Die 5 häufigsten GameMaker Fehler

Description

Vor allem durch die GameMaker eigenen Sprache GML lassen sich tolle Spiele realisieren. Doch auch erfahrenen Entwicklern können im Eifer des Gefechts leichte Fehler unterlaufen. ByteGame.de zeigt die fünf häufigsten Fehler, welche die Entwicklung stocken lassen.

5. Sprite ansteuern

Beim Umgang mit Sprites gibt es gleich drei typische Fehler. Brauche ich nun *image_index* oder *sprite_index*? *sprite_index* gibt den Spritenamen an, beispielsweise spr_player. *image_index* hingegen gibt an, welche Spritenummer innerhalb einer Animation angezeigt werden soll. Beispiel:

```
sprite_index = spr_testsprite;
image_index = 5;
```

Die erste Zeile definiert die Animation, die zweite, dass Sprite 5 angezeigt werden soll. Wirklich 5? Nein, hier sind wir beim zweiten Fehler. *image_index* beginnt immer bei 0. 5 ist somit 6.

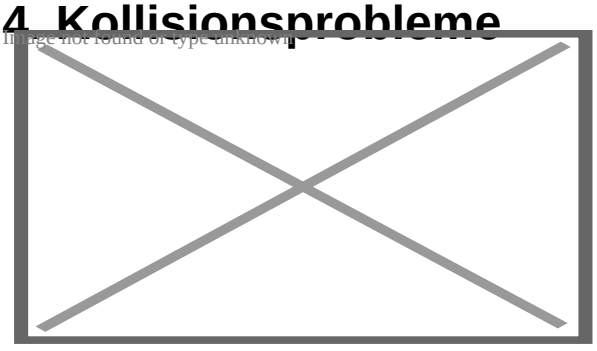
Der dritte kleine Fehler der gerne vorkommt ist der, dass man vergisst eine Animation zu stoppen.

```
image_speed = 0;
```

Das ist nötig wenn in einer Animation mehrere Zustände sind, etwa bei Buttons die Frames für „normal“ und „Mouse over«. In der Eile kann man diese Zeile im Create-Event schon Mal vergessen, aber der Fehler fällt zum Glück sofort auf.

4. Kollisionsprobleme

Image not found or type unknown



Kollision ist in Spielen ein großes Thema und die

auftretenden Probleme sind vielseitig. Ein häufiges Problem ist, dass ein Objekt ein paar Pixel vor der Wand anhält. Das liegt in den meisten Fällen an der Kollisionsmaske, wenn der Sprite nicht die ganze Fläche füllt. Ist es 64×64 groß, der Inhalt aber 60×60, dann haben wir zu jeder Seite 2 Pixel Luft. Hier muss einfach die Kollisionsmaske angepasst werden. *Automatic* ist nicht immer die beste Wahl, ebenso wie *Precise*. Eine allgemeine Aussage, was am besten ist, kann man leider nicht tätigen. In speziellen Fällen sollte die Maske per Hand angepasst werden, etwa wenn die Spielfigur einen Umhang hat, die nicht kollidieren soll.

Was man aber sagen kann: Solid sollten ausschließlich Objekte sein die sich nicht bewegen. Im schlimmsten Fall kann es sonst passieren, dass die Objekte ineinander hängen bleiben.

Häufig steht man auch vor der Frage, ob nun *place_empty* oder *place_free* die richtige Abfrage ist. *place_free* nimmt man, wie der Name sagt, um zu prüfen, ob am Ziel der Platz frei ist. Zur Anwendung kann es kommen, wenn man ein Objekt zufällig im Raum platzieren will, ohne es auf ein neues Objekt zu setzen. x und y werden zufällig generiert. Dann wird geschaut ob der Platz frei ist. Wenn nicht, erzeugt man die Zufallswerte neu. Wenn der Platz frei ist, setzt man das Objekt an die entsprechende Stelle. **Wichtig:** Das gilt nur für Objekte mit der Eigenschaft Solid. Ist das an den angegebenen Koordinaten ein Objekt ohne Solid, bekommt man als Rückmeldung *false*.

place_empty fragt ebenfalls ob die angegebenen Koordinaten frei sind, bezieht sich aber auf alle Objekte, die eine Kollisionsmaske haben.

Mit *place_meeting(x, y, object)* kann man die Kollision mit einem bestimmten Objekt prüfen. Auch diese Abfrage basiert auf den Kollisionsmasken.

Bevor man mit Kollisionen in Spielen arbeitet sollte man ein Testprojekt starten und experimentieren, bis man die Anwendung der Befehle verinnerlicht hat.

3. Spiel wird immer langsamer

Als Entwickler sollte man immer ein Auge auf die FPS haben, auch bei 2D-Spielen. Jedes Feature kostet ein paar Frames pro Sekunde und irgendwann überschreitet man eine Grenze, ab der das Spiel auf kleinen PCs nicht mehr flüssig läuft. Oder es läuft auf dem Desktop ganz gut, aber nicht mehr im Browser.

Dieses Problem kann viele Ursachen haben. Die drei häufigsten sind zu viele Objekte, zu viele aufwändige Abfragen und ein schlechtes Partikelsystem, was gerne in zu viele Objekte mündet.

Im GameMaker macht man i. d. R. aus jedem Sprite ein Objekt und stellt dieses dar. Selbst wenn es nur ein Pixel ist, welches über den Bildschirm wandert, hat es viele unsichtbare Eigenschaften. Sobald wir eine Instanz erstellen, geben wir ihm unbewusst diese Eigenschaften mit. GameMaker macht das automatisch. Die aktuellen Koordinaten, die Startkoordinaten, Layer bzw. Tiefe, id-Nummer und vieles mehr. Bei ein paar Dutzend Instanzen im Raum spielt das keine große Rolle, wenn es aber in die Hunderte oder Tausende geht, macht irgendwann der schnellste Prozessor schlapp.

Ein Klassiker ist der, dass zu viele Instanzen erzeugt werden. Wenn die Instanzen übereinander liegen, etwa bei Bullets oder Gegnern, kann man diese nicht erkennen. Mit *draw_text(20, 20, instance_count)*

kann man sich die Anzahl aller Instanzen im Spiel auf den Bildschirm zaubern lassen.

2. Instanz nicht gefunden

Nur zu gerne geht man als Entwickler davon aus, dass alle Instanzen da sind, die man gerade prüfen will. Sei es bei Kollisionen oder wenn man auf Variablen von Instanzen zugreifen möchte. Doch oh Wunder: Aus irgendeinem Grund existiert die Instanz noch nicht oder nicht mehr und schon quittiert GameMaker es mit einer schönen Fehlermeldung.

Unable to find any instance for object index '1' name 'obj_mouse'

Um dies zu vermeiden sollte man immer prüfen, ob die Instanz überhaupt existiert. Beispiel:

```
if instance_exists(obj_mouse)
{
 x = obj_mouse.x;
 y = obj_mouse.y;
}
```

instance_exists rettet Spiele!

1. Variable nicht definiert

Noch häufiger als bei Instanzen haben wir das Problem mit Variablen. Das kann zwei Ursachen haben. Entweder wir haben die Variable nicht definiert bevor wir darauf zugreifen, oder wir halten sie lokal.

Zur Anschauung:

Event Create

```
var sauerkraut = 10;
```

Event Draw

```
draw_text(20, 20, sauerkraut);
```

Variable obj_test.sauerkraut(100002, -2147483648) not set before reading it.

Mit *var* definieren wir eine lokale Variable. Lokal bedeutet in diesem Fall nicht Kneipe, sondern das sie auf ein Event beschränkt ist. In vielen Fällen braucht man nur lokale Variablen und da sie speichersparend sind, sollte man sie auch nutzen. Sobald wir eine Variable über mehrere Instanzen hinweg brauchen, dürfen wir sie nicht als lokale Variable behandeln, sonst passiert das, was im Beispiel gezeigt wurde.

Es lassen sich auch mehrere lokale Variablen in einer Zeile definieren:

```
var a, i, count, lederstrumpf;
```

Je mehr Variablen es werden, umso unübersichtlicher wird diese Methode, weshalb es ratsam ist, sie Zeile für Zeile mit einem Wert zu versehen.

```
var a = 0;
var i = 0;
var count = 1;
var lederstrumpf = „Romane von James Fenimore Cooper.";
```

GMS2 zeigt lokale und normale Variablen in unterschiedlichen Farben. Außerdem erkennt es, ob eine Variable bereits existiert. Wenn man sie tippt und sich das Wort nicht sofort verfärbt, dann existiert sie noch nicht. Wer also nicht gerade in die Luft oder auf die Tastatur starrt, während er tippt, kann dies auf dem Bildschirm recht gut erkennen.

Häufige Fehlerquellen gibt es genug. Welche passieren Dir immer wieder? Über Anregungen in den Kommentaren würde ich mich sehr freuen.

Date Created
17. Juli 2018
Author
sven