

Skipper – Ein Spiel mit vielen Zufallselementen

Description

Im Übungsprojekt Skipper wenden wir wieder viele Dinge an, die wir in den letzten Tutorials gelernt haben. Der Fokus liegt heute darauf, mit möglichst vielen Zufälligkeiten ein dynamisches, aber dennoch faires Spiel zu erstellen.

Zu meinen ersten elektronischen Spielen zählten LCD-Spiele. Die meisten davon waren simple Geschicklichkeitsspiele, bei denen man zum Beispiel ein Fahrzeug über eine Strecke navigieren musste, während man mit einer konstanten Geschwindigkeit Autos überholen und dem Gegenverkehr ausweichen musste. Ein weiteres war ein Ruderboot, bei dem es die Aufgabe war, vom Himmel fallende Fallschirmspringer zu retten. Genau so ein Spiel machen wir im folgenden Tutorial.

Ganz unten befindet sich die Datei zum downloaden. Du hast somit die Möglichkeit, entweder das Tutorial nach zu basteln, oder Dir den Code direkt anzuschauen.

Das Spiel hat 5 Level und wird immer schwieriger. Diese 5 Level steuern wir über die globale Variable *global.level*.

Zufallszahlen

Das Thema hatten wir bereits in mehreren Artikeln. Im Artikel [Glück: Der Teufel des Game-Designs](#) schrieb ich, wie gut, aber auch wie schlecht Zufallselemente sein können. Das Tutorial [Arbeiten mit Zufallszahlen](#) zeigt die Grundlagen, wie man im GameMaker Zufallszahlen generiert. Tutorials wie [Schaltjahr berechnen](#) und [Aktienkurs zeichnen](#) zeigen konkrete Beispiele, wie Zufallszahlen helfen können.

Unser heutiges Ziel ist es, ein kleines Spiel zu bauen, bei dem wir möglichst viele sinnvolle Anwendungen für den Zufall finden.

Vorbereitung

Da wir ein komplettes, wenn auch kleines, Spiel machen, brauchen wir etwas mehr als sonst.

Sprites

- spr_boot -> ein Ruderboot mit 128x64 Pixel
- spr_jumper -> ein Fallschirmspringer mit 48x96 Pixel
- spr_sonne -> eine Sonne mit 128x128 Pixel
- spr_welle01 -> Wellen mit 32*32 Pixel
- spr_wolke_01 -> eine Wolke mit 64x64 Pixel
- spr_wolke_02 -> eine Wolke mit 128x32 Pixel
- spr_wolke_03 -> eine Wolke mit 128x128 Pixel

Alle Sprites müssen unter **Origin** auf *Center* gestellt werden. Beim Ruderboot solltest Du darauf achten, dass die Maske nicht zu groß wird. Wenn das ganze Boot eine Maske ist, ist das Spiel zu einfach und das einsammeln der Fallschirmspringer wirkt recht komisch.

Sounds

- snd_game_over
- snd_fail
- snd_jumperpickup
- snd_nextlevel

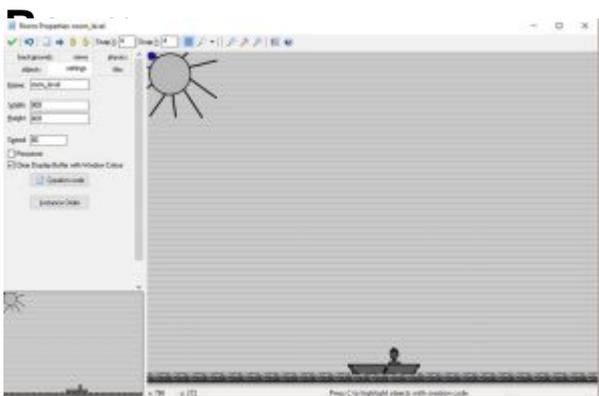
Da wir ein sehr simples Spiel machen, können alle Effekte auf Mono gestellt werden. Eine Musik habe ich nicht im Beispiel, der Code ist aber vorhanden, nur auskommentiert.

