Objekte bewegen

Description

In GameMaker gibt es mindestens drei verschiedene Ansätze, um Objekte, egal ob Dekoration, Gegner oder den Player, zu bewegen. Heute schauen wir uns diese einmal genauer an.

Abgesehen von den Pfaden, die wir uns bereits angesehen haben, gibt es drei Möglichkeiten, ein Objekt bewusst zu steuern.

x und y Koordinate

Mit dem Koordinatensystem kann man im GameMaker sehr viel anfangen. So kann man jedes beliebige Objekt an einen gewünschten Punkt im Raum oder außerhalb des Raumes platzieren, indem man zum Beispiel im **Create-Event** eingibt:

```
x = 100;
y = 100;
```

Nachdem das Objekt, bzw. eine Instanz des Objekts erzeugt wurde, springt es an die Position 100 / 100. Man kann dieses System auch dazu benutzen, ein Objekt während des Spiels springen, oder bewegen zu lassen. Nehmen wir an, wir haben einen Raum und platzieren in der Mitte ein Objekt mit einem 32×32 großen Sprite. Im **Step-Event** geben wir folgenden Code ein:

```
x += 2;
y += 1;
```

Wenn wir das Spiel starten, bewegt sich das Objekt pro Step zwei Pixel nach rechts und ein Pixel nach unten. Es gleitet, sozusagen, nach rechts unten. Wenn wir bei beiden Werten eine 2 wählen, haben wir sogar einen korrekten 45°-Winkel.

Den Umgang mit den *x* und *y*-Werten muss man in GameMaker beherrschen, da sie immer gebraucht werden, nicht nur um Objekte zu positionieren. In anderen Tutorials verwendeten wir sie auch um Texte auf dem Bildschirm zu platzieren, für Abfragen und vieles mehr. Viele Funktionen fußen sogar darauf, man muss also konkrete oder relative Koordinatenwerte benutzen. Eines dieser Funktionen ist

folgende:

```
move_towards_point(100, 100, 4);
```

Wenn man das in den **Step-Event** schreibt, bewegt sich das Objekt mit der Geschwindigkeit von 4 in Richtung der Koordinaten 100, 100. Wenn es die erreicht hat, bleibt es stehen, oder zittert ein wenig, wenn es den Punkt nicht direkt erreicht. Das kann passieren, wenn der Abstand und die Geschwindigkeit nicht teilbar sind. Das springt das Objekt etwas drüber, wieder zurück und so weiter.

Zu *x* und *y* gibt es noch *xstart* und *ystart*. Diese Werte hält GameMaker für jede Instanz fest, sobald sie erstellt wird. In einigen Fällen ist es sinnvoll, ein Objekt an seinen Startpunkt zurück zu schicken, statt ihn zu zerstören. Zum Beispiel, wenn der Spieler seine Lebensenergie verliert, aber noch Leben hat. Dann schickt man ihn wie folgt an den Anfang zurück:

```
x = xstart;
y = ystart;
```

vspeed und hspeed

Für eine Bewegung lassen sich auch, statt *x* und *y*, die Befehle *vspeed* und *hspeed* benutzen. Ersteres steht für die vertikale, letzteres für die horizontale Geschwindigkeit. Der Wert, den man definiert, gibt die Pixel pro Step an.

```
hspeed = 2;
vspeed = 1;
```

Der Code bewirkt das Selbe wie die oben stehenden Zeilen. Wir bewegen das Objekt nach schräg unten. Das Ergebnis ist gleich, aber dennoch gibt es Unterschiede!

Fangen wir mit dem offensichtlichen an. Da wir von **speed** reden, ist sofort klar, dass es keine klare Koordinate, sondern eine Geschwindigkeit, somit eine Bewegung ist. Das springt gleich ins Auge, was bei der Programmierung hilfreich sein kann. Noch interessanter ist aber das Gleichzeichen. Wenn wir von Geschwindigkeit reden, meinen wir in den meisten Fällen eine konstante Geschwindigkeit. Im ersten Beispiel hingegen reden wir von Koordinaten, in diesem Fall sogar relativen Werten, weshalb wir mit **+=** statt **=** arbeiten.

