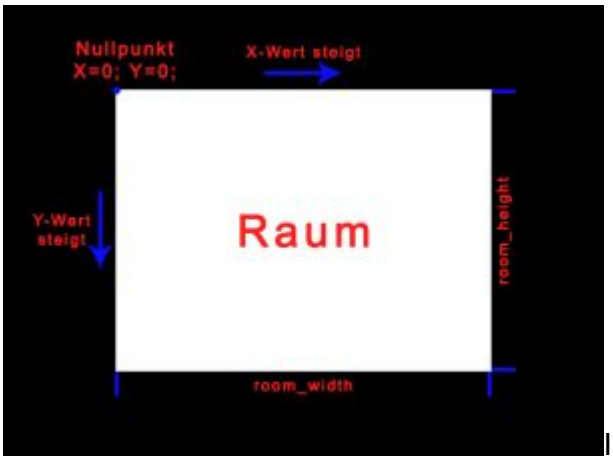
Koordinatensystem und Views in GMS2

Description

Mit Views lassen sich im GameMaker allerlei Dinge anstellen. Soll die Kamera den Spieler verfolgen oder auf das Geschehen zoomen? Alles kein Problem. Dieses Tutorial zeigt einige Kniffe und ganz nebenbei, wie man mit mehreren beweglichen Hintergründen arbeitet.



In der 2D-Welt gibt es, um sich zu orientieren, die X und die

Y-Koordinate. X beschreibt immer die horizontale, Y die vertikale Achse. Im GameMaker gibt es in jedem Raum einen Nullpunkt, der sich links oben in der Ecke befindet. Hier sind X und Y beide auf 0. Von diesem Punkt aus gesehen wird nach rechts der X-Wert positiv, nach links (außerhalb vom Raum) negativ. Y wird nach unten positiv und nach oben (außerhalb vom Raum) negativ. Ist einer der beiden Werte negativ, befindet sich die Koordinate somit auf jeden Fall außerhalb des Raums. Gleiches gilt für Werte, die größer sind als der Raum selbst.

In vielen Fällen ist der Raum genau so groß wie der Sichtbereich des Spielers. Spiele wie Tetris, Kartenspiele, Schach, Pong und andere gehören dazu. Auch bei Menüs gilt oft: Raumgröße = Sichtbereich. Das muss aber nicht so sein. Ein Level kann im GameMaker fast beliebig groß sein und wir können mit einer Kamera arbeiten, die uns nur einen bestimmten Bereich des Levels anzeigt.

Dazu gibt es im GameMaker 2 vorwiegend zwei Möglichkeiten. Wir können tatsächlich mit einer Kamera arbeiten oder wir beschränken uns rein auf die View-Funktionen. In diesem Tutorial werden wir zunächst nur mit den Views im Room arbeiten und dann auf die Kamera umsteigen.

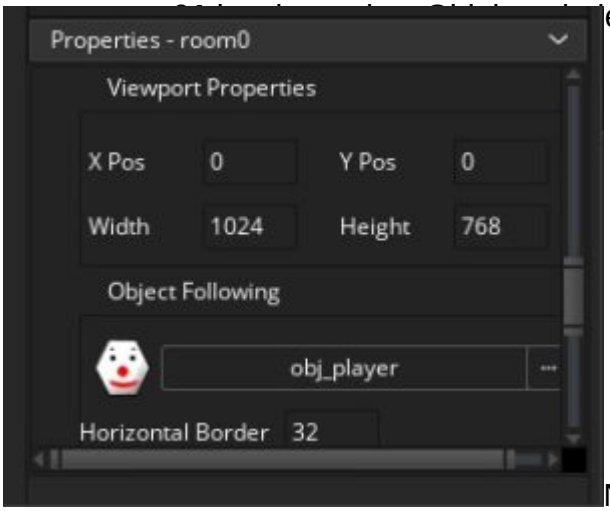
Ausgangssituation

Bevor wir richtig loslegen, brauchen wir ein paar Dinge.

spr_bg01 und **spr_bg02**. Das sind zwei Hintergrundbilder. Bei mir sind sie 256×256 Pixel groß und kachelbar. 02 ist dabei transparent, damit wir eine wunderbare Bewegung hinbekommen.

