Arrays (Datenfelder)

Description

Arrays sind eine wichtige Datenstruktur-Variante, auch „Felder” genannt. Viele Aufgaben in der Programmierung lassen sich nur mit der Hilfe von Arrays mit einem überschaubaren Aufwand bewältigen.

Bei Arrays handelt es sich immer um zusammengehörige, ähnliche Daten. Diese können Listen, aber auch Tabellen sein, die sich im Speicher des Computers befinden und dort sortiert, manipuliert und auf dem Bildschirm ausgegeben werden können. In einigen Fällen dienen Arrays als Hilfsmittel im Hintergrund, um Aufgaben zu bewältigen, von denen der Anwender / Spieler nichts ahnt.

Dazu ein konkretes Beispiel: Wir wollen eine Anzahl von Namen auf dem Bildschirm ausgeben. Eventuell wissen wir nicht, wie viele Namen es am Ende werden, aber angenommen, wir wissen, dass es sich um fünf Namen handelt, so kann man diese, mit dem bisherigen Wissen, wie folgt verarbeiten:

```
namen0 = 'Tom';
namen1 = 'Andreas';
namen2 = 'Susi';
namen3 = 'Caroline';
namen4 = 'Peter';
```

Im **Draw-Event** können diese Namen nun ausgegeben werden, zum Beispiel mit folgenden Zeilen:

```
draw_text(20, 20, namen0);
draw_text(20, 40, namen1);
draw_text(20, 60, namen2);
draw_text(20, 80, namen3);
draw_text(20, 100, namen4);
```

Bei fünf Namen hält sich der Aufwand in Grenzen, doch bei 100 Namen oder einer unbekannten Anzahl von Namen, kann das ein Problem werden. Genau hier kommen Arrays zum Einsatz.

Eindimensionale Arrays

Eindimensionale Arrays sind im Prinzip listen. Sie bestehen aus einem Index-Wert und dem Inhalt. Der Index ist nötig, um die Werte unterscheidbar zu machen. Der Index steht im GameMaker immer in einer eckigen Klammer. Beispiel:

```
namen[0] = 'Tom';
```

Beim Deklarieren fällt Leuten, die sich bereits ein wenig mit GML befasst haben auf, dass es ähnlich aussieht wie das Aufrufen von Alarmen. Das kommt daher, weil die 12 Alarme (von 0 bis 11) über ein Array aufgerufen werden. Wenn Du also schon einmal einen Alarm aufgerufen hast, hast Du bereits mit Arrays gearbeitet, ohne Dir dessen bewusst zu sein.

Wandeln wir das eingangs gezeigte Beispiel in ein eindimensionales Array um. Im **Create-Event** würde das dann so aussehen:

```
namen[0] = 'Tom';
namen[1] = 'Andreas';
namen[2] = 'Susi';
namen[3] = 'Caroline';
namen[4] = 'Peter';
```

Das ist noch kein bedeutender Unterschied zu vorher. Da wir aber über einen Index verfügen, können wir im **Draw-Event** die Namen über eine for-Schleife ausgeben.

```
yy = 20;

draw_set_color(c_white);

for (i = 0; i < 5; i++)
{
    draw_text(20, yy, namen[i]);
    yy += 20;
}
```

Auf dem Bildschirm erscheinen nun die 5 Namen untereinander.

Zugegeben, bei 5 Namen ist das noch keine echte Verbesserung, aber selbst wenn wir 1000 Namen hätten, wäre der Code im **Draw-Event** nicht komplexer.

Schauen wir uns aber erst einmal den Code genauer an. Am Anfang definieren wir mit *yy* eine Hilfsvariable. Dann definieren wir eine Farbe für den Text. Nun kommt die for-Schleife. Diese wird 5 Mal durchlaufen. *i* ist dabei die entscheidende Variable. In der **draw_text-Zeile** sehen wir für *x* 20, *yy* und unser Array. In der eckigen Klammer steht keine Zahl, sondern die Variable *i*. Die beginnt bei 0 und wird mit jedem Durchlauf um 1 hoch gezählt, bis sie bei 4 ist (kleiner 5). Mit jedem Durchlauf wird *yy* um 20 erhöht. Somit erscheint jede neue Zeile 20 Pixel weiter unten.

Wer mit der for-Schleife noch Probleme hat, sollte sich das Schleifen-Tutorial noch einmal durchlesen.

Wie gesagt, funktioniert das auch bei einer unbekannten Anzahl von Einträgen. Das kann passieren, wenn der Benutzer / Spieler nacheinander beliebig viele Namen eingeben darf, oder wir eine Liste aus einer Datenbank auslesen. Um bei einem eindimensionalen Array die Anzahl der Einträge zu ermitteln, bietet der GameMaker den Befehl **array_length_1d** an.