Wenn wir den Player steuern, spielt es keine große Rolle, ob wir bei der Steuerung *x += 2;* oder *hspeed = 2;* schreiben. Einen Unterschied kann es machen, wenn es sich um Gegner handelt, deren Geschwindigkeit von Außen beeinflusst wird. Will man dies tun, muss man den Wert 2 bei *x += 2;* durch eine Variable ersetzen und diese von Außen ansteuern. Die Befehle *hspeed* und *vspeed* hingegen kann man im **Create-Event** starten und die Geschwindigkeit von Außen, also einem weiteren Objekt, manipulieren.

Mit den Befehlen *vpseed* und *hspeed* lassen sich darüber hinaus Beschleunigungen und Bremsmanöver leichter realisieren. Dazu ein Beispiel:

```
if (hspeed < 4)
{
    hspeed += 0.05;
```

```
}

vspeed = 2;
```

Der Code im **Step-Event** bewirkt, dass das Objekt so lange nach rechts beschleunigt wird, bis die Geschwindigkeit 4 erreicht hat. Die Beschleunigung erfolgt in 0.05er Schritten. Nach unten gibt es keine Beschleunigung, ließe sich aber mit dem selben Code bewältigen.

Um dies mit den *x* und *y*-Koordinaten zu realisieren, brauchen wir eine eigene Variable, die hochgezählt wird. Der Code würde so aussehen:

Event Create

```
xSpeed = 0;
```

Event Step

```
if (xSpeed < 4)
{
    xSpeed += 0.05;
}

x += xSpeed;
y += 1;
```

Um es sauber zu machen, sollte man die Variable *xSpeed* im **Create-Event** anlegen. Auch wenn es nur wenige Zeilen sind, wirkt der Code nicht mehr so sauber.

speed und direction

Die dritte Methode besteht daraus, die Bewegung in Geschwindigkeit und Richtung anzugeben. *speed* ist eine universelle Geschwindigkeit, sagt aber nicht aus, in welche Richtung die Reise hin geht. Dafür brauchen wir den Information *direction*, oder zu Deutsch, die Richtung. Die Richtung ist immer ein Wert zwischen 0 und 360. Dazu muss man wissen, dass 0 in GameMaker rechts ist. Oben ist 90, links 180 und unten 270. Es geht also gegen den Uhrzeigersinn im Kreis herum, beginnend auf 3 Uhr.

Unser Beispielobjekt haben wir nun zwei Mal mit den Geschwindigkeiten 2 und 1 bewegt. Wie würde der Code lauten, wenn wir *speed* und *direction* benutzen?

```
speed = 2;
direction = 337.5;
```

Der Wert für *direction* sieht komisch aus, ist aber recht einfach erklärt. Wie ich oben schrieb, haben wir bei 2 / 2 einen 45°-Winkel, 2 / 1 liegt also genau zwischen 0 und 45°, also bei 22,5°. Ein kompletter Kreis hat 360°. Wenn Wir die 22,5° abziehen, kommen wir auf die 337.5°.

direction wird eigentlich nur dann benutzt, wenn man ein Objekt in eine Richtung lenken will, die variabel sein kann. Angenommen, wir haben einen Spieler in einem Top-Down Spiel. Dieser hält eine Waffe in der Hand und kann sich per Maus in alle Richtungen drehen. Wenn er schießt, sollte die

Munition direkt aus dem Lauf der Waffe kommen und in die Richtung fliegen, welche die Waffe anzeigt. Genau für solche Fälle sind *speed* und *direction* sehr hilfreich.

Zusammenfassend kann man sagen, dass man mit *x* und *y* sehr viele einfache Bewegungen schnell realisieren kann. Sobald eine Bewegung, zum Beispiel durch Beschleunigung, komplexer wird, oder man die Geschwindigkeit von Außen steuern möchte, sollte man zu *hspeed* und *vspeed* greifen. Sobald es darum geht, ein Objekt in eine bestimmte Richtung zu bewegen, wie etwa eine Pistolenkugel, dann sind *speed* und *direction* die erste Wahl.

Date Created
7. Dezember 2016
Author
sven