Mehrsprachige Spiele

Description

Die Welt ist ein Dorf. Allerdings eines, in dem sehr viele Sprachen gesprochen werden. Als Entwickler ist es wichtig, in seinem Projekt zumindest so viele Sprachen wie möglich einzuplanen. Um dies tun zu können, sind ein paar Vorkenntnisse nötig.

Die Grundlagen

Wer nur eine Sprache unterstützen möchte, macht es sich oft bequem und schreibt den Text direkt in den Code. In PHP würde dies so aussehen:

```
echo „Hallo Spieler!";
```

Wenn man wirklich nur wenige Zeilen hat, ist das auch kein Problem. Je größer ein Projekt wird, umso nerviger wird es allerdings, die Texte im Code zu warten. Textliche Inhalte können sich ändern, Fehler gefunden werden und wenn man zunächst unzählige Dateien durchsuchen muss um den Text zu finden, kann das durchaus nervtötend sein. Ein anschauliches Beispiel wäre, dass man eine Geschichte erzählen möchte und sich am Ende der Produktion der Name eines Hauptcharakters ändert. Nun wird aus „Harry“ ein „Sam“ und der Programmierer hat irre viel Spaß, alle Harrys zu suchen und zu ersetzen. Dafür gibt es in Entwicklungsumgebungen zwar Funktionen, aber je nach Name kann sich der zu ersetzende Text auch auf Variablen auswirken. Oder Bestandteil anderer Wörter in Texten sein.

Deshalb ist es ratsam, möglichst früh den Text auszulagern. Dies geschieht heutzutage nicht nur mit Texten so. Im Webdesign lagert man aus ähnlichen Gründen auch Design-Dateien aus. Kurz und gut, der Text muss in einen eigenen Bereich.

Abgeschobene Texte

Die einfachste Methode besteht darin, eine extra Datei oder ein eigenes Skript (GameMaker) anzulegen. In PHP könnte die Datei *en.php* lauten und sich in einem Ordner mit dem Namen *language*

befinden. Im GameMaker kann man eine Sprachdatei auch als INI, TXT oder JSON einbinden. Auch Varianten, in denen die Texte in einer Datenbank abgelegt werden, sind möglich und werden praktiziert. Dies kann den Vorteil haben, dass bei der Übersetzung in viele Sprachen auch mehrere Übersetzer gleichzeitig bequem am Projekt arbeiten können.

Im simpelsten Fall, wie der PHP Datei, bekommt jeder Text einen individuellen Variablennamen.

```
$lang_langmenu_header= "language";
$lang_langmenu_en= "english";
$lang_langmenu_de= "german";
```

Der Vorteil bei dieser Bezeichnung ist, dass man im Code sofort sieht, dass es eine Sprachdatei ist, in welchen Bereich die Variable gehört (hier durch *langmenu* beziffert) und was sich dahinter verbergen könnte. Diese Eindeutigkeit kann sehr nützlich sein, da man bereits im Code erkennen kann, ob der Text an der richtigen Stelle sitzt.

Im GameMaker könnte das Skript **scr_text_en** heißen. Die einfachste Methode besteht darin, die Texte als globale Variablen aufzurufen.

```
global.txt_languageName = „english";
```

Das hat den Vorteil, dass die Texte nur einmal geladen werden müssen. Der Nachteil ist allerdings auch leicht zu sehen: Die gezeigten Methoden sind nicht gerade Speicherschonend.

Optimierungen

Nehmen wir an, wir hätten Listen wie Nationen oder Kontinente. Da bietet es sich an, aus normalen Variablen Arrays zu machen. Im GameMaker könnte das so aussehen:

```
// Continents
global.txtContinent[0] = „No Continents";
global.txtContinent[1] = „Asia";
global.txtContinent[2] = „Australia";
global.txtContinent[3] = „Antarctica";
global.txtContinent[4] = „Africa";
global.txtContinent[5] = „Europe";
global.txtContinent[6] = „North America";
global.txtContinent[7] = „South America";
```

Doch wenn man schon dabei ist, warum sollte man nicht gleich alle Texte als ein Array machen? In PHP sieht das heutzutage meist so aus:

```
<?php
$lang = array();

// Language Menu
$lang = array_merge($lang, array(
    'LANGMENU_HEADER' => 'Sprache',
    'LANGMENU_EN' => 'englisch',
    'LANGMENU_DE' => 'deutsch',
));
```

Ausgegeben wird dann der Text als

```
echo $lang['LANGMENU_HEADER'];
```

Diese Methode hat gleich mehrere Vorteile. Ein Array ist um einiges effizienter als tausende einzelner Variablen. Im Gegensatz zu der gezeigten Array-Methode mit GameMaker haben wir aber hier nicht einfach Zahlen, sondern eine konkrete Bezeichnung (*LANGMENU_HEADER*). Zudem lässt sich die Sprachdatei mit *array_merge* sehr übersichtlich halten. Einzelne Abschnitte für Bereiche können extra kommentiert werden, was unheimlich praktisch ist.

