



Arbeiten mit Vererbung in GMS2

Description

Anfänger in der Programmierung können mit dem Begriff „Vererbung“ zunächst nicht viel anfangen. Im GameMaker kann man bei Objekten mit Vererbungen arbeiten, was extrem praktisch sein kann. Wie das funktioniert, zeigt das dieses Tutorial.

Einführung

Aus der objektorientierten Programmierung (OOP) kennt man das Konzept der Vererbung sehr gut. Seit langem (ziemlich genau seit 2001 mit GameMaker 4) besitzt der GameMaker ein ähnliches Feature, in dem man bestimmte Eigenschaften von einem Objekt auf ein anderes übertragen kann, aber wozu soll das gut sein?

In der Programmierung versucht man möglichst doppelten Code zu vermeiden. Das hat nicht zwingend etwas mit Faulheit zu tun (gut, ein bisschen schon) sondern viel mehr mit der Wartbarkeit. Je weniger unnütze Zeilen Code wir in einem Spiel haben, umso einfacher lässt er sich pflegen. Und ja, doppelter Code ist fast immer unnützlich.

Skripte erfüllen eine ganz ähnliche Funktion. Man schreibt Code in einem isolierten Bereich und kann diesen an vielen Stellen aufrufen. Das ist sehr praktisch, aber manchmal reicht das nicht aus.

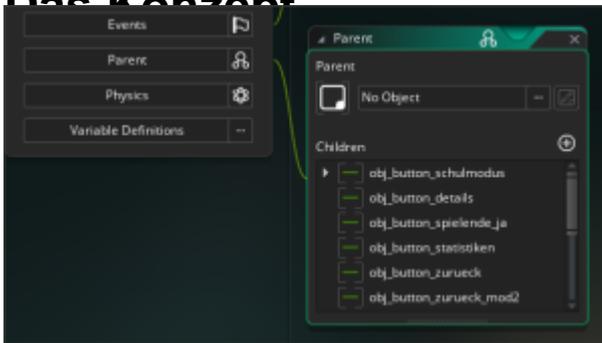
Gerade in Spielen kommt es oft vor, dass man viele Objekte mit sehr ähnlichen Eigenschaften hat. Mehrere Gegner, die sich nur bei Laufrichtung und Geschwindigkeit unterscheiden. Buttons in Spielen, die jeweils nur eine andere Variable an ein weiteres Objekt geben. Verschiedene Spielercharaktere, die sich nur in ein paar Attributen unterscheiden.

Ohne Vererbung würden wir für jedes Objekt immer wieder die gleichen Objekte kopieren und die wenigen Werte, die es unterscheidet, anpassen. Problem: Wir haben nicht nur hier doppelten Code, sondern blähen auch Objekte auf, die sich darauf beziehen.

Ein simples Beispiel: Unser Spiel hat ein Menü mit vielen Buttons. Der Mauszeiger wechselt immer, wenn er auf einem Button liegt, die Farbe. Im Code müssen wir entweder bei jedem Button prüfen, ob

die Maus drauf liegt, oder beim Mauszeiger jeden Button berücksichtigen. Bei zwei bis vier Buttons mag das nicht weiter stören, wenn es einmal zehn werden, nervt es gewaltig.

Das Konzept

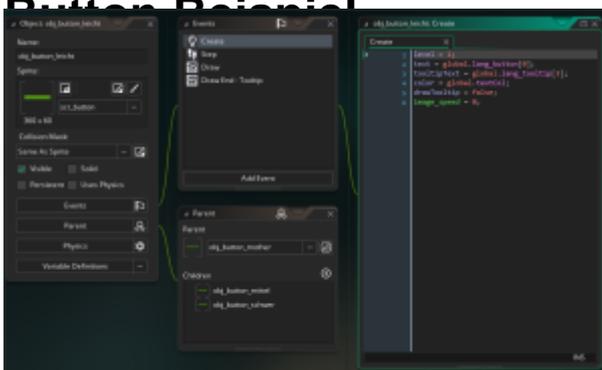


Das Konzept ist relativ einfach. Wir erstellen ein Elternobjekt

und weisen diesem alle Eigenschaften zu, die es braucht. Wenn wir nun ein zweites Objekt mit ähnlichen Eigenschaften brauchen, dann müssen wir zunächst auf *Parent* klicken und im neuen Fenster das Mutterobjekt angeben. Nun erbt das Kind (*Children*) alle Eigenschaften, also auch den Code aus allen Events von der Mutter.

Nun ist es so, dass unser neues Objekt andere Eigenschaften hat als das Mutterobjekt, sonst wäre es das Gleiche. Wir können im Kindobjekt die Events neu anlegen, neue Events erstellen bzw. Code der Mutter abändern. Wichtig: Die Vererbung passiert Eventweise. Wenn ich ein eigenständiges Event, z. B. **Create**, anlege, dann verliere ich den Create-Code der Mutter. Da wir möglichst viel erben wollen (doppelten Code vermeiden) ist es sinnvoll, alle individuellen Parameter im Create-Event auszulagern. Das heißt, dass ich alles, was ein Objekt individuell macht, über Create steure und der Rest vererbt wird.

