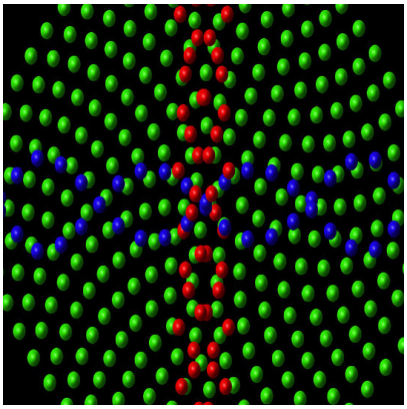


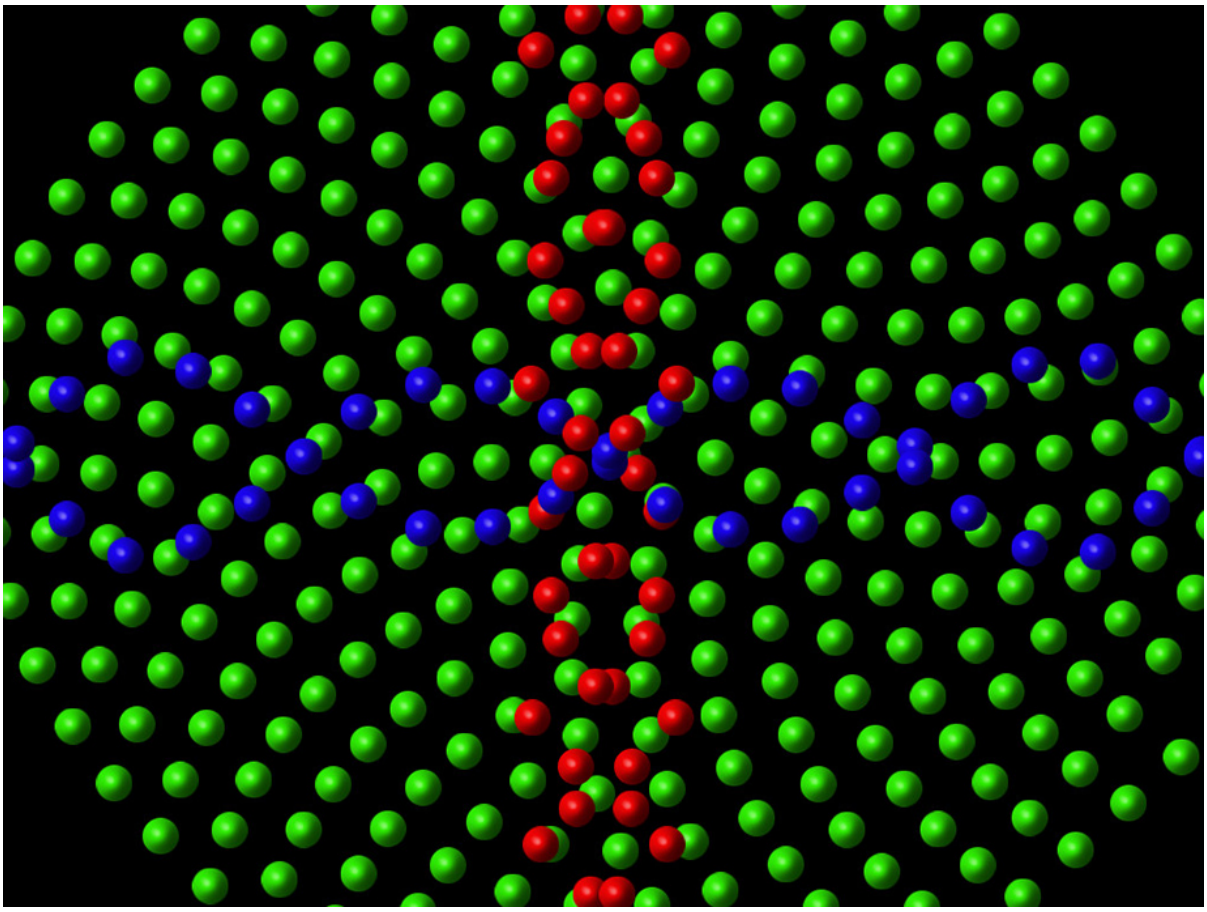
Vectorballs auf Pfaden

Eine der vielen praktischen Hilfen im GameMaker sind Pfade. Mit ihnen lassen sich beispielsweise Gegner durch ein Level bewegen. Mit dem folgenden kleinen Grafikeffekt kann man die Pfade sehr gut kennenlernen.