Hintergrund

- bg_streifen -> 32x32 Pixel Hintergrund, der im Raum gekachelt und animiert wird

Schriften

- fnt_lives -> Schriftart für Leben und Punkteanzeige. Hier ist eine Retro-Schrift angebracht, habe mich aber beim Projekt für Arial 16 entschieden, da diese Schrift auf jedem Rechner vorhanden sein sollte
- fnt_gameover -> auch hier wäre Retro besser, ist aber Arial 48. Damit wird am Ende das GameOver angezeigt



room_level in der Auflösung 800x600. Speed auf 60 stellen

Unter **backgrounds** wählen wir *bg_streifen* aus. **Tile Horizontal** und **Vertikal** werden angekreuzt. Die **horizontale Geschwindigkeit** lassen wir bei 0, vertikal stellen wir auf 1.

Objekte

Wir brauchen 6 Objekte:

- obj_boot -> das ist der Spieler -> Depth: 0
- obj_mechanik -> für die Spielmechanik -> Depth: -15
- obj_jumper -> die Fallschirmspringer -> Depth: 0
- obj_sonne -> Sonne -> Depth: 10
- obj_welle_01 -> die Wellen -> Depth: -10
- obj_wolke_01 -> das sind unsere Wolken -> Depth: 5

Du musst allen Objekten die entsprechenden Sprites zuweisen. Bei den Wolken nimmst Du die erste Wolke, der Rest wird über den Code gesteuert. Die Sortierwerte sind wichtig, damit die Wellen vor dem Boot sind und die Wolken hinter den Fallschirmspringern.

Jetzt platzierst Du die Wellen unten im Raum, das Ruderboot unten in der Mitte, oben rechts oder links die Sonne und noch das Objekt *obj_mechanik* irgendwo im Raum. Anschließend kannst Du den Raum schließen. Wir arbeiten jetzt nur noch mit den Objekten.

obj_jumper

Event Create

```
// Je nach Level wird die Fallgeschwindigkeit ermittelt
switch (global.level)
{
    case 1: jspeed = 3; break;
    case 2: jspeed = 4; break;
    case 3: jspeed = 5; break;
    case 4: jspeed = 5; break;
    case 5: jspeed = 6; break;
}
```

```
// Der Fallschirmspringer fällt
motion_set(270, jspeed);
```

```
// Die Position des Fallschirmspringers wird in obj_mechanik in Alarm0 festgelegt
```

Auch wenn wir die Variable noch nicht definiert haben, fragen wir sie schon ab. Je nach Level gibt es eine andere Fallgeschwindigkeit der Springer. Die Geschwindigkeit wird mit einer [switch](#) ermittelt.

Event Step

```
// Wenn der Fallschirmspringer unter dem Bildschirmrand ist, also nicht
// aufgefangen wurde...
if (y > (room_height + 64))
{
```

```
lives -= 1; // Ein Leben wird abgezogen
// Das Sample wird abgespielt
audio_sound_gain(snd_fail, 1, 0);
audio_play_sound(snd_fail, 2, false);
instance_destroy(); // Die Instanz wird vernichtet
}

// Wenn das Spiel aus ist, werden alle verbleibenden Instanzen vernichtet
if (lives < 1)
{
    instance_destroy();
}
```

obj_boot

Wir legen hier die Steuerung fest und sagen, was passiert, wenn der Fallschirmspringer das Boot berührt.