Wichtig zu wissen ist, dass es, genau genommen, nicht beliebig viele Namen sind. GameMaker gibt eine Grenze von 32 000 vor. Für die meisten Fälle sollte das aber reichen.

Das oben stehende Beispiel ändern wir nun ab. Dabei tun wir so, als wüssten wir nicht, dass es 5 Einträge sind. Unter die letzte Zeile im **Create-Event** schreiben wir noch folgende Zeile:

```
anzahl = array_length_1d(namen);
```

Im **Draw-Event** machen wir aus der Zahl 5 die Variable *anzahl*. Um uns zu kontrollieren, setzen wir an das Ende des Events noch folgende Zeile:

```
draw_text(20, yy, string(anzahl));
```

Wenn wir das Programm starten, sehen wir die gleiche Liste wie vorhin, allerdings mit der Zahl 5 in der unteren Zeile.

Selbst wenn wir keine unbekannte Anzahl an Namen haben, können wir die Liste der Namen beliebig erweitern, ohne die for-Schleife anpassen zu müssen.

Natürlich lassen sich auf die gleiche Weise auch Zahlen verarbeiten. In den meisten Programmiersprachen kann man solche Listen auch beliebig sortieren, in GameMaker ist das aber etwas schwierig. Die Entwicklungsumgebung bietet aber einen eigenen Befehl, der wie ein eindimensionales Array funktioniert, mit dem wir aber unter Anderem bequem sortieren können.

ds_list als Array-Ersatz

In absehbarer Zeit wird es ein extra Tutorial zum Thema geben, aber wir kommen jetzt nicht drum herum, ein wenig vorzugreifen. Wie bereits gesagt, sind die Möglichkeiten von eindimensionalen Arrays im GameMaker limitiert. Dies kompensieren die **ds_list**-Befehle. Im Kern funktioniert eine ds_list wie ein Array, wir haben dafür aber die Möglichkeit, die Listen einfach zu handhaben.

Wir bleiben beim oben stehenden Beispiel und ändern dieses so ab, dass wir mit einer Liste arbeiten können. Die Liste wird dabei gleich sortiert:

Event Create:

```
namen = ds_list_create();

ds_list_add(namen, 'Tom');
ds_list_add(namen, 'Andreas');
ds_list_add(namen, 'Susi');
ds_list_add(namen, 'Caroline');
ds_list_add(namen, 'Peter');
```

```
anzahl = ds_list_size(namen);

ds_list_sort(namen, true);
```

In der ersten Zeile erstellen wir die Liste, die wir mit „namen” benennen. Anschließend fügen wir mit **ds_list_add** die einzelnen Namen hinzu. In den Klammern geben wir die Liste an (namen) und den Wert, der eingetragen werden soll. Um den Index müssen wir uns hier nicht kümmern.

Wie gewohnt ermitteln wir dann die Anzahl. Bei Listen funktioniert das mit dem Befehl **ds_list_size**. Der Befehl sagt uns, wie viele Einträge in der Liste „namen” sind und gibt diesen Wert an die Variable *anzahl* weiter.

In der letzten Zeile sortieren wir diese Liste. **ds_list_sort** sortiert die Liste automatisch. *true* sortiert die Liste aufsteigend, *false* würde die Liste absteigend sortieren.

Event Draw:

```
yy = 20;

draw_set_color(c_white);

for (i = 0; i < anzahl; i++)
{
    draw_text(20, yy, ds_list_find_value(namen, i));
    yy += 20;
}

draw_text(20, yy, string(anzahl));
```

Der Code ist fast identisch mit der Array-Variante. Lediglich bei der Ausgabe wird der Wert mit **ds_list_find_value** ausgegeben.

Wenn man das Prinzip eines Arrays verstanden hat, ist die Verwendung von Listen eigentlich ganz einfach.

Weitere Beispiele für eindimensionale Arrays

Arrays sind nicht nur bei Namen praktisch, sondern auch bei Zahlenwerten. Im Beispiel für einen Aktienkurs werden viele Zahlen zufällig generiert und in ein Array gespeichert. Statt eines Aktienkurses könnte man auch Temperaturen, Luftdruck und andere Daten verwenden. Mit wenigen Zeilen Code kann man, vor allem in Kombination mit Schleifen, sehr viele Daten handhaben.

Zweidimensionale Arrays

Eindimensionale Arrays sind zwar sehr praktisch, zweidimensionale Arrays dagegen sind für manche Anwendungsgebiete unentbehrlich. Von ihrer Logik her funktionieren sie wie eine Tabelle. Es handelt sich somit um Zeilen und Spalten. Der Aufbau sieht so aus: array[index, spalte]

Um nicht unnötig zu verwirren, schauen wir uns ein konkretes Beispiel an. Angenommen, wir wollen eine Highscoreliste anzeigen. Jeder Spieler hat einen Namen, eine Punktzahl und ein Datum, wann diese Punktzahl erspielt wurde. Wir haben somit drei Werte und einen Index.