Einen Pfad haben wir bereits beim Pixeltunnel benutzt. Bei den Vectorballs möchte ich aber gezielt auf sie eingehen.

Was sind Vectorballs?

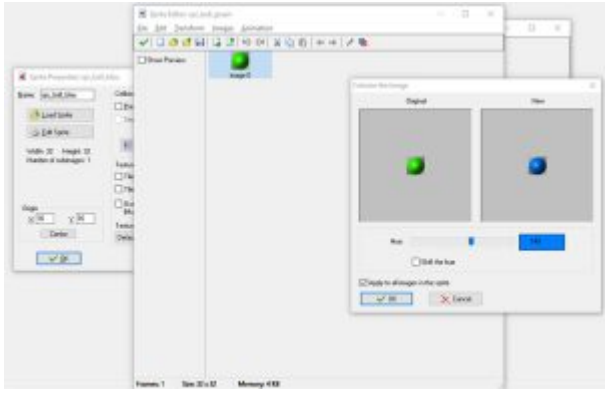
Wie so viele der Effekte, die ich hier vorstelle, ist auch dieser sehr alt, vielleicht sogar, abgesehen von Textscrollern, der älteste. Im Prinzip ist das nur ein rundes Sprite, welches koordiniert über den Bildschirm bewegt wird. Da nur ein Sprite keinen Spaß macht, sieht der Effekt erst dann richtig gut aus, wenn man viele davon verwendet. Am Ende des Tutorials wird unser Effekt so aussehen:



Wozu brauche ich Pfade?

So banal es klingt, ist es doch eine sehr gute Frage. Im Prinzip kann man im GameMaker sehr gut ohne Pfade auskommen, schließlich kann man Bewegungen auch programmieren. Doch in einigen Fällen ist das sehr komplex. Außerdem kann man gerade Pfade sehr gut bei komplexen Bewegungen verwenden. Es gibt dafür unzählige Beispiele. Ein Gegner, der ein bestimmtes Bewegungsmuster hat, aber auch Grafikeffekte wie Partikel können Pfaden folgen, wenn Beispielsweise der Spieler eine Kiste geöffnet hat und die Partikel schlängelnd bis zum Bildschirmrand tanzen sollen. Auch bei Menüs, die ganz schnell in den Raum rein kommen sollen, können Pfade praktisch sein.

Vorbereitung

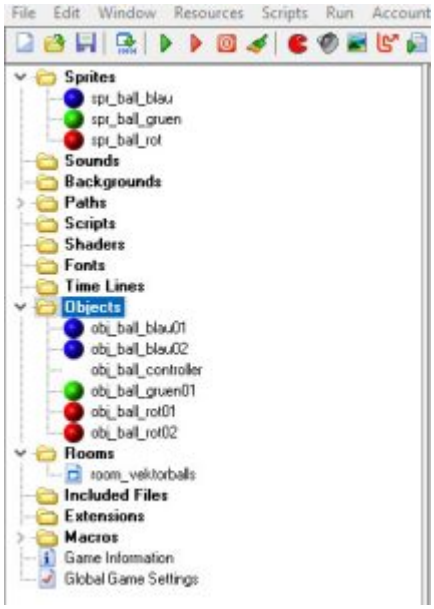


Sprites färben.

Legen wir endlich los. Als Ressource brauchst Du einen Sprite. Es muss nicht unbedingt ein Kreis sein, Du kannst auch einen Stern oder sonst was nehmen. Es sollte eine Auflösung von 32×32 Pixel haben. Lade es in GameMaker und nenne es *spr_ball_blau*, wenn es, wie bei mir, ein blauer Ball ist. Dupliziere es zwei Mal und benenne es entsprechend. Mein zweiter heißt *spr_ball_gruen* und mein dritter *spr_ball_rot*. Wenn Du nur eine Farbe hast, kannst Du die Sprites im Editor umfärben. Klick dafür auf **Edit Sprite**, dann auf **Images** (im oberen Menü) und hier auf **Colorize**. Mit dem Schieberegler kannst Du nun eine gewünschte Farbe einstellen. Mit **OK** wird der Sprite umgefärbt.