spr_player ist bei mir 32×32 groß und teiltransparent. Für das Beispiel spielt das aber eine untergeordnete Rolle.

spr_wall ist eine rote Wand mit einer Größe von 64×64 Pixel.



em wir der Testszene etwas Leben einhauchen können. Es

Nun gehen wir in den Raum. Wir behalten die

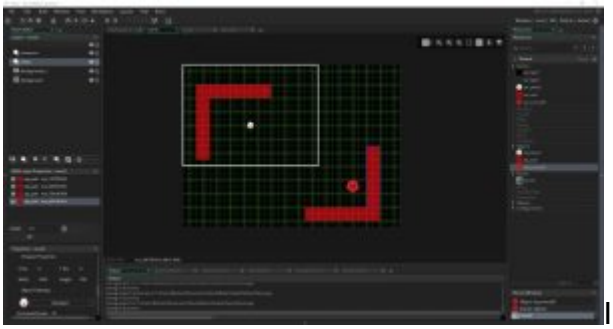
Standardgröße von 1024×768 und schaffen einen zweiten Background-Layer. Den Namen belassen wir einfach bei **Backgrounds_1**. Nun klicken wir zunächst auf **Background** und stellen dort **spr_bg01** als Bild ein. Wir klicken *Horizontal* und *Vertical Tile* an, damit das Bild gekachelt dargestellt wird. Wenn wir etwas weiter runter scrollen sehen wir, dass man noch die horizontale und vertikale Geschwindigkeit einstellen kann. Hier habe ich *Vertical Speed* auf 2 gestellt.

Jetzt gehen wir auf **Backgrounds_1** und wiederholen das Ganze mit **spr_bg02**. Die Geschwindigkeit sollte von vorherigen Layer abweichen, ich verwende im Beispiel *Horizontal Speed* 2 und *Vertikal Speed* -1.

Wenn wir das Spiel (ohne Objekt darin) nun starten, dann sehen wir zwei scrollende Layer. Das ist übrigens ein guter Test um kachelbare Grafiken zu überprüfen.

Zurück zum Spiel. Zunächst brauchen wir drei Objekte, nämlich **obj_player**, **obj_wall** und **obj_enemy01** mit den jeweiligen Sprites. Die Wand stellen wir noch aus *Solid*. Wir erstellen den Layer **Walls**

, der zwischen **Background_1** und **Instances** liegen sollte. Dort platzieren wir ein paar Wände. Im Layer Instances setzen wir den Spieler etwas links oben (bei mir 320, 288) und den Gegner etwas



Im Room-Editor gibt es links unten ein kleines Fenster mit der Beschriftung *Properties*. Hier können wir einen View einrichten. Wir sehen, dass uns hier 8 Viewports (0 bis 7) zur Verfügung stehen. Wir klicken auf *Enable Viewports* und anschließend auf **Viewport 0** und stellen die Sichtbarkeit mit einem Klick auf *Visible* ein. Unter **Camera Properties** geben wir bei der Auflösung 640×480 Pixel ein. Die **Viewport Properties** lassen wir bei 1024×768.

Wenn wir etwas weiter runter scrollen, können wir unter **Object Following** noch ein Objekt wählen. Das machen wir, indem wir hier **obj_player** wählen. Die Geschwindigkeiten lassen wir beide auf -1, *Horizontal Border* und *Vertical Border* stellen wir beide auf 96 ein.

Nach dem Spielstart merken wir, dass wir die Spielfigur zwar nicht bewegen können (ohne Code ist das schwierig) aber immerhin eine relativ große Spielfigur sehen. Das liegt daran, dass uns GameMaker nun den **Viewport 0** anzeigt, dessen Kamera wir auf 640×480 gestellt haben. Dieser Bereich wird nun auf die 1024×768 hoch gezogen.

Spieler bewegen

Damit sich die Kamera mit dem Spieler bewegt, brauchen wir ein paar Zeilen Code. Wir öffnen **obj_player** und erstellen einen Step-Event.

Event Step

```
/// @description Steuerung
if (keyboard_check(vk_down)) && (place_empty(x,y+4))
{
 y+=4;
}

if (keyboard_check(vk_up)) && (place_empty(x,y-4))
{
 y-=4;
}

if (keyboard_check(vk_left)) && (place_empty(x-4,y))
{
 x-=4;
}
```

```
if (keyboard_check(vk_right)) && (place_empty(x+4,y))
{
 x+=4;
}

// Spiel beenden
if (keyboard_check_released(vk_escape))
{
 game_end();
}
```

Ab jetzt können wir die Figur frei bewegen und haben zugleich eine kleine Kollisionserkennung. Mit Esc schließen wir das Spiel. Wenn wir das Projekt nun mit F5 starten, können wir durch das Level fliegen und die Kamera bewegt sich mit. Mit den 96 Pixeln beim Border haben wir eingestellt, wann die Kamera sich mit dem Spieler mitbewegen soll.

View per Code

In vielen Spielen reicht das schon aus. In zahlreichen Maze-Spielen und RPGs verfolgt die Kamera nur den Spieler. Allerdings ist dieses System nicht sehr flexibel. In Spielen kann es nützlich sein, wenn die Kamera auf einen Bossgegner zeigt, sie herein oder heraus zoomt (bei Dialogen) oder bei einem Erdbeben wackelt. Per Code sind wir viel flexibler und können so ziemlich alles machen, was der Gamedesign-Gott verboten hat.

Bevor wir mit dem Code beginnen, machen wir im Raum bei den **Viewports** alles rückgängig. Hierzu reicht es das Objekt auf *No Object* zu stellen, *Visible* und *Enable Viewports* auszuschalten.

In den meisten Fällen ist es sinnvoll, ein eigenes Kamera-Objekt zu erstellen. In diesem Tutorial sparen wir uns das und schreiben den Code in **obj_player**.

Event Create

```
/// @description Create View
view_visible[0] = true;  // Schaltet Viewport 0 ein
view_enabled = true;  // Macht den Viewport sichtbar
view_wport[0] = 640;  // Breite des Viewports
view_hport[0] = 480;  // Höhe des Viewports

// Wir stestellen eine Kamera und weisen ihr die Größe zu
view_camera[0] = camera_create();
camera_set_view_size(view_camera[0], view_wport[0], view_hport[0]);

camera_set_view_pos(view_camera[0], x-camera_get_view_width(view_camera[0]) /
```

Die letzte Zeile richtet die Kamera auf den Spieler aus. Diese nehemen wir und kopieren sie in die letzte Zeile unseres Step-Events.

Wenn wir das Spiel starten, merken wir, dass die Kamera unmittelbar dem Spieler folgt. Das fühlt sich etwas unnatürlich an, aber zumindest funktioniert alles. Wir haben einen View und können über *view_wport[0]*

und *view_hport[0]* bzw. *camera_set_view_size(view_camera[0], view_wport[0], view_hport[0]);* die Abmessungen des Views ändern. Das werden wir nun auch machen, indem wir einrichten, dass wir mit dem Mausrad raus- und reinzoomen können.

Event Mouse Wheel Up

```
/// @description Zoomt raus
if (view_wport[0] < room_width)
{
 view_wport[0] += 26;
 view_hport[0] += 20;

 camera_set_view_size(view_camera[0], view_wport[0], view_hport[0]);
}
```

Event Mouse Wheel Down

```
/// @description Zoomt rein
if (view_wport[0] > 256)
{
 view_wport[0] -= 26;
 view_hport[0] -= 20;

 camera_set_view_size(view_camera[0], view_wport[0], view_hport[0]);
}
```

Event Key Up – Space

```
/// @description Kamera in Ausgangsposition
view_wport[0] = 640;
view_hport[0] = 480;

camera_set_view_size(view_camera[0], view_wport[0], view_hport[0]);
```