Event Create

```
bspeed = 10; // Geschwindigkeit
```

Event Step

```
// Steuerung nach links und recht. Die 32 sagt aus, dass wir nicht zu weit über
if (keyboard_check(vk_left)) && (x > 32)
{
    x -= bspeed;
    y -= 0;
}

if (keyboard_check(vk_right)) && (x < (room_width - 32))
{
    x += bspeed;
    y += 0;
}

// Wenn wir keine Leben haben, werden wir vernichtet
if (lives < 1)
{
    instance_destroy();
}
```

Event Collision with obj_jumper

```
// Der Fallschirmspringer wird zerstört
with (other) instance_destroy();

// Hier gibt es, je nach Level, die Punkte
switch (global.level)
{
    case 1: score += 100; break;
```

```
    case 2: score += 200; break;
    case 3: score += 400; break;
    case 4: score += 800; break;
    case 5: score += 1600; break;
}

// Sample wird abgespielt
audio_sound_gain(snd_jumperpickup, 1, 0);
audio_play_sound(snd_jumperpickup, 2, false);
```

Auch für die Punkte benutzen wir eine switch.

obj_wolke_01

Event Create

```
// Sprite
snummer = random(9);

// Geschwindigkeit der Bewegung
wspeed = random_range(1, 3);

// Wolke bewegt sich von links nach rechts
motion_set(0, wspeed);
```

Hier haben wir gleich zwei Zufallselemente. Der Sprite wird ebenso zufällig bestimmt, wie die Geschwindigkeit der Wolke.

Event Step

```
// Wenn die Instanz rechts weg ist, wird sie vernichtet
if (x > (room_width + 128))
{
    instance_destroy();
}
```

Event Draw

```
// Je nach Zufall wird ein entsprechendes Sprite angezeigt. Die große
// Wolke am wenigsten.
if (snummer <= 4)
{
    draw_sprite(spr_wolke_01, -1, x, y);
} else if (snummer > 4) && (snummer < 8) {
    draw_sprite(spr_wolke_02, -1, x, y);
} else {
    draw_sprite(spr_wolke_03, -1, x, y);
}
```

Hier sieht man schön, wie man Wahrscheinlichkeiten beeinflussen kann. Wir haben im **Create-Event** eine Zahl zwischen 0 und 9 erzeugt. Liegt die Zahl zwischen 0 und 4, wird die erste Wolke angezeigt. Liegt sie zwischen 4 und 8, kommt Wolke 2. Wenn die Zahl zwischen 8 und 9 liegt, kommt die dritte

Wolke.

obj_mechanik

Event Create

```

/* Eigentlich ist es besser, solche Startparameter wie im ersten Block
in ein Script auszulagern, vor allem alle globalen Variablen, damit
man einen guten Überblick hat. Bei diesem kleinen Beispiel spare ich mir
das.
*/
gameover = false; // Variable wird im Step-Event gebraucht
global.level = 1; // Startlevel

lives = 5; // Anzahl Leben bei Spielbeginn
score = 0; // Anzahl Punkte bei Spielbeginn
alarm[0] = 1 * room_speed; // Hier werden die Fallschirmspringer generiert
alarm[1] = 60 * room_speed; // Hier steigt die Levelstufe jede Minute an

// Wolkengenerator am Anfang. Auch das könnte man in ein Script auslagern.
anzahl = random_range(4, 8); // Am Anfang gibt es mindestens vier, aber höchstens
// Die Wolken werden generiert und positioniert. Aussehen und Geschwindigkeit
// wird im Wolkenobjekt bestimmt
for (i = 0; i < anzahl; i++)
{
    xx = random(room_width);
    yy = random(128) + 76; // Eine Wolke wird auf einer Höhe zwischen 76 und 200
    instance_create(xx, yy, obj_wolke_01);
}

alarm[2] = 1.5 * room_speed; // Hier werden weitere Wolken generiert

// Musik startet (ist hier nicht eingebaut)

//audio_sound_gain(snd_music, 1, 0);
//audio_play_sound(snd_music, 2, 1);

/* Anmerkung: es bietet sich an, für die Lautstärke Variablen zu verwenden. Meistens
für Musik eine Variable und für Soundeffekte eine andere. Dann kann man den Wert der
Einstellungen steuern. Bei diesem kleinen Beispiel habe ich mir das gespart.
*/

```

Wenn Du noch Probleme mit for-Schleifen hast, solltest Du Dir das [entsprechende Tutorial](#) anschauen.

