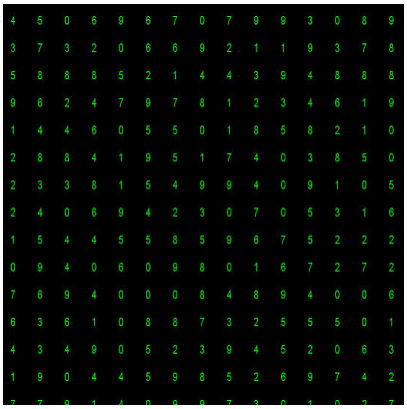
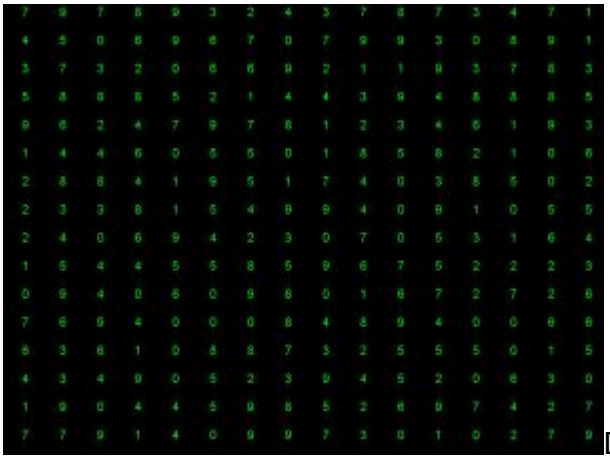


Falling numbers

Description

In einigen Tutorials habe ich bereits gezeigt, wie man mit dem Erzeugen von vielen Instanzen coole Effekte erzeugen kann, zum Beispiel beim Pixeltunnel. Heute möchte ich einen etwas anderen Ansatz zeigen.

Als Beispiel nehmen wir zufallsgenerierte Zahlen, die in mehreren Spalten über den Bildschirm scrollen. Dafür brauchen wir lediglich ein Objekt und zwei Scripte.



Das Ganze funktioniert relativ einfach. Mit dem bisherigen Wissen hätten wir ein Objekt erzeugt, das eine Zahl beinhaltet und von oben nach unten scrollt. Wenn es unten ankommt, würde es wieder nach oben springen. In eine zweiten Objekt hätten wir per Array den Bildschirm mit Instanzen des ersten Objekts gefüllt.

Problem: Auf dem Bildschirm befinden sich hinderte von Instanzen (im vorliegenden Beispiel wären es 288) und das kostet viel Performance, da jede Instanz einen Haufen Variablen mitbringt, die wir überhaupt nicht brauchen.

Lösung: Wir steuern direkt die einzelnen Zahlen im Array an, ohne die vielen Instanzen zu benötigen. Die Schlüsselstelle besteht darin zu wissen, wie man eine Zahl, die außerhalb des Bildschirms ist,

wieder nach oben bekommt.

scr_numbers_create

Dieses Script erzeugt lediglich unser Array und generiert die Zufahlszahlen. Das Script wird später im Create-Event unseres Objekts aufgerufen. Dem Script sagen wir nur, wie viele Zahlen wir brauchen (argument0).

```
// Create the Array

var allNumbers = argument0;

randomize();

for(i = 0; i < allNumbers; i++)
{
 var value = floor(random(10));
 number[i] = value;
}
```

Wie wir in anderen Tutorials bereits gelernt haben, rufen wir den Zufallsgenerator auf. Anschließend lassen wir eine Schleife laufen, erzeugen jeweils einen Zufallswert und schreiben diesen in den Array.

scr_numbers_draw

Das zweite Script zeichnet die Zahlen und kommt somit folgerichtig in das Draw-Event. Die Parameter legen wir später im Create-Event an.

```
/// Draw the number scroller

var allNumbers = argument0;
var rows  = argument1 + 2;
var columns  = argument2;
var font  = argument3;
var textCol  = argument4;
var alpha  = argument5;
var distanceX = argument6;
var distanceY = argument7;
var scrollSpeed = argument8;

var count  = 0;

draw_set_font(font);
draw_set_alpha(alpha);
draw_set_color(textCol);

for(a = 0; a < columns; a++)
{
 for(i = 0; i < rows; i++)
 {
  // Calculate the Y position
  var posY = i * distanceY + scrollerY + distanceY;
```

```
  // If the number is outside the screen, jump up
  if (posY > (room_height + distanceY))
  {
   // The result is the negative value of PosY
   posY = -(room_height + distanceY * 2 - posY);
  }

  // Draw the number
  draw_text(distanceX / 2 + a * distanceX, posY, string(number[count]));

  // Counter
  if (count < allNumbers){count++;}
 }
}

if (scrollerY < room_height)
{
 scrollerY += scrollSpeed;
} else {
 scrollerY = 0;
}
```

Bis einschließlich Zeile 17 legen wir nur den Rahmen fest. Die Argumente werden in Variablen geschrieben, wir haben einen Counter (var count) und legen die Schrift samt Alpha fest. Anhand der Argumente kann man auch sehen, dass man später den Effekt gut anpassen kann. Die Anzahl von Spalten und Reihen ist ebenso von Außen steuerbar wir die Geschwindigkeit, der Abstand und das Aussehen.