Event Create:

```
// Wert 1 = Name
// Wert 2 = Punktzahl
// Wert 3 = Datum

sp_punkte[0, 0] = 'Tom';
sp_punkte[0, 1] = '16000';
sp_punkte[0, 2] = '2016-04-16';

sp_punkte[1, 0] = 'Andreas';
sp_punkte[1, 1] = '17500';
sp_punkte[1, 2] = '2016-06-02';

sp_punkte[2, 0] = 'Susi';
sp_punkte[2, 1] = '31250';
sp_punkte[2, 2] = '2016-02-27';

sp_punkte[3, 0] = 'Caroline';
sp_punkte[3, 1] = '12825';
sp_punkte[3, 2] = '2015-12-31';

sp_punkte[4, 0] = 'Peter';
sp_punkte[4, 1] = '20900';
sp_punkte[4, 2] = '2015-09-16';

anzahl = array_height_2d(sp_punkte);
```

Ersteinmal erstellen wir unseren Array, der *sp_punkte* heißt. Ich schreibe bei solchen Arrays gerne an den Anfang, für was die Werte stehen, dann gibt es später keine Verwirrung.

Für jeden Spieler haben wir einen Block aus drei Zeilen. Schauen wir uns den ersten Block genauer an.

```
sp_punkte[0, 0] = 'Tom';
sp_punkte[0, 1] = '16000';
sp_punkte[0, 2] = '2016-04-16';
```

Zwei Dinge fallen auf. Zunächst einmal die Tatsache, dass die erste Zahl in den eckigen Klammern immer eine 0 ist. Das ist, wenn wir beim Bild der Tabelle bleiben, die Zeile. Die zweite Zahl hingegen unterscheidet sich immer. Das ist die Spalte. 0 ist Wert 1, somit der Name. 1 ist Wert 2, also die Punktzahl und 2 ist der dritte Wert, nämlich das Datum. Nicht verwirren lassen: Ich habe das amerikanische Datumsformat verwendet.

Da wir das Prinzip verstanden haben, können wir uns diese drei Werte wunderbar in einer Zeile einer Tabelle vorstellen. Der zweite Block ist demnach die zweite Zeile mit seinen drei Werten und so weiter.

Am Ende ermitteln wir wieder, wie viele Zeilen wir haben. Bei einem zweidimensionalen Array hilft uns

der Befehl **array_height_2d** weiter. Die Anzahl der Spalten würden wir mit **array_length_2d** erfahren, aber das brauchen wir jetzt nicht.

Nun wollen wir die Tabelle zeichnen.

Draw-Event:

```
yy = 20;

draw_set_color(c_white);

for (i = 0; i < anzahl; i++)
{
    draw_text(20, yy, sp_punkte[i, 0]);
    draw_text(200, yy, sp_punkte[i, 1]);
    draw_text(400, yy, sp_punkte[i, 2]);
    yy += 20;
}

draw_text(20, yy, string(anzahl));
```

Auf dem Bildschirm erscheint nun eine tabellarische Liste. Ganz unten steht noch die Zahl der Einträge zur Kontrolle.

Im Vergleich zum eindimensionalen Array hat sich kaum etwas geändert. Wir haben für jede Spalte eine eigene **draw_text** Zeile und in dieser wird entsprechend das zweidimensionale Array ausgegeben. Die Variable *i* gibt weiterhin die Zeile an, wie schon bei der Liste. Die zweite Zahl ruft die Spalte ab. Wir können nun auch ganz einfach Punktzahl und Name Tauschen:

Draw-Event:

```
yy = 20;

draw_set_color(c_white);

for (i = 0; i < anzahl; i++)
{
    draw_text(20, yy, sp_punkte[i, 1]);
    draw_text(200, yy, sp_punkte[i, 0]);
    draw_text(400, yy, sp_punkte[i, 2]);
    yy += 20;
}

draw_text(20, yy, string(anzahl));
```

Der Unterschied ist, dass wir nun erst die Punkte (1) anzeigen und dann den Namen (0).

Weitere Beispiele für zweidimensionale Arrays

Solche Arrays kann man natürlich nicht nur verwenden, um Tabellen auf dem Bildschirm zu erzeugen.

Man kann auch ein Brettspiel wie Schach oder Dame damit steuern, oder ein Inventar. Für Kartenspiele kann man Karten und ihre Werte speicher. Ein weiterer Anwendungsfall sind Rollenspiele und natürlich Match3-Games. Hier bilden, wie bei Brettspielen, Arrays die Grundlage für die Spielkoordinaten.