Event Alarm0

```

if (lives > 0)
{
    // Hier regeln wir die Zeit, in welcher der Alarm aufgerufen wird.
    // Je kürzer die Zeit, umso mehr Fallschirmspringer werden generiert.
    switch (global.level)

```

```
{
    case 1: alarmspeed = 2.00 * room_speed; break;
    case 2: alarmspeed = 1.80 * room_speed; break;
    case 3: alarmspeed = 1.50 * room_speed; break;
    case 4: alarmspeed = 1.30 * room_speed; break;
    case 5: alarmspeed = 1.15 * room_speed; break;
}

// Eine zufällige horizontale Position wird gewählt.
// Dabei wollen wir nicht zu nah an die Ränder kommen
xx = random_range(32, room_width - 32);

instance_create(xx, -64, obj_jumper); // Fallschirmspringer wird erstellt

alarm[0] = alarmspeed; // Alarm wird neu gestartet
}
```

Event Alarm1

```
// Jede Minute geht es ein Level hoch, bis Level 5 erreicht wurde.
if (global.level < 5) && (lives > 0)
{
    global.level += 1;
    audio_sound_gain(snd_nextlevel, 1, 0);
    audio_play_sound(snd_nextlevel, 2, false);
    alarm[1] = 60 * room_speed;
}
```

Event Alarm2

```
wolkenalarm = random_range(2 * room_speed, 4 * room_speed);

yy = random(128) + 76; // Eine Wolke wird auf einer Höhe zwischen 76 und 204 P
instance_create(-128, yy, obj_wolke_01);

alarm[2] = wolkenalarm;
```

Die Wolken werden auch zeitlich zufällig erzeugt.

Event Alarm3

```
//Game Over Sample

audio_sound_gain(snd_game_over, 1, 0);
audio_play_sound(snd_game_over, 2, false);
```

Hier dient der Alarm nur als Code-Container.

Event Step

```
if (!gameover) && (lives < 1)
{
    alarm[3] = 1;
```

```

    gameover = true;
}

```

Wenn wir weniger als 1 Leben haben, wird auf gameover geschaltet und der Sound in Alarm3 ertönt.

Event Draw

```

// Rahmen
rdicke = 10; // Dicke des Rahmens
draw_set_color(c_black);

draw_line_width(0, room_height - (rdicke / 2), room_width, room_height - (rdicke / 2), rdicke); // Unten
draw_line_width(0, 0, room_width, 0, rdicke); // Oben
draw_line_width(-2 + (rdicke / 2), 0, -2 + (rdicke / 2), room_height, rdicke); // Links
draw_line_width(room_width - (rdicke / 2), 0, room_width - (rdicke / 2), room_height, rdicke); // Rechts

if (lives > 0)
{
    // Leben, Punkte und Level

    draw_set_color(c_black);
    draw_set_font(fnt_lives);
    draw_set_valign(fa_middle);
    draw_set_halign(fa_center);

    draw_text(room_width - 32, 32, "" + string(lives));
    draw_text(200, 32, "" + string(score));
    draw_text(room_width / 2 + 50, 32, "Level: " + string(global.level));
} else {
    draw_set_color(c_black);
    draw_set_font(fnt_gameover);
    draw_set_valign(fa_middle);
    draw_set_halign(fa_center);
    draw_text(room_width / 2, room_height / 2, "GAME OVER");
    draw_set_font(fnt_lives);
    draw_text(room_width / 2, room_height / 2 + 100, "SCORE: " + string(score));
}

```

Hier zeichnen wir Punktestand, Leben oder, wenn das Spiel vorbei ist, den entsprechenden Bildschirm.

Event release Escape

```
game_end();
```

Event release R-key

```
game_restart();
```

Fertig ist das kleine Spiel. Hier kannst Du Dir das ganze Projekt downloaden:



[Skipper](#)

1 Datei(en) 464.82 KB

[Download](#)

Date Created

21. November 2016

Author

sven