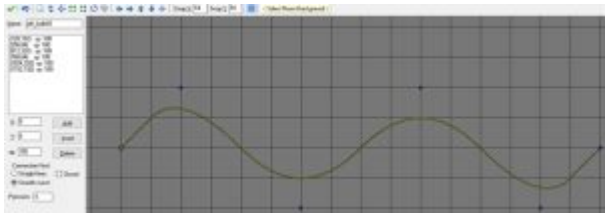
Als nächstes brauchst Du einen Raum. Ich habe bei mir eine Auflösung von 1024×768 eingestellt und wie immer 60 Frames bei **Speed**. Ein schwarzer Hintergrund eignet sich auch am besten.

Jetzt brauchen wir 6 Objekte. Benenne sie analog zum Screenshot *obj_ball_blau01*, *obj_ball_blau02*, *obj_ball_controller*, *obj_ball_gruen01*, *obj_ball_rot01* und *obj_ball_rot02*. Wenn Du schon die Sprites anders benannt hast, dann passe die Namen der Objekte an. Den Objekten, bei denen ich „ball“ im Namen habe, weist Du die entsprechende Sprites zu. Unterhalb des Namens, beim Objekt, kann man den entsprechenden Sprite einstellen. Jetzt musst Du nur noch das Objekt *obj_ball_controller* in den Raum setzen und die Vorbereitung ist abgeschlossen.



Alle benötigten Ressourcen.

Der erste Pfad



Der erste Pfad.

Nun erstelle einen neuen Pfad. Das kannst Du entweder in der oberen Menüleiste tun, oder wenn Du mit rechts auf den Ordner **Paths** und dann auf **Create Path** klickst. Nun befindest Du Dich in einem grauen Fenster. Hier kannst Du Pfade zeichnen, indem Du die einzelnen Punkte eines Pfades anklickst. Damit die Punkte nicht so nah beieinander liegen und Du dich besser orientieren kannst, kannst Du das Raster auf 64×64 stellen. Klicke auf **Smooth Curve** und deaktiviere **Closed**, da wir keinen geschlossenen Pfad haben möchten. Den haben wir ja bereits beim Pixeltunnel kennengelernt.

Benenne den Pfad in *pth_balls01* um. Wie mein Pfad aussieht, zeigt Dir mein Screenshot. Mache etwas ähnliches, mit ein paar Punkten. Wichtig ist, dass der erste und der letzte Punkt auf der gleichen Höhe sind.

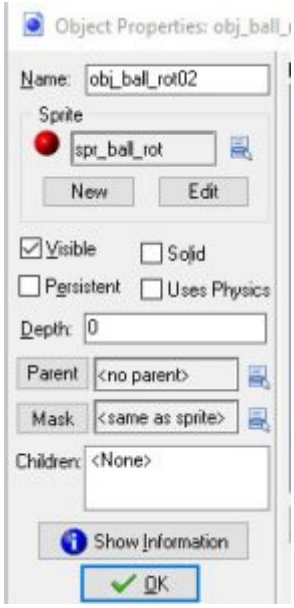
Wenn Du fertig bist, kannst Du das Fenster mit einen Klick auf das grüne Häkchen schließen.

obj_ball_blau01

Im Ball-Objekt brauchen wir zwei Events.

Event Create

```
path_start(pth_balls01, 4, path_action_continue, 0);
```



Einstellungen am Objekt.