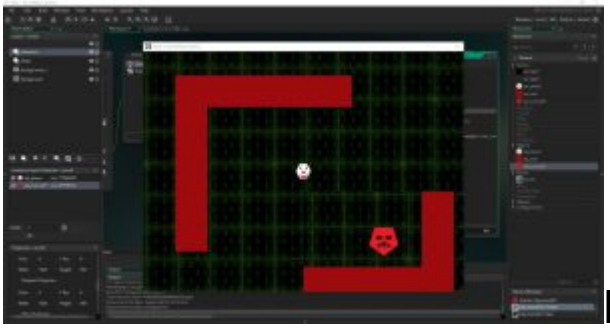
Wird das Spiel nun gestartet, können wir mit dem Mausrad bis zu den definierten Grenzen zoomen. Außerdem wird die Kamera in die Ausgangsposition versetzt sobald wir die Leertaste drücken.
Jetzt kümmern wir uns um den Border. In vielen Fällen wirkt die zentrale Positionierung nur dann komisch, wenn wir an den Levelrand geraten. Das heißt, wir brauchen eine Kameraeinstellung, die dem Spieler so lange folgt, bis ein Rand erreicht wird und dann stehen bleibt.
Hierzu ersetzen wir die letzte Zeile im Step-Event. Aus

```
camera_set_view_pos(view_camera[0], x-camera_get_view_width(view_camera[0]) /
```

machen wir

```
var cx = x-(camera_get_view_width(view_camera[0])/2);
var cy = y-(camera_get_view_height(view_camera[0])/2);
cx = clamp(cx, 0, room_width-camera_get_view_width(view_camera[0]));
cy = clamp(cy, 0, room_height-camera_get_view_height(view_camera[0]));
camera_set_view_pos(view_camera[0], cx, cy);
```

Das sieht auf den ersten Blick komplizierter aus als es ist. *clamp* gibt uns einen Wert, der zwischen zwei Werten liegt. In diesem Fall zwischen 0 und der Raumgrenze, die durch Raumbreite (bzw. Höhe) abzüglich der View_Ma8 definiert wird.



Eine zweite Kamera

Wir sind fast fertig. Der Gegner soll nicht ganz umsonst gewesen sein. Wir erstellen eine zweite Kamera und richten sie auf den Gegner aus. Damit sich die Kamera etwas bewegt, bewegen wir im Step-Event das Objekt ein wenig im Kreis.

Event-Create

```
/// @description Create View
view_visible[1] = true;  // Schaltet Viewport 0 ein
view_enabled = true;  // Macht den Viewport sichtbar
view_wport[1] = 320;  // Breite des Viewports
view_hport[1] = 200;  // Höhe des Viewports

// Wir stestellen eine Kamera und weisen ihr die Größe zu
view_camera[1] = camera_create_view(0, 0, view_wport[1], view_hport[1], 0, obj
camera_set_view_size(view_camera[1], view_wport[1], view_hport[1]);

camera_set_view_pos(view_camera[1], x-camera_get_view_width(view_camera[1]) /
```

Event Step

```
/// @description Follow Player
// Gegner im Kreis bewegen
y = y + (sin(direction) * 5);
x = x + (cos(direction) * 5);
direction += 0.2;

// Die Kamera folgt dem Gegner
camera_set_view_pos(view_camera[1], x-camera_get_view_width(view_camera[1]) /
```

Beides kommt natürlich in **obj_enemy01**. Nun haben wir eine zweite Kamera auf den Gegner ausgerichtet. Die Kamera hat eine Größe von 320×200 und klebt links oben in der Ecke. Das lässt sich durch die Variablen *view_xport[1]* und *view_yport[1]* verschieben. Mit diesen beiden Zeilen können wir die zweite Kamera rechts unten positionieren:

```
view_xport[1] = view_wport[0] - view_wport[1]; // x Position der Kamera
```

```
view_yport[1] = view_hport[0] - view_hport[1]; // y Position der Kamera
```

Fortgeschrittene Funktionen

Dieses Tutorial sollte vor allem eine Einführung sein. Mit Kameras lassen sich noch viele andere Dinge anstellen. So kann man beispielsweise den Neigungswinkel verändern oder die Kamera rotieren lassen. Hierzu ist es hilfreich, wenn man die Matrixfunktionen von GMS2 nutzt. Per **matrix_build** können wir eine z-Achse definieren und über die Skallierfunktionen das Bild sogar verzerren.

Auf das vorliegende Beispiel bezogen könnte man den Ausschnitt mit dem Gegner verkleinert darstellen, was natürlich wesentlich praktischer ist.

Bevor man sich allerdings da heran wagt, sollte man die hier vorgestellten Grundlagen beherrschen.

Date Created
7. Juli 2018
Author
sven