Im GameMaker sähe das so aus:

```
lang[0, 0] = „Hello world!";

// und im Draw-Event

draw_text(x, y, string(lang[0, 0]));
```

Moment, da scheint etwas nicht zu stimmen. Was soll man im Code mit *lang[0, 0]* anfangen? Bei einem zweidimensionalen Array kann die erste Zahl als Kategorie und die zweite als String gesehen werden. Wirklich hilfreich ist das nicht. Das Problem: Es gibt im GameMaker keine assoziativen Arrays. Es gibt **ds_grid**, was manches vereinfachen kann, aber wirklich helfen tut es diesbezüglich auch nicht. Wie wir das Problem lösen, sehen wir gleich.

Sprachen mit ds_grid

Wir legen zunächst irgendwo die Texte fest.

```
lang = ds_grid_create(2, 1);

// englisch
ds_grid_add(lang, 0, 0, „Hello World!");

// deutsch
ds_grid_add(lang, 1, 0, „Hallo Welt!");
```

Wie gesagt, entspricht **ds_grid** einem zweidimensionalem Array. *lang* ist das Grid. Anschließendfolgen *x* und *y*, die wir mit Zahlen belegen. Die erste Zahl ist die gewählte Sprache. 0 steht hier fürEnglisch, 1 für Deutsch. Die zweite Zahl wird dem String zugeordnet. Nun muss der Text nochangezeigt werden.

```
// Englischer Text
var currentLang = 1;
draw_text(x, y, string(ds_grid_get(lang, currentLang, 0)));

// Deutscher Text
var currentLang = 1;
draw_text(x, y, string(ds_grid_get(lang, currentLang, 1)));
```

Das wirkt auf den ersten Blick ein wenig umständlicher, was es auch ist. Wenn man sich aber schon die Mühe macht, dann wäre es durchaus sinnvoll, die Daten in eine INI auszulagern und anschließend mit einer Schleife in ein grid einzuladen. So viel zur Theorie. Das Problem ist nur, dass man im GameMaker nicht prüfen kann, wie viele Einträge eine Sektion beinhaltet. In INIs können wir nur eine Sektion und einen key gezielt ansteuern. Mit einer TXT wäre das Möglich, beispielsweise mit *file_text_readln*, aber dann fehlt uns die Sektion. Die Unterteilung in Sektionen kann nützlich sein, wenn man gezielt einen Abschnitt einlesen möchte. Eine Möglichkeit wäre, alles in einzelne Textdateien auszulagern. Jede TXT würde einer Sektion der INI entsprechen.

Das hätte zwei Vorteile: Erstens haben wir so die Möglichkeit aufgrund der Textfunktionen mit Schleifen zu arbeiten und zweitens können wir immer nur die Texte laden, die wir gerade brauchen. Aufgrund der Komplexität verzichte ich in diesem Tutorial auf ein entsprechendes Beispiel.

Externe Dateien haben den Vorteil, dass Spieler bzw. eine Community hinter dem Spiel mit der Übersetzung helfen können. Wer das nicht haben möchte, kann diese Dateien natürlich auch verschlüsseln. So kann man die unverschlüsselten Dateien immer noch an ausgewählte Leute senden, die mit einer INI oder TXT wesentlich besser klarkommen sollten als mit GML-Code.

Konkretes GameMaker Beispiel mit ds_list

Da in Foren oft **ds_list** empfohlen wird, möchte ich dies zeigen und auf ein paar Probleme verweisen. Zunächst legen wir eine INI für die Sprache an.

INI Datei

```
[start]

hello = „Hallo Welt!"
player = „Du bist diese Spielfigur."
```

Das ist die de.ini. Eine zweite Datei könnte en.ini sein.

In einem Skript oder einem Create-Event laden wir die entsprechende Datei ein und füllen unser Grid.

Create-Event

```
currentLang = 0; // Wir legen hier Deutsch fest

if (currentLang == 0)
{
 ini_open(working_directory + „de.ini");
} else {
 ini_open(working_directory + „en.ini");
}

text = ds_grid_create(2, 2);

ds_grid_add(text, currentLang, 0, ini_read_string("start", „hello", „huhu"));
ds_grid_add(text, currentLang, 1, ini_read_string("start", „player", „player")

ini_close();
```

Die Sprache currentLang könnten wir auch global steuern. Im Beispiel geht es nur um die beiden Dateien. Wenn mehr Sprachen gewünscht sind, sollte man die Abfrage über Switch regeln und erst dann **ini_open** anwenden. Die Variable *text* ist unser Grid. Wichtig ist: Wir haben zwei Sprachen und für jede Sprache gibt es zwei Zeilen. *huhu* und *player* sind default-Werte, falls der Eintrag nicht gefunden wird.

Anschließend schaufeln wir jede Zeile einzeln in das Grid. Aufgrund der Section und dem Key haben wir im Code wenigstens eine Ahnung, was für ein Text angezeigt wird. Der default-Text ist nur für den Testzwecke.

Nun zeichnen wir den Text.

Draw-Event

```
draw_set_color(c_white);

draw_text(x, y, string(ds_grid_get(text, currentLang, 0)));
draw_text(x, y+30, string(ds_grid_get(text, currentLang, 1)));
```

Ein generelles Problem beim GameMaker ist die Umstellung auf ein Sandbox-System. **working_directory** muss nicht immer und überall funktionieren. Auf dieses Problem gehe ich bei einem späteren Tutorial ein.

Es wird deutlich, dass **ds_grid** keine große Hilfe im Code ist. Man stelle sich hunderte von Zeilen Code vor, in denen die Texte nur Zahlen sind. Das kann schon bei einem kleineren Projekt zum Horror werden.

Sprachen mit ds_map

ds_map ist eine von zahlreichen Datenstrukturen im GameMaker und in diesem Fall am ehesten das

Mittel der Wahl. Es ist das, was einem assoziativen Array am ehesten nahe kommt. Es besteht aus einer id, einem key und dem eigentlichen Wert. Ein Eintrag könnte so lauten:

```
ds_map_add(id, „Key", „Das ist ein Text!");
```

Damit kann man vernünftig arbeiten!

Create-Event

```
currentLang = 0; // Wir legen hier Deutsch fest

if (currentLang == 0)
{
 ini_open(working_directory + „de.ini");
} else {
 ini_open(working_directory + „en.ini");
}

text = ds_map_create();

ds_map_add(text, „hello", ini_read_string("start", „hello", „huhu"));
ds_map_add(text, „player", ini_read_string("start", „player", „player"));

ini_close();
```

Im Prinzip ist das der selbe Code wie bei **ds_list**, nur mit den Map-Anweisungen. Im Beispiel legen wir den Key so fest, wie er in der INI staht.

Draw-Event

```
draw_set_color(c_white);

draw_text(x, y, string(ds_map_find_value(text, „hello")));
draw_text(x, y+30, string(ds_map_find_value(text, „player")));
```

Jetzt erkennt man auch, was sich für ein Text dahinter verbirgt. Damit kann man arbeiten. Hier noch einmal der Unterschied:

```
draw_text(x, y, string(ds_grid_get(text, currentLang, 0)));
draw_text(x, y, string(ds_map_find_value(text, „hello")));
```

Wenn **ds_map** an dieser Stelle einen Nachteil hat, dann der, dass hier nicht bekannt ist, welche Sprache benutzt wird. Das ist aber aus meiner Sicht nicht relevant. Es ist immer die Sprache, die der Spieler gewählt hat.

Rechtfertige den Aufwand

Man sollte nicht bei jedem kleinen Projekt mit Kanonen auf Spatzen schießen. Es wird aber auch schnell klar, das die Trennung zwischen Code und Text eine sinnvolle Angelegenheit ist. In der Praxis sieht das meistens so aus, dass man sich irgendwann für eine Methode entscheidet und diese auch für

mehrere Projekte anwendet. Grundlegende Funktionen, und dazu zählt auch die Handhabung von Texten, sollte man ohnehin definieren, bevor man mit dem eigentlichen Spiel beginnt.

Eine brauchbare Lösung im GameMaker ist die mit INIs und **ds_map**. Ausgeklügelter wäre es mit TXT-Dateien, bei denen der Inhalt über Schleifen eingelesen wird. Das hat allerdings auch seine ganz eigene Tücken. Denkbar sind auch JSON-Dateien. Meine persönliche Meinung in Bezug auf GameMaker ist die, diesen Weg eher kritisch zu sehen. JSON Wird vom GM noch nicht lange unterstützt und Yoyo-Games hat ein großes Talent dafür, mit der eigenen Sprache GML Schabernack zu treiben. Das bedeutet: Heute kann das alles funktionieren, mit der nächsten Version vielleicht nicht. Diesbezüglich sollte man auf Funktionen vertrauen, die den Sprung von GMS 1 auf 2 halbwegs überlebt haben.

Date Created
27. Mai 2018
Author
sven