Mit diesem Befehl binden wir den Ball an den Pfad. *pth_balls01* ist der Pfad, den wir eben erstellt haben. 4 ist die Geschwindigkeit und *path_action_continue* gibt an, was mit dem Objekt passiert, wenn es das Ende des Pfades erreicht hat. Hier gibt es mehrere Möglichkeiten:

- path_action_stop = Das Objekt stoppt am Ende des Pfades
- path_action_restart = Das Objekt springt an den Start zurück und beginnt die Bewegung erneut
- path_action_continue = Das Objekt beginnt den Pfad neu, aber von der aktuellen Position aus
- path_action_reverse = Läuft den Pfad rückwärts

Du kannst Dir sicher vorstellen, dass es für jede Möglichkeit nützliche Anwendungen in Spielen gibt. Wir haben uns für die dritte Möglichkeit entschieden. Deshalb war es auch wichtig, dass sich Anfangs- und Endpunkt auf der selben Höhe befinden.

Die 0 am Ende gibt an, ob die Koordinaten absolut oder relativ sein sollen. 0 steht für relativ, 1 für absolut. Wir wollen relative Koordinaten, weil wir dadurch die Bälle beliebig im Raum platzieren können.

Event Step

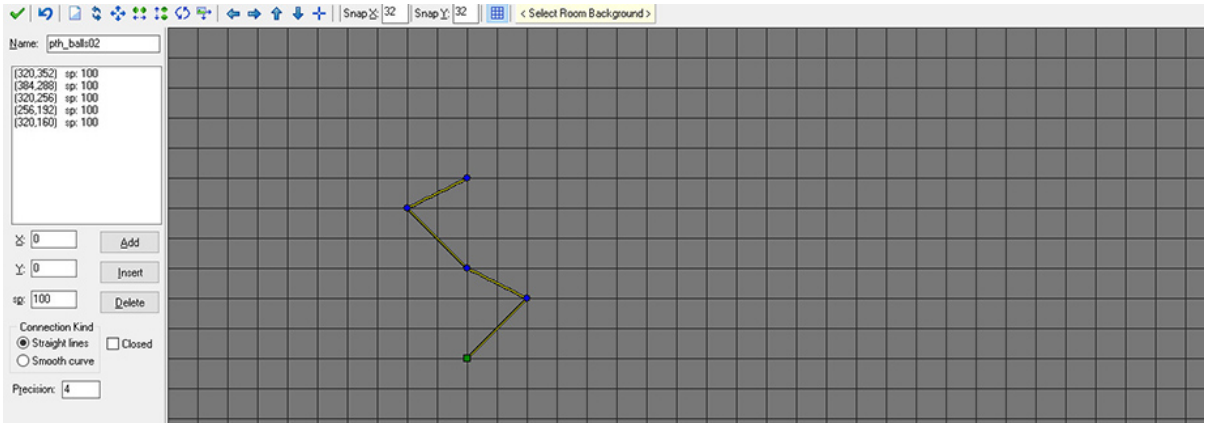
```
if (x < -64)|| (x > room_width + 64) || (y < -64)|| (y > room_height + 64)
{
    instance_destroy();
}
```

Hiermit sagen wir nur, dass das Objekt vernichtet werden soll, wenn es außerhalb vom Raum ist. Die Toleranz von 64 existiert nur, damit wir den Start des Objekts etwas Außerhalb platzieren können.

Du kannst das Objekt einmal im Raum platzieren und testen, was passiert. Es wird sich dem Pfad entlang schlängeln und sich vernichten, wenn er nicht mehr sichtbar ist. Zugegeben, das ist noch recht langweilig, aber es wird gleich interessanter.

Der zweite Ball

Erstelle erst einmal einen zweiten Pfad, wie auf dem Screenshot zu sehen ist, und nenne ihn *pth_balls02*. Anschließend kannst Du den Code aus *obj_ball_blau01* in *obj_ball_rot01* kopieren. Im **Create-Event** musst Du lediglich den Pfadnamen anpassen und dort *pth_balls02* rein schreiben. Jetzt wollen wir mal beide Animationen auf den Bildschirm zaubern, aber gleich mit mehreren Bällen. Bevor wir uns darum kümmern, schau bitte nach, dass sich im Raum nur noch das Objekt *obj_ball_controller* befindet.



obj_ball_controller

Wir brauchen hier ebenfalls nur zwei Events. Wenn Du möchtest, kannst Du noch ein drittes machen, in dem Du den Effekt mit der Esc-Taste abbrechen kannst.

Event Create

Hier reicht eine kurze Zeile:

```
alarm[0] = 1;
```

Wir rufen nun den Alarm auf.

