Schöner programmieren Teil 3 – Ressourcen und Kommentare

Description

Jeder, der im GameMaker mal gearbeitet hat, kennt das Problem. Wie benenne ich am besten meine Ressourcen? Natürlich geht dieses Problem weit über den GM hinaus. Die richtige Benennung von Dateien ist ein universelles Problem.

In diesem Teil setze ich einfach voraus, dass wir die sprachliche Hürde genommen haben. Unabhängig vom eigentlichen Namen stellt sich hier wie auch schon bei den Variablen die Frage, ob ich Worte mit Unterstrichen oder Großbuchstaben voneinander trenne. Ich selber habe es mir zur Gewohnheit gemacht in GML Variablen mit Großbuchstaben und Ressourcen mit Unterstrichen zu trennen. Das hat den Vorteil, dass man gleich sieht, ob es sich um eine Ressource oder eine Variable handelt.

Bei den Ressourcen kommt es aber gerne (eigentlich ungern) vor, dass man sie doppelt verwendet, beispielsweise Sprites und Objekte. Man hat einen Sprite mit dem Player und macht daraus ein Objekt mit dem Player. In GML ist es üblich, Ressourcen ein Präfix voran zu stellen. Gebräuchlich sind folgende Präfixe:

- spr_ für Sprite
- obj_ für Objekt
- scr_ für Script
- fnt_ für Font
- bg_ für Background
- snd_ für Soundeffekt

shd_ für Shader, oft wird aber auch sh_ genommen

Bei Räumen gibt es unterschiedliche Präfixe: rm_, r_, room_

Ich verwende letzteres, dann gibt es keine Missverständnisse.

Anschließend folgt der eigentliche Name, den man, wie oben beschrieben, durch Unterstriche trennen kann. Bei Level und anderen Ressourcen kann es passieren, dass man viele mit dem gleichen Namen

hat und nur hoch zählt. Meiner Erfahrung nach eignet sich dabei eine dreistellige Zahl ganz gut, beispielsweise room_level_001. Das System hilft auch bei der Benennung von Sprites, wenn man beispielsweise Bilder rendert, weil viele Programme die Zahlen nicht als ganze Zahl sortieren sondern nach den einzelnen Ziffern. Dann kommt nach 1 nicht die 2 sondern die 10, weshalb es angebracht ist, der 1 eine oder besser zwei Nullen voran zu stellen.

Wenn sich das Spiel in Missionen oder Akte unterteilt, die wiederum in Level gegliedert sind, sollte man das bei der Bezeichnung berücksichtigen. Dann ergibt sich ein room_m001_level_001. Das ist noch gut sichtbar und man kann es bequem in GM sortieren. Man könnte ebenso gut room_m_001_level_001 oder room_m001_level001 verwenden. Wichtig ist, dass man alle einheitlich beziffert.

Im GM hat man nicht nur die Möglichkeit die Ressourcen nach dem Alphabet zu sortieren, sondern auch die Option Ordner anzulegen, was man schon bei mittelgroßen Projekten tun sollte. Allerdings sollte man sich auch Ordnernamen und Sortierung gut überlegen, bevor man hier im Chaos versinkt und am Ende viel Zeit damit verliert, Ressourcen zu suchen. Die Ordner in GM sind nur virtuell und nicht physisch, weshalb ich Ressourcen die ich gerade brauche, gerne in den Hauptordner verschiebe und sie am Ende meiner Arbeit wieder einsortiere. Das spart unnötiges suchen und ist für mich der ideale Kompromiss zwischen Ordnung und effektives arbeiten.

Codekommentare

Code kommentieren, dass klingt so simpel wie überflüssig. Warum sollte man auch etwas, das ohnehin schon dran steht, noch einmal beschreiben? Man könnte tatsächlich meinen, wer viel Code kommentiert, hat einfach zu viel Zeit, oder schlicht keine Ahnung vom programmieren. Wer das behauptet, ist entweder ein Genie, oder hat selbst keine Ahnung.

Wie bereits in dieser Serie gesagt, ist Programmcode nicht gerade etwas, dass der Mensch intuitiv versteht. Selbst Programmierer haben ab und an Mühe, längeren Code zu lesen, selbst den eigenen. Das liegt an vielen Dingen, vor allem aber daran, dass man sich beim lesen ein technisches, teils recht abstraktes Bild eines Programms machen muss, um zu verstehen, was da passiert. Man muss also wie ein Computer denken, in Bildern, die Menschen verstehen.

Wer gut Kommentiert, hat es nicht nur im Team leichter. Kommentare helfen einem schnell dabei, den nachfolgenden Code zu verstehen, ohne ihn zu lesen. Vorausgesetzt der Kommentar ist richtig, verständlich und aktuell. Kommentare können aber auch mehr verwirren als helfen. Beispielsweise, wenn er so ausfällt:

```
// TODO: Das muss noch angepasst werden.
```

Missverständlich ist der Kommentar, weil man nicht weiß, was angepasst werden muss und warum. Deshalb sollten solche Kommentare ruhig etwas länger sein, damit man auch in ein paar Wochen, also meist viele 100 oder 1000 Zeilen später, noch weiß, was man sich dabei gedacht hat. Kommentare sind also meistens Nachrichten an sich selber in der Zukunft. Im schlimmsten Fall sind es Stolpersteine für sich selber in der Zukunft, oder für ein ganzes Team.

Wie das Beispiel oben schon zeigt, kann man mit Kommentaren auch Stellen markieren, an denen

man noch schrauben will. TODO oder FIXME sind dabei sehr hilfreich. Selbst mit Notepad++ kann man dann alle offenen Dateien danach absuchen und abarbeiten. Das ist sehr nützlich!

Kommentare sollten ehrlich sein. Wenn man eine Sache nicht hin bekommt oder nicht versteht, gehört das in den Kommentar. Beispiel:

```
// Hier wird geschaut, wie die Benutzergruppe heißt.
// TODO: Ggf. würde man die Datenbank weniger belasten, wenn man das vorher au
// Array speichert. Allerdings hatte ich keine Zeit das zu versuchen.
```

Kommentare sind auch dann nützlich, wenn man Code aus einem Fremden Forum oder einer Webseite verwendet, ggf. Hinweise auf Referenzen, Hilfen etc. kann man gerne mit rein nehmen. Wichtig ist vor allem ein Link zur Quelle, damit man immer wieder dort hin kann, wenn man ein Problem hat.

Am nützlichsten finde ich aber Kommentare mittlerweile, bevor ich etwas programmiere. Anfänger coden einfach mal darauf los, pfuschen herum, ändern ggf. das Konzept und kommentieren am Ende nicht einmal den Knäuel an Code. Viel besser ist es, vorher zu beschreiben, was man programmieren will. Dann hat man schon einen Leitfaden und kann am Ende kontrollieren, ob man wirklich das erhalten hat, was man sich vorgenommen hat. Beispiel:

```
/* Login Abfragen starten. Das „Verhalten" steuert dabei, was unter bestimmten
* Bedingungen passiert. 0 ist nichts, 1 ist, er wird angemeldet und 2 ergibt e
* Fehler auf der Login-Seite.
*/
```

Diese Technik hat mir schon manchmal geholfen, den Code recht entspannt herunter zu tippen. Wer dies noch nicht verwendet, sollte es auf jeden Fall einmal probieren.

Im vierten Teil geht es dann um die Einrückung. Das ist ein größeres Thema, welches einen eigenen Teil verdient hat.

Date Created
11. Oktober 2016
Author
sven