einen statischen Text anzuzeigen, aber es erklärt das Prinzip ganz gut. Man kann sich auch gut vorstellen, wie man hier beliebig langen Code erstellen kann.

Als nächstes versuchen wir ein etwas komplexeres Script und geben an das Objekt, welches es aufruft, eine Variable zurück.

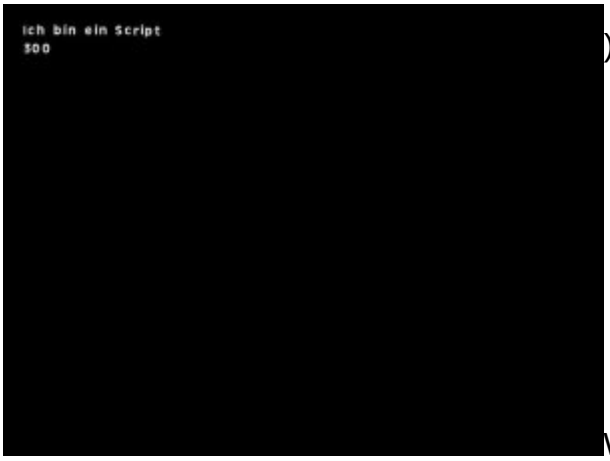
Wir erstellen, wie oben beschrieben, ein neues Script und nennen dieses *scr_ergebnis*. In das Script schreiben wir folgenden Code:

```
zahl1 = 100;
```

```
zahl2 = 50;

ergebnis = zahl1 * 2 - zahl2;

return ergebnis;
```

Dann gehen wir in unser Objekt und schreiben unter die Zeile *scr_hallo();* folgende Zeilen:

```
wert1 = scr_ergebnis();

wert2 = wert1 * 2;
```




```
);
```

Wenn Du das ganze startest, siehst Du eine zweite Zeile auf dem Bildschirm. Unter dem vorherigen Text taucht nun die Zahl 300 auf.

Jetzt versuchen wir einmal zu verstehen, was wir da eigentlich gemacht haben. Schauen wir uns noch einmal den Code vom Script an. Am Anfang haben wir 2 Variablen definiert, nämlich *zahl1* und *zahl2*. Diese enthalten die Werte 100 und 50. Die Variable *ergebnis* enthält das Ergebnis der Rechnung *zahl1* * 2 – *zahl2*. Da kommt dann 150 raus. Mit **return** geben wir diesen Wert zurück.

Im Objekt haben wir das Script nicht wie vorhin aufgerufen, sondern mit der Variable *wert1*. Das heißt, dass hier der Wert landet, den uns das Script mitteilt, in diesem Fall *ergebnis*, bzw. die Zahl 150. Mit diesem Wert kann man nun weiter rechnen, was wir mit wert2 machen. Wir multiplizieren *wert1* mit der Zahl 2 und geben dies, eine Zeile weiter unten, als Text aus.Daher kommt die 300 auf dem Bildschirm.

Wir stellen aber auch fest, dass die Schrift weiterhin weiß ist. Diese kommt noch vom Script *scr_hallo*. Sollten wir eine andere Farbe für die 300 haben wollen, können wir dies anpassen, indem wir beispielsweise *draw_set_color(c_aqua)* vor die Textausgabe von *wert2* stellen.

Variablen an ein Script übergeben

Nun ist es im oberen Beispiel vielleicht besser, *zahl1* und *zahl2* beim Aufruf des Scripts zu definieren. Dadurch sind wir viel flexibler. Um das zu realisieren, brauchen wir im Script Argumente. Das sind Werte, die das Script beim Aufruf durch ein Objekt oder ein anderes Script mitgeteilt bekommt. Wir ändern dazu das Script *scr_ergebnis* wie folgt ab.

```
// Rechnet mit zwei Zahlen
// Argument 1 = zahl1
// Argument 2 = zahl2

zahl1 = argument0;
zahl2 = argument1;

ergebnis = zahl1 * 2 - zahl2;

return ergebnis;
```

Viel mussten wir nicht ändern. Oben habe ich einen Kommentar hinzugefügt. Vor allem wenn man mit Argumenten arbeitet, sollte man erklären, was sie bedeuten. Besser ist noch, wenn man im Kommentar genau beschreibt, wozu dieses Script dient. Anschließend haben wir nur die zwei Zahlen durch *argument0* und *argument1* ersetzt. Das Programm kann man nun nicht mehr ohne Fehlermeldungen starten, da wir diese beiden Zahlen definieren müssen. Das machen wir im **Draw-Event** des Objekts. Wir ändern nur eine Zeile, indem wir *wert1* wie folgt definieren:

```
wert1 = scr_ergebnis(100, 50);
```

Spätestens jetzt wird klar, wozu die Klammern gut sind. Für Scripte kann man bis zu 16 Variablen definieren (von argumen0 bis argument15) und diese setzt man, wie auch bei den GM Befehlen, in eine Klammer und trennt sie mit einem Komme. Wenn wir das nun starten, erhalten wir das gleiche Ergebnis wie vorhin. Der Vorteil ist, dass wir nun beim Aufruf des Scripts die Zahlen definieren können, sogar müssen. Das macht die Sache noch flexibler.

Einsatzgebiete

Jetzt weißt Du, wie Scripte generell funktionieren, doch wie kann man einsetzen? Nachfolgend ein paar Beispiele, die Dich zur Verwendung von Scripten inspirieren sollen.

Ein Script ist hilfreich, wenn man seine globalen Variablen an einem Ort aufbewahren will. Man kann auch alle Config-Infos in ein Script schreiben, wie etwa die Lautstärke von Musik, Soundeffekten und ob das Spiel im Fenster- oder Fullscreenmodus angezeigt werden soll. Routinen, die INI-Dateien erzeugen, lesen und speichern kann man in Scripte auslagern. Man kann ein Script erstellen, welches einen Text manipuliert, zum Beispiel einem Text einen Schatten gibt.

Den Aufruf des 3D-Modus und die Erstellung einer Kamera kann man ebenso in ein Script auslagern wie Partikel, andere Effekte und sogar 3D-Objekte. Scripten sind kaum Grenzen gesetzt und wenn man Mal wieder dabei ist, Code in ein Objekt zu schreiben, den man mehrfach verwenden könnte, sollte man darüber nachdenken, ob man das nicht als Script erzeugt.

Date Created
6. Oktober 2016
Author
sven