Event Alarm 0

```
instance_create(0, room_height / 2, obj_ball_blau01);
instance_create(room_width / 2, room_height, obj_ball_rot01);

alarm[0] = 0.25 * room_speed;
```

Jetzt starte den Effekt und schau Dir an, was passiert ist.

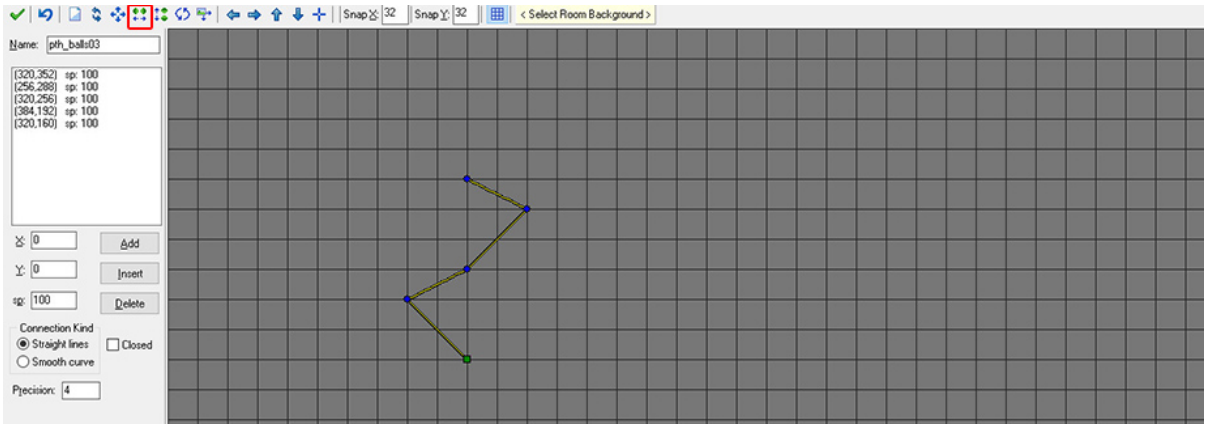
Der blaue Ball schlängelt sich von links nach rechts durch den Bildschirm, der rote von unten nach oben. Dabei kommt jede viertel Sekunde ein neuer Ball, weil wir den Alarm immer wieder aufrufen.

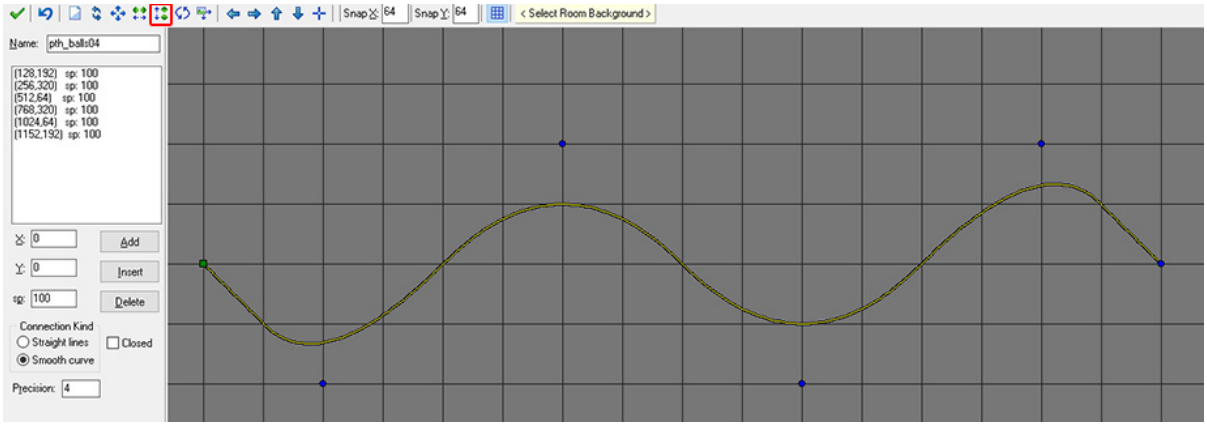
Das ist zwar schon ganz nett, aber die Idee liegt nahe, jeweils einen zweiten Pfad zu erzeugen, der in

der Horizontalen, bzw. Vertikalen genau entgegengesetzt verläuft. So würde eine Art Verzwirbelung entstehen.

Pfade duplizieren und spiegeln

Dupliziere den zweiten Pfad und nenne ihn *pth_balls03*. Dann duplizierst Du den ersten Pfad und nennst diesen *pth_balls04*. Jetzt haben wir jeden Pfad doppelt und müssen diese nur noch spiegeln. Praktischerweise bietet uns der GameMaker diese Option in der oberen Leiste des Pfades. Klick beim *pth_balls03* auf **Mirror the path horizontally** und bei *pth_balls04* auf **Flip the path vertically**.





Jetzt kopierst Du in die Objekte *obj_ball_blau02* und *obj_ball_rot02* den Code der beiden anderen Objekte und passt den Pfadnamen an.

Zuletzt gehst Du in den **Alarm 0 Event** von *obj_ball_controller* und fügst zwei weitere Zeilen hinzu, damit der Code so aussieht:

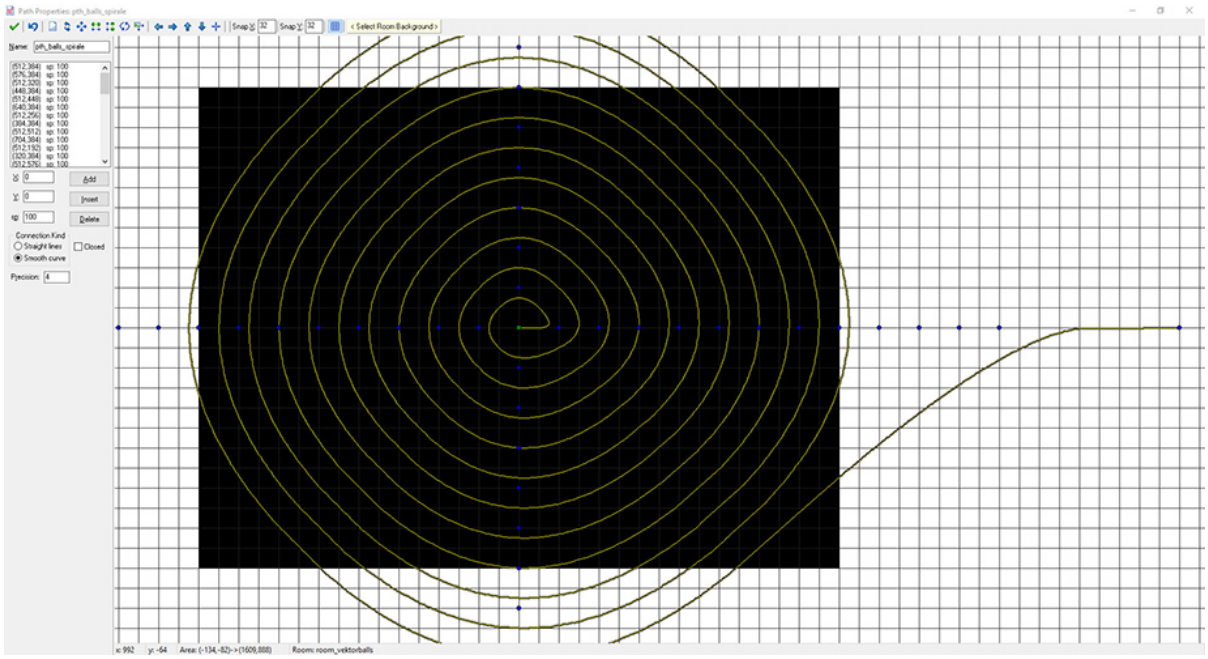
```
instance_create(0, room_height / 2, obj_ball_blau01);
instance_create(0, room_height / 2, obj_ball_blau02);
instance_create(room_width / 2, room_height, obj_ball_rot01);
instance_create(room_width / 2, room_height, obj_ball_rot02);

alarm[0] = 0.25 * room_speed;
```

Das sieht jetzt schon richtig gut aus! Doch wir sind noch nicht fertig und wollen noch ein drittes Objekt auf einen eigenen Pfad bringen.

pth_balls_spirale

Wie Du schon auf dem Screenshot siehst, wird der Pfad etwas komplexer. Hier hilft es, den Hintergrund des Raumes zur Orientierung zu verwenden. Auch dabei hilft uns der GameMaker. Klicke auf **Select Room Background** und wähle hier den Raum aus. Wenn Du dein Mausrad gedrückt hältst, kannst Du den ganzen Path-Raum verschieben und so auch sehen, was sich außerhalb des Sichtbereiches befindet.



Der fertige, spiralförmige Pfad.

Wir wollen nun eine Spirale erzeugen, die in der Mitte beginnt und außerhalb des Raumes endet. Der Start sollte also genau in der Mitte des Raumes liegen. Bei einem Raum mit der Größe 1024×768 ist das bei den Koordinaten 512×384. Ich habe mein Raster auf 32×32 gestellt und bin nun immer gegen den Uhrzeigersinn zwei Felder vom Mittelpunkt weg. Der zweite Punkt landet also auf 3 Uhr, bei 576×384, der dritte bei 512×320, also auf 12 Uhr. Dann kommt mit 9 Uhr 448×384 und mit 6 Uhr 512×448. Auf 3 Uhr gehen wir wieder zwei Felder weiter raus, also auf 640×384 und so geht es immer weiter.

Die gelbe Linie zeigt den tatsächlichen Verlauf. Du siehst, dass diese Linie ziemlich eng verläuft und weit ab von den tatsächlichen Punkten. Das liegt an der Option **Smooth curve**.

Mein letzter Punkt liegt bei 1280×384. Das ist so weit außen, dass man ihn nicht mehr sehen kann. Von hier aus habe ich noch einen weiteren Punkt auf 1568×384 gesetzt. Der Ball fliegt also noch weiter nach rechts.

obj_ball_gruen01

Wir sind mit dem Pfad fertig. Editiere nun das Objekt *obj_ball_gruen01*.

Event Create

```
path_start(pth_balls_spirale, 4, 0, 0);
```

Der Unterschied ist, dass wir den Pfad nicht mehr fortsetzen, sondern stoppen. Letztlich spielt das hier keine große Rolle, da das Objekt nie am Ende ankommen wird.

Event Step

```
if (x > 1560)
{
    instance_destroy();
}
```

Wir rufen nur noch die x-Koordinate ab. Da das Objekt auf dem Pfad immer wieder über den Bildschirmrand geht, wäre ein anderer Code sinnlos. Hier zerstört sich das Objekt erst, wenn *x* größer ist als 1560.

Nun gehst Du in das Objekt obj_ball_controller und fügst vor dem erneuten Alarmaufruf im **Alarm 0 Event** folgende Zeile ein:

```
instance_create(room_width / 2, room_height / 2, obj_ball_gruen01);
```

Das war es schon. Nun hast Du einen Effekt, der dem auf dem oberen Screenshot gezeigten sehr nahe kommen dürfte.

Allerdings gibt es noch ein optisches Problem. Die grünen Bälle überlagern alles. Das liegt am Sortierwert. Man kann im GameMaker angeben, auf welcher Ebene eine Grafik liegen soll. Öffne noch einmal das Objekt *obj_ball_gruen01*. In der Eingabe bei **Depth** sollte eine 0 stehen. Wenn dieser Wert größer ist als 0, wird die Grafik weiter hinten angezeigt. Ist er kleiner, hat also einen negativen Wert, wird er weiter vorne angezeigt. Stelle den Wert auf 100 und schon sind alle grünen Bälle im Hintergrund.

Im Angehängten Beispiel habe ich noch eine FPS-Anzeige eingebaut, die Dir zeigt, wie die Geschwindigkeit mit jedem weiteren Objekt runter geht. Da die Objekte sich selbst zerstören, pendelt sich eine konstante Geschwindigkeit ein, sobald alle grünen Objekte erzeugt sind und sich der erste vernichtet. Bei mir sind es dann immerhin noch rund 100 FPS.

Vectorballs auf Pfaden

1 Datei(en) 206.47 KB
Download
Date Created
26/10/2016
Author
sven