Wenn man sich das genau überlegt, sind zweidimensionale Arrays nicht nur Tabellen, sondern auch Gitter. Das fällt vor allem auf, wenn man sich Inventar und Brettspielen befasst.

Ebenso lassen sich über solche Arrays viele Einheiten steuern. Die Zeilen bilden einzelne Instanzen ab, in den Spalten stehen die zuständigen Werte. Letztlich ist es egal, ob es um Karten oder Panzer auf einem Schlachtfeld geht, das Prinzip dahinter ist gleich.

Kurz gesagt: Alles, was man als Tabelle oder in einer Gitterstruktur darstellen kann, kann eine Aufgabe für Arrays sein.

ds_grid als Array-Ersatz

Wenn man ein zweidimensionales Datenfeld sortieren will, ist es auch hier einfacher auf Funktionen des GameMakers zurückzugreifen. Dafür gibt es ds_grid. Mit **ds_grid_create** legt man das Gitter bzw., um beim Beispiel zu bleiben, die Tabelle an. Das Beispiel würde nun so aussehen.

Event Create:

```
// Wert 1 = Name
// Wert 2 = Punktzahl
// Wert 3 = Datum

sp_punkte = ds_grid_create(3, 5);

ds_grid_set(sp_punkte, 0, 0, 'Tom');
ds_grid_set(sp_punkte, 1, 0, '16000');
ds_grid_set(sp_punkte, 2, 0, '2016-04-16');

ds_grid_set(sp_punkte, 0, 1, 'Andreas');
ds_grid_set(sp_punkte, 1, 1, '17500');
ds_grid_set(sp_punkte, 2, 1, '2016-06-02');

ds_grid_set(sp_punkte, 0, 2, 'Susi');
ds_grid_set(sp_punkte, 1, 2, '31250');
ds_grid_set(sp_punkte, 2, 2, '2016-02-27');

ds_grid_set(sp_punkte, 0, 3, 'Caroline');
ds_grid_set(sp_punkte, 1, 3, '12825');
ds_grid_set(sp_punkte, 2, 3, '2015-12-31');

ds_grid_set(sp_punkte, 0, 4, 'Peter');
ds_grid_set(sp_punkte, 1, 4, '20900');
ds_grid_set(sp_punkte, 2, 4, '2015-09-16');

anzahl = ds_grid_height(sp_punkte);
```

```
ds_grid_sort(sp_punkte, 1, false);
```

Im Kommentar halten wir erneut die Werte fest, um uns daran zu orientieren. Darunter legen wir das Gitter an.

```
sp_punkte = ds_grid_create(3, 5);
```

Unsere Tabelle heißt *sp_punkte*. **ds_grid_create** erzeugt die Tabelle. Die 3 steht für die Anzahl der Spalten, die 5 für die Anzahl der Zeilen. Eine Eigenheit ist, dass GameMaker bei diesem Befehl nicht bei 0 beginnt, sondern bei 1. 3, 5 ist somit wirklich 3 und 5 und nicht 4 und 6!

Die Tabelle ist angelegt, nun werden die Zellen mit **ds_grid_set** gefüllt. Hier gilt es zu beachten, dass Länge und Breite vertauscht sind. Wir geben somit erst die Spalte und dann die Zeile an.

Am Ende ermitteln wir mit **ds_grid_height** die Zeilen und sortieren die Tabelle nach Punkten.

Vorsicht: Bei **ds_grid_create** wird die 0 nicht berücksichtigt, bei **ds_grid_set** aber schon!

Nun haben wir unsere sortierte Tabelle und möchten diese, wie gewohnt, ausgeben.

Event Draw:

```
yy = 20;

draw_set_color(c_white);

for (i = 0; i < anzahl; i++)
{
    draw_text(20, yy, ds_grid_get(sp_punkte, 1, i));
    draw_text(200, yy, ds_grid_get(sp_punkte, 0, i));
    draw_text(400, yy, ds_grid_get(sp_punkte, 2, i));
    yy += 20;
}

draw_text(20, yy, string(anzahl));
```

Auch hier ist fast alles wie in den vorherigen Beispielen. Der Zellenwert wird mit **ds_grid_get** ausgelesen. Hier gilt: Die 0 muss berücksichtigt werden, außerdem wird **erst die Spalte und dann die Zeile** angegeben!

Das Thema Datenfelder kommt in Spielen immer wieder aufs Tapet, weswegen es zu den Spezialfunktionen des GameMakers noch weitere Tutorials geben wird. Ich hoffe, dass ich bis hierhin die Grundlagen vermitteln und ein grundlegendes Verständnis für das Thema schaffen konnte.

Date Created
25. Oktober 2016
Author
sven