Schöner programmieren - Programmierstile und Gruppenarbeit

von THE_BLACK_WALL

Hinweis: wer hier einen Einstieg in Game Maker oder speziell in GML erwartet, ist hier falsch. Wer solch ein Tutorial sucht, sollte mal hier einen Blick riskieren: HTTP://WWW.GM-D.DE/WBB/INDEX.PHP/THREAD/2270-GML-DIE-EINSTEIGERREFERENZ/

Kapitel

0 Vorwort

Eines der wichtigsten Regeln bei der Programmierung ist folgende: Man arbeitet immer im Team, auch wenn man alleine arbeitet. Was nach einem Widerspruch klingt, hat tatsächlich einen tieferen Sinn.

Selbst wenn man ganz alleine arbeitet, kann es immer mal vorkommen, dass man Hilfe braucht. In der Regel geht man in ein Forum, beschreibt kurz sein Problem und kopiert den Code hinein, womit meistens die Schwierigkeiten beginnen. Verstehen die Leute in einem englischen Forum den Code, wenn die Variablen auf Deutsch sind? Ist der Code gut kommentiert? Ist er so aufgebaut, dass man ihn gut lesen und verstehen kann? Je besser man gearbeitet hat, umso eher ist jemand bereit und auch in der Lage zu helfen.

Aber selbst wenn man alleine arbeitet und keine Hilfe braucht, kommt irgendwann der Tag, an dem man sich seinen eigenen, älteren Code anschaut. Spätestens dann versteht man, was mit Teamarbeit gemeint ist, obwohl man alleine arbeitet. Man muss mit seinem früheren Ich zusammen arbeiten und je nach Codequalität ist das nicht einfach, manchmal sogar fast unmöglich.

Ich bin zwar kein Programmier-Guru, aber ich habe viele Erfahrungen, vor allem Projekt-Erfahrungen und einiges davon würde ich hier gerne weiter geben. Es richtet sich vorwiegend an Anfänger, aber einige Dinge könnten auch für Fortgeschrittene interessant sein. Mein Schwerpunkt liegt bei der Programmierung, wenn es gewünscht ist, kann ich gerne noch einen Text zum Thema Teamarbeit verfassen.

Neben vielen Tipps sind natürlich auch subjektive Meinungen dabei und natürlich würden mich eure Erfahrungen und Meinungen interessieren.

Bevor es los geht, noch ein paar Worte zu meinem eigenen Background. Ich programmiere zwar seit 1993, allerdings teilweise mit erheblichen Pausen dazwischen. Persönliche Erfahrungen habe ich mit QBasic, Pascal, THP (ne Scriptsprache unter DOS), Visual Basic, VBA unter Excel, Lite-C, JavaScript, PHP und GML. Ich habe viele kleinere, mittelgroße aber auch ein paar große Projekte abgewickelt, nicht nur Spiele und auch nicht immer als Programmierer. Der krasseste Fehler den ich bewundern durfte, betraf die Steuerung einer Entgiftungsanlage für chemische Abwässer, bei dem der Programmierer beim Programmablauf für die Entgiftung ein falsches Ventil öffnete und dadurch die tödliche Blausäure entstand. Der Vorfall kostete mich fast das Leben, nur weil so ein Programmieraffe nicht bei der Sache war. Das vom Ablauf her kurioseste Projekt war mit einer

indischen Programmierfirma. Die konnten kein Deutsch, ich nicht genug Englisch aber nach fast zwei Jahren Entwicklung kam eine ziemlich coole Firmensoftware raus die es ermöglichte eine komplexe Produktion zu steuern. Ich hoffe, dass ich aus dem Erfahrungsschatz etwas an euch weiter geben kann.