Buttons Beispiel



Bei Buttons gehe ich so vor, dass ich zunächst ein Objekt

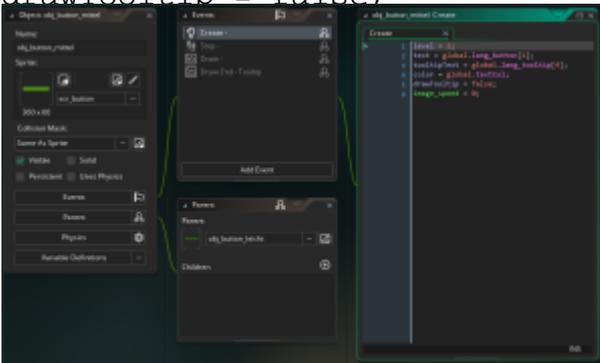
mit dem Namen *obj_button_mother* erstelle, das die Mutter aller Buttons ist, sonst aber keine Eigenschaften aufweist. Mein Mauszeiger bezieht sich dann immer auf dieses eine Objekt. Obwohl das Objekt im ganzen Spiel nicht auftaucht, funktioniert die Vererbung und ich muss beim Mauszeiger nur die Kollision mit dem Mutterobjekt abfragen.

Nun gibt es Buttons, die ähnlich sind. Etwa wenn der Schwierigkeitsgrad ausgewählt wird. *obj_button_leicht* wäre das Mutterobjekt, *obj_button_mittel* und *obj_button_schwer* die Kinder. Das Create-Event der Mutter könnte so aussehen:

```
level = 1;  
text = global.lang_button[0];  
tooltipText = global.lang_tooltip[3];  
color = global.textCol;  
drawTooltip = false;  
image_speed = 0;
```

Neben dem Create-Event gibt es noch Step, Draw und DrawEnd für die Tooltips. *obj_button_mittel* ist das Kind von *obj_button_leicht* und hat nur ein eigenes Create-Event, der Rest wird vererbt.

```
level = 2;  
text = global.lang_button[1];  
tooltipText = global.lang_tooltip[4];  
color = global.textCol;  
drawTooltip = false;
```



Die ersten drei Zeilen sind die individuellen Eigenschaften

von *obj_button_mittel*. Der Rest, auch alle Events, ist identisch.

Ein Objekt kann nur eine Mutter haben, eine Mutter aber mehrere Kinder. Das heißt, dass *obj_button_leicht* das Objekt *obj_button_mother* als Mutter hat. Die Eigenschaften von *obj_button_mother* werden dann an die Kinder von *obj_button_leicht* weitergereicht.

Player Beispiel

Im Projekt [GM Tank Combat](#) werden wir im zweiten Teil einen zweiten Player einbauen und dem Panzer individuelle Eigenschaften sowie einen eigenen Schuss geben. Das Prinzip funktioniert wie bei den Buttons. *obj_tank01* ist die Mutter, *obj_tank02* das Kind. Bereits im ersten Teil der Serie habe ich alle Eigenschaften im Create-Event erstellt.

```
// Tank-Speed  
maxSpeed = 6;  
minSpeed = -3;  
rotationSpeed = 3;  
speedUp = 0.6;  
speedDown = 0.3;
```

```
brakeUp = 0.15;
brakeDown = 0.2;

// Bullet
canShoot = true;
bulletDelay = 60;
shooteTimer = bulletDelay;
bulletObject = obj_bullet01;
bulletDontKill = id;

// Steuerung
keyLeft = vk_left;
keyRight = vk_right;
keyUp = vk_up;
keyDown = vk_down;
keyShoot = vk_rcontrol;
```

obj_tank02 bekommt im zweiten Teil alle Events vererbt, bis auf Create. Hier können wir nun bequem alle individuellen Eigenschaften des Panzers einstellen. Die Grafik stellen wir natürlich über die Objekteigenschaften ein.

Weitere Anwendungsmöglichkeiten

Wichtig ist, wie gesagt, dass wir im Create-Event alle Eigenschaften definieren und in den anderen Events nur noch mit den Variablen arbeiten, dann sollte es bei der Vererbung nie Probleme geben.

Es gibt aber noch weitere Anwendungsmöglichkeit in Spielen. Wenn wir keine Tiles benutzen sondern Objekte als Wände, könnten wir auch verschiedene Farben/Sprites nutzen, je nach Level. Auch hier kann man eine Wand als Mutter definieren und die restlichen als Kind. Die Kollisionserkennung läuft dann nur über die Mutter und wir müssen nicht jede Wand einzeln ansprechen.

Bullets, bei denen sich nur Grafik, Richtung und Geschwindigkeit unterscheiden, kann man ebenfalls so handhaben. So lange es keine gravierenden Unterschiede gibt, ist das kein Problem.

Gleiches gilt für NPCs, die je nach Komplexität in Spielen recht ähnlich sind. Wir können in einem Top-Down Shooter einen Charakter bestimmen und dann, ähnlich wie beim Panzer-Beispiel, den Rest über die Parameter im Create-Event lösen. Die Bewegungen, KI, Kollision, Verhalten bei Treffern etc. sind aber immer gleich oder leiten sich von den definierten Parametern ab.

Vor allem wenn man noch nicht so viele Erfahrungen damit hat sollte man sich vorher etwas Gedanken machen, wie man die Herausforderungen am effektivsten Lösen kann. Ebenso wie man nicht für jede Kleinigkeit ein eigenes Skript braucht, kann auch eine Vererbung überflüssig sein. Viel häufiger wird aber der umgekehrte Fall auftreten. Man hackt Objekte mit Code rein und später stellt man fest, dass man es auch hätte viel einfach lösen können. Deshalb: Vorher etwas überlegen und hinterher viel Zeit sparen.

Date Created

28. Juni 2018

Author

sven