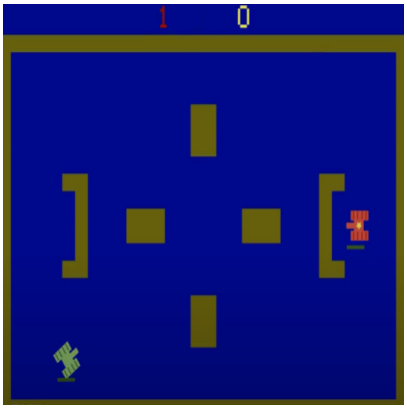

GM Tank Combat Teil 2

Description

Im ersten Teil haben wir bereits eine gute Ausgangslage geschaffen. Nun wird es Zeit den zweiten Spieler einzubauen.

Ziele

Der wichtigste Aspekt des zweiten Teils ist die praktische Arbeit mit Vererbungen. Wer das noch nie gemacht hat, sollte sich zunächst das entsprechende Tutorial durchlesen. Im Prinzip bauen wir heute alles ein, was wir für einen simplen Zweispielermodus brauchen. Einen zweiten Panzer mit anderen Eigenschaften und ein eigenes Geschoss. Gesteuert wird der zweite Panzer über W-A-S-D. Dazu bauen wir einen Punktezähler ein. Wenn ein Panzer stirbt, bekommt der andere Spieler einen Punkt und beide Panzer landen am Ausgangspunkt.

Die hier gezeigte Umsetzung ist nicht perfekt, aber zweckmäßig. Wer nur einen simplen Zweispielermodus haben will, braucht von der Mechanik nicht mehr. Im Prinzip kann man es noch durch eine Levelauswahl garnieren und ggf. einen Endwert bei den Punkten setzen. In dieser Serie möchten wir aber mehr erreichen, weshalb wir in Teil 3 einiges umbauen werden.

Der zweite Panzer

Zunächst erstellen wir eine zweite Grafik mit dem Namen **spr_tank02**. Der Panzer sollte gleich groß sein wie der erste, sich aber optisch etwas unterscheiden. Ich habe mich für dieses Design entschieden, ihr seid aber natürlich in der Gestaltung frei.

Anschließend erstellen wir **obj_tank02**. Als *Parent* geben wir **obj_tank01** an. Das Create-Event erstellen wir extra, alle anderen Events werden geerbt.

Event Create

```
image_angle = -180;
direction = image_angle;

// Tank-Speed
maxSpeed = 4;
minSpeed = -3;
rotationSpeed = 3;
speedUp = 0.6;
speedDown = 0.3;
brakeUp = 0.15;
brakeDown = 0.2;

// Bullet
canShoot = true;
bulletDelay = 80;
shooteTimer = bulletDelay;
bulletObject = obj_bullet02;
bulletDontKill = id;

// Steuerung
keyLeft = ord("A");
keyRight = ord("D");
keyUp = ord("W");
keyDown = ord("S");
keyShoot = vk_lcontrol;

player = 2;
```

Wir sehen ein paar Änderungen gegenüber dem ersten Panzer. In den ersten beiden Zeilen drehen wir das Bild auf -180. Die Panzer sollen sich beim Start beide anschauen und da wir die Grafik von Panzer 2 in die gleiche Richtung schauen lassen wie Panzer 1 müssen wir per Code etwas nachhelfen. Ganz unten haben wir noch die Zeile *player = 2;*. Die Variable brauchen wir, damit wir später dem Controller sagen, welcher Spieler gestorben ist.

Von den Eigenschaften her ist der zweite Panzer etwas langsamer. Die Nachteile gleicht er durch sein Geschoss aus.

Das zweite Geschoss

Wir erstellen zunächst die Grafik **spr_bullet02**. Das erste Geschoss hat eine Auflösung von 10×10 Pixel, das zweite machen wir 12×12 groß. Als Farbe nehme ich #e8e85c. Da das erste Bullet aus Teil 1 rot ist, können wir mit dem Gelb die beiden Geschosse sehr gut unterscheiden. Nun erstellen wir **obj_bullet02** und weisen als *Parent* **obj_bullet01** zu. Auch hier müssen wir nur das Create-Event individuell erstellen.

Event Create

```
liveTime = 150;
speed = 8;
bounceCount = 0;
bounceMax = 4;
```

```
dontKill = 0; // Variable wird vom Panzer übergeben
```

Im Vergleich zu **obj_bullet01** lebt es 50% länger, ist aber 20% langsamer. Neben der Lebensdauer und der Größe besteht ein weiterer Vorteil darin, dass es viermal statt nur zweimal von Wänden abprallen kann. Das macht **obj_bullet02** extrem gefährlich. Die Nachteile des Panzers sollten damit ausgeglichen sein, ihr könnt aber die Werte durch Tests gerne optimieren.

Modifikationen am ersten Panzer

Wir öffnen nun **obj_tank01**. im Create-Event fügen wir ganz unten eine weitere Zeile hinzu:

Event Create

```
player = 1;
```

Anschließend erstellen wir ein neues Event.