1 Disziplin

Programmierung ist vor allem logisches Denken, je nach Aufgabe auch Mathematik, aber sie sollte immer aus Disziplin bestehen. Jeder der mal schnell einige Zeilen Code geschrieben hat und sich diesen Code nach einigen Tage, Wochen oder Monaten ansieht, weiß worauf ich hinaus will. Ohne die nötige Disziplin versteht man irgendwann seinen eigenen Code nicht mehr und im schlimmsten Fall versucht man an einer Stelle einen Fehler zu beseitigen, die man nicht mehr nachvollziehen kann. Wenn alle Stricke reißen, programmiert man die Stelle neu und stellt am Ende fest, dass dieser Bereich überhaupt keinen Fehler hatte.

Wie solche Dinge passieren können? Es beginnt mit fragwürdigen Variablen, Hilfsvariablen, schlechten oder gar falschen Kommentaren, chaotische Einrückung und mündet dann in einer Sackgasse voll mit wirren Gedanken. Dabei ist es schon schwierig den eigenen Code zu verstehen, spätestens wenn zwei Programmierer oder mehrere zusammenarbeiten, hat Disziplin oberste Priorität. Die Grundregel lautet dabei: Der Code muss nicht funktionieren, aber er muss lesbar sein!

Wer nach dieser scheinbar einfachen Regel programmiert, erspart sich viel Ärger. Da es aber sehr schwierig ist, völlig diszipliniert zu programmieren, erfordert es Regeln und viel Übung. Selbst wenn ein Abschnitt gut aussieht und funktioniert sollte man noch einmal die Disziplin aufbringen und schauen, was man verbessern kann.

Viele Programmierer leben nach dem Motto: Guten Code muss man nicht kommentieren! Ich habe da eine ganz andere Meinung und vertrete die Auffassung, dass jeder Code zumindest einen kleinen Kommentar benötigt der sagt, was der Programmierer damit beabsichtigt. Selbst wenn der Code wirklich toll ist, ist ein Kommentar manchmal kürzer und somit bei der Suche nach Fehlern sehr hilfreich wenn es darum geht, ob man an der richtigen Stelle sucht. Um dies zu realisieren und die unten stehenden Ratschläge zu befolgen, braucht es sehr viel Disziplin.

Wer nicht nur in GML programmiert sondern sich der großen Welt der Webprogrammierung öffnet, wird hier irgendwann auch darauf stoßen, dass man mit HTML, JavaScript und PHP drei

Sprachen mischt. Vor allem bei Umgang mit HTML und PHP braucht es viel Disziplin, wenn es um die Trennung geht. Eine wichtige Grundregel ist, auf die Trennung von Ausgabecode und Seitenlogik zu achten. Im Code, der die eigentliche Seitenlogik darstellt, darf kein HTML auftauchen. Umgekehrt heißt das aber nicht, dass man im Ausgabecode kein PHP verwenden darf.

Wie so oft muss man auch die nötige Disziplin erlernen und trainieren. Deshalb ist es ratsam, erst viele kleine Projekte zu realisieren um sich Stück für Stück zu verbessern. Besonders wenn man Anfänger ist, sollte man nicht einfach drauf loslegen sondern sich vorher Gedanken machen, was man dabei lernen will. Das ist ein Fehler, den ich selber zu oft gemacht habe.

2 Die richtige Sprache

Nein, damit meine ich nicht die Programmiersprache. Dieses kleine Tutorial ist ohnehin sehr allgemein gehalten, weshalb es meistens total egal ist, ob in C, C++, GML, Basic oder VBA Programmiert wird.

Mit Sprache meine ich, in welcher Sprache man Variablen und Kommentare verfasst. Wie oben bereits beschrieben können die ersten Probleme auftauchen, wenn man seinen Code in ein Forum postet. Natürlich kann nun jeder von sich behaupten, dass man ohnehin alles auf Deutsch macht und nur in Deutschen Foren unterwegs ist und ohnehin alles alleine macht, aber dann kommt dieser Tag X, wenn einem im deutschsprachigen Forum keiner helfen kann oder will. Oder es kommt Tag Y, an dem sich ein Russe meldet, der Interesse daran hat beim Projekt zu helfen, allerdings kann der nur Russisch und Englisch.