Bevor wir auf den großen Block mit den zwei Schleifen schauen, blicken wir in Zeile 41. Hier wird das Scrollen abgefragt. Die Zahlen fallen von oben nach unten. So lange die Zahl sich innerhalb des Raums befindet, wird die Y-Verschiebung weiter auf addiert. Wenn die Position darüber hinaus geht, wird die Verschiebung auf 0 gesetzt. Das wirkt sich auf den ganzen Block an Zahlen aus. Wir scrollen also nicht jede einzelne Zahl, sondern das ganze Gitter.

Nun zu den for-Schleifen. Die Zahlen werden wie in einer Tabelle dargestellt, bestehen also aus Spalten und Reihen. Hierfür brauchen wir zwei Schleifen. In der ersten Schleife erzeugen wir die Spalten (columns) und in der zweiten die Reihen (rows). In Zeile 34 zeichnen wir die Zahlen. Hierfür braucht man X, Y und den String. Der String ergibt sich aus dem Array. X berechnen wir direkt beim Zeichnen mit *distanceX / 2 + a * distanceX*. *distanceX* ist eine Variable, die wir später festlegen. Die gibt die Abstände der Spalten an. Die Variable a kommt aus der ersten Schleife.

Etwas kniffliger ist es mit Y. *posY* müssen wir vorher berechnen. In Zeile 24 berechnen wir, wo Y sein müsste. Die Position ergibt sich aus der Variable *i* (zweite Schleife) *distanceY* und *scrollerY*. So weit, so einfach. Nun kommt der knifflige Teil. Wir fragen ab, ob die Zahl außerhalb des Bildschirms ist. Damit die Zahl am unteren Rand nicht einfach verschwindet sondern sanft scrollt, gibt es bei der Abfrage in Zeile 27 noch den Zusatz „+ *distanceY*“. Wenn *posY* unten raus scrollt, brauchen wir einen neuen Wert. Die Zahl muss nach oben springen. Die neue Position berechnen wir in Zeile 30. Schauen wir uns die Zeile etwas genauer an:

```
posY = -(room_height + distanceY * 2 - posY);
```

Ich habe es hier gleich verkürzter dargestellt als im Video (unten). *posY* haben wir bereits in Zeile 24 berechnet. Nehmen wir an, wir haben die letzte Zeile. Im Beispiel haben wir 16 Spalten und 16 Zeilen (definieren wir noch im Objekt). i wäre somit 16. *distanceY* berechnen wir auch im Objekt vor. Damit wir eine Zahl haben, nehmen wir die 48. Wenn scrollerY bei 0 ist, ist die posY nach der Berechnung von Zeile 24 **816**. Bei einer Raumhöhe von 768 Pixeln ist die Bedingung der Abfrage noch nicht ergeben. posY ist 816 und *room_height* + *distanceY* ist ebenfalls 816. Nun wird einen Step später *scrollerY* von einer 0 zu einer 2 (das ist unsere Geschwindigkeit). *posY* ist somit 818. Jetzt tragen wir diese Zahlen in die herausgehobene Zeile ein:

```
posY = -(768 + 48 * 2 - 818);
```

Das Ergebnis ist -46. Die Zahl springt somit von der Position 818 auf -46, landet also oben, außerhalb des Bildschirms und kann von da aus weiter runter scrollen.

In der letzten Zeile der Schleife wird lediglich der Counter hoch gezählt.

obj_falling_numbers

Die größte Hürde ist gemeistert. Jetzt brauchen wir nur noch das Objekt, welches wir in den Raum setzen.

Event-Create

```
// Config the Matrix
font  = fnt_numbers;
alpha  = 1;
textCol  = c_lime;
scrollSpeed = 2;
rows  = 16;
columns  = 16;
distanceX = floor(room_width / columns);
distanceY = floor(room_height / rows);
allNumbers = (rows + 2) * columns;

scrollerY = 0;

// Create the array
scr_numbers_create(allNumbers);
```

Zunächst legen wir die Variablen fest. Bei *rows* geben wir die Anzahl der Zeilen auf dem Bildschirm an. Tatsächlich aber brauchen wir zwei mehr, da wir oberhalb und unterhalb eine weitere Zeile haben. Die Abstände und die Anzahl aller Zahlen werden vorberechnet.

Event-Draw

Da wir alles über unser Script steuern, müssen wir dieses Script nur noch korrekt aufrufen:

```
scr_numbers_draw(allNumbers, rows, columns, font, textCol, alpha, distanceX, d
```

Übrigens: Der Effekt sieht mit folgenden Einstellungen noch etwas cooler aus, wird dadurch aber auch

(von der Real FPS her) langsamer:

```
rows  = 24;
columns  = 36;
```

Das war es auch schon. Der Effekt ist fertig und kann von Dir beliebig angepasst werden. Wenn Du das Ganze auch noch als Video sehen möchtest, bitte:

falling numbers

1 Datei(en) 38.80 KB
Download
Date Created
6. März 2017
Author
sven