Event Destroy

```
/// @description Call Controller

obj_gamecontrolle.playerDead = player;
obj_gamecontrolle.xStart = xstart;
obj_gamecontrolle.yStart = ystart;
obj_gamecontrolle.alarm[1] = 1;
```

Das neue Event wird natürlich auch von **obj_tank02** geerbt. Hier teilen wir dem Controller mit, welcher Panzer gestorben ist, welche Startposition der Panzer hatte und starten Alarm1 des Controllers, den wir gleich noch anlegen werden.

obj_gamecontrolle

Bevor wir uns um das Objekt kümmern, brauchen wir noch eine Schrift. Ich habe mich für *Connection Serif* entschieden. Das ist zwar nicht ganz authentisch, sieht aber nach Retro aus und passt gut zum Spiel. Ihr könnte die Schriftart hier herunterladen. Die Schrift im Spiel nennen wir **fnt_score** und stellen die Schriftgröße auf 42. Wenn ihr euch für eine andere Schriftart entscheidet, müsst ihr die Schriftgröße ggf. anpassen.

Nun zu **obj_gamecontrolle**. Zunächst müssen wir das Create-Event erweitern. Hier der ganze Code:

Event Create

```
debug = true;

frames = round(fps_real);
alarm[0] = 1;

player1Score = 0;
player2Score = 0;
```

```
centerX = room_width / 2;

playerDead = 0;
xStart = 0;
yStart = 0;
```

Die oberen Zeilen sind identisch zu Teil 1. Nach Alarm0 definieren wir ein paar Variablen, von denen die meisten selbstredend sind. *centerX* verwende ich ganz gerne, damit ich im Draw-Event nicht laufend die Mitte bestimmen muss.

Event Alarm1

```
/// @description Player dead

// Alle Bullets sofort löschen
with(obj_bullet01){instance_destroy();}
with(obj_bullet02){instance_destroy();}

if (playerDead = 1)
{
 // Wenn Spieler 1 stirbt
 player2Score++;
 instance_create_layer(xStart, yStart, „Instances", obj_tank01);
 with(obj_tank02)
 {
  x = xstart;
  y = ystart;
  image_angle = -180;
 }

 with(obj_tank01)
 {
  image_angle = 0;
 }
} else if (playerDead = 2) {
 // Wenn Spieler 2 stirbt
 player1Score++;
 instance_create_layer(xStart, yStart, „Instances", obj_tank02);
 with(obj_tank01)
 {
  x = xstart;
  y = ystart;
  image_angle = 0;
 }

 with(obj_tank02)
 {
  image_angle = -180;
 }
}

playerDead = 0;
```

In Alarm1, der beim Tod eines Panzers ausgelöst wird, legen wir fest, was anschließend passiert.

Zunächst zerstören wir alle Bullets, die sich im Raum befinden können. Anschließend schauen wir ob Spieler 1 oder Spieler 2 gestorben ist. Die Punktzahl wird entsprechend erhöht, der zerstörte Panzer neu erstellt und beide Panzer werden auf die Ausgangsposition gesetzt und richtig herum gedreht. Am Ende setzen wir die Variable *playerDead* wieder auf 0. Im Prinzip wäre das nicht nötig, aber für den Fall, dass der Alarm mal versehentlich ausgelöst wird, stellen wir keinen Unsinn an.

Der Vorteil an dieser Herangehensweise ist, dass das in jedem Level funktioniert. Einzige Bedingung ist, dass sich die beiden Panzer anschauen müssen.

Event Draw GUI

Jetzt wollen wir die Punkte noch anzeigen. Der jeweilige Score bekommt die Farbe des Bullets. Spieler 1 wird links und Spieler 2 rechts angezeigt.

```
/// @description Debug
if (debug)
{
 draw_set_font(fnt_debug);
 draw_set_halign(fa_left);
 draw_set_valign(fa_middle);
 draw_set_alpha(1)
 draw_set_color(make_color_rgb(236, 236, 236));
 draw_text(34, room_height-15, „FPS: „ + string(frames));
}

draw_set_font(fnt_score);
draw_set_halign(fa_center);
draw_set_valign(fa_middle);
draw_set_color(make_color_rgb(136, 0, 0));
draw_text(centerX-100, 36, string(player1Score));
draw_set_color(make_color_rgb(232, 232, 92));
draw_text(centerX+100, 36, string(player2Score));
```

Aussichten

Das war es auch schon wieder. Im dritten Teil müssen wir, wie gesagt, einige Dinge umstellen. Das Problem ist derzeit, dass wir die beiden Panzer auf die jeweilige Steuerung fixiert haben. Das heißt: Spieler 1 kann nie Panzer 2 nehmen und umgekehrt. Noch schlimmer: Wenn wir eine KI einbauen wollen, müssten wir das mit extra Objekten lösen, also für jeden Panzer ein eigenes KI-Objekt. Bei zwei Panzern ist das nicht dramatisch, aber wenn wir einen ganzen Fuhrpark an Panzern haben möchten wird das Projekt von der Wartung her ein Desaster.

Bis der dritte Teil erscheint könnt ihr euch gerne darüber Gedanken machen, wie man dieses Problem möglichst flexibel löst und in die Kommentare schreiben. ?

Download

GM Tank Combat v. 0.0.2

1 Datei(en) 78 kb
Download
Date Created
12. Juli 2018
Author
sven