Wenn man der deutschen Sprache mächtig ist, hat man folgende Möglichkeiten: man kann alles in Deutsch schreiben und hoffen, dass weder Tag X noch Tag Y kommen. Allerdings kann auch Tag Z kommen, wenn man ein neues, größeres Projekt startet und man einige Bestandteile des kleineren Projekts mit einbauen will, etwa den Umgang mit INIs, Savegames und andere universelle Dinge. Und weil man das Projekt bereits mit dem Russen und dem Inder startet, kann man diese Passagen gleich überarbeiten.

Der zweite Weg ist ein Kompromiss, den viele machen. Code komplett in Englisch, Kommentare auf Deutsch. Das hat den Vorteil, dann an den Tagen X, Y und Z nur der Kommentar neu gemacht werden muss und somit die Funktionalität des Codes nicht beeinträchtig wird.

Der dritte Weg ist, man macht alles auf Englisch. Das fällt vielen (auch mir) schwer, aber vor allem bei größeren Projekten nützlich, erst recht wenn man mit mehrsprachiger Software arbeitet. Für Übersetzter sollte es eine Basis geben und das ist nun einmal die englische Sprache. Wenn Variablen und/oder Kommentare mal Deutsch und mal Englisch sind, gibt es ein riesen Chaos.

Eine Ausnahme können Tutorials und Beispiele sein, die für ein spezielles Forum verfasst werden. Da ist es sogar von Vorteil, wenn die jeweilige Landessprache verwendet wird.

Wichtig: Vor Projektbeginn sollte man sich festlegen und diesen Standard konsequent und diszipliniert durchziehen. Das fällt oft bei Variablen schwer, wenn einem das deutsche Wort eher einfällt als das englische, aber hier sollten Geschwindigkeit und Bequemlichkeit keinen Vorrang erhalten.

Noch kurz ein Beispiel für die Leute, die meinen, dass so ein paar Worte auf Deutsch doch kein Problem sein sollten. Angenommen, der Russe sendet euch einen älteren Code für das neue, gemeinsame Projekt. Der Code sieht dann so aus:

```
// Код сравнивает животных вместе

if (kot3 > cobaka2)
{
    ocen = pbiqa * kot3 + cobaka2;
}else{
    ocen = kot3 - pbiqa;
}
```

3 Variablen

Das Thema Variablen ist ein sehr komplexes und ich versuche meine wichtigsten Aussagen zum Thema zu raffen, damit es nicht komplett ausartet.

Wie ich später noch erwähnen werde, ist GML eine relativ einfache Programmiersprache, die aber dadurch auch eine gewisse Schlamperei fördert. Dazu kommt noch der Umstand, dass GM einen nur in gravierenden Fällen vor Fehlern warn, der Rest taucht erst beim spielen auf. Ein Klassiker sind Variablen, die man nicht deklariert, aber irgendwo im Spiel abfragt.

Die in Kapitel 2 erwähnten sprachlichen Probleme schieben wir einfach mal beiseite und beschränken uns auf die übrigen Probleme mit Variablen.

Das erste Problem ist die Deklaration. Eigentlich ist es eine bequeme Sache, dass man nicht gezwungen ist, eine Variable erst einzurichten. Die Sache hat allerdings auch einen Hacken, über den ich immer mal wieder gestolpert bin: man verwendet Variablennamen mehrfach. Das Problem taucht bevorzugt dann auf, wenn man den Variablen keinen gescheiten Namen gibt (siehe weiter unten) und auch ganz gerne dann, wenn man nach einer bestimmten Zeit den Code erweitern will, aber nicht mehr den ganzen Code liest. Mehrfach ist mir das im GM beim Draw-Event passiert, wo man gerne, beispielsweise für die GUI, viele Dinge auf eine Seite programmiert (auch so eine Sache die man nicht machen sollte). Wenn man jede Variable artig oben oder beim Create deklariert, gibt es hier weniger Probleme. Bevor man den Code erweitert schaut man erst, welche Variablennamen bereits existieren und denkt sich für die neuen Aufgaben auch neue Variablenamen aus.

Die zweite Sache ist etwas, worüber man streiten kann, ich will es aber dennoch als Anregung erwähnen, auch wenn ich zugeben muss, dass ich das in GM auch nicht so mache. In GML haben Variablen keine Vorzeichen wie beispielsweise in PHP das $-Zeichen. Manchmal kann es etwas stören, wenn man nicht sofort sieht, wo die Variablen im Code sind. Manche machen folgendes: jede Variable bekommt einen bis drei Buchstaben vorangestellt, also Beispielsweise ein v oder ein var. Aus der Variable ObjectCounter wird also varObjectCounter. Das ist zwar etwas Zusatzarbeit, kann aber dennoch hilfreich sein, vor allem bei Diskussionen im Teamchat, damit man gleich weiß, dass man von einer Variable spricht.

Nun aber zum wichtigsten Thema, welches auch in Tutorials und Fachbüchern gerne vernachlässigt wird: Namen für Variablen. Einige Programmierer finden es besonders witzig und cool, wenn die Variablen kryptisch, sehr kurz oder lustig sind. Das mag für einen Gag in der Mittagspause gut sein, aber auf Dauer sind solche Variablen unnütz und ein blanker Horror. Dabei gilt es zwei Dinge zu beachten: eine Variable muss einen klaren, individuellen Namen haben und man muss den Namen auch lesen können. Hier eine kleine Liste mit schlechten Namen:

zahl1

flt

zaehlerfuerallesundjenes

isdoublescore

mU_ha_ha

essen

meinevariable

Im Prinzip ist es sehr nützlich, wenn eine Variable aus mehreren Wörtern besteht, weil man damit besser beschreiben kann, wozu sie da ist, allerdings kommt es dann auf die Schreibweise an. Ich empfehle die Wörter mit Großbuchstaben voneinander zu trennen. Also isDoubleScore statt isdoublescore. Eine andere Möglichkeit ist die Verwendung von Unterstrichen, etwa bei einer Variable mit dem Namen lang_imp_sektions_start_date. Auch auf solche Standards sollte man sich vor Projektbeginn einigen. Wenn man sich nicht gleich einigen kann, sollte man schauen was in der jeweiligen Programmiersprache meistens verwendet wird.

Tipp: man sollte nicht stur auf sein beliebtes System pochen, nur weil man zu faul ist sich anzupassen. Sofern es gut lesbar ist, sind die Standards egal, sie sollten nur nicht zu kompliziert sein.

Beispiel für übertriebenen Standard: var_is_Double_Score

Ich behaupte mal, dass sich Anfänger über nichts so wenig Gedanken machen als über die Namen ihrer Variablen. Dabei sollte man sich dessen bewusst sein, dass es mühsam ist Code zu lesen, nicht nur fremden sondern auch eigenen. Selbst wenn man den Code gut dokumentiert und beschreibt, was die einzelnen Variablen bedeuten, kann es im Code dennoch sehr mühsam sein zu verstehen was passiert, wenn man als Variablenamen nur zahl1, zahl2, zahl3 etc. hat oder gar kryptische, unübliche Abkürzungen. Das Thema Abkürzungen wird im nächsten Kapitel noch eines sein, aber an dieser Stelle möchte ich darauf hinweisen, dass man möglichst auf Abkürzungen verzichten sollte und nur dann welche nimmt, wenn diese üblich und für alle klar sind.

Bevor man also einfach darauf losprogrammiert, sollte man sich für einige Sekunden die Zeit nehmen und überlegen, welcher Name für die Variable am besten wäre. Wenn einem selbst nach einigen Minuten nichts einfällt, kann man auch mal ein Provisorium nehmen, sollte das aber im Code deutlich vermerken, damit man sich der Sache später annehmen kann.

Im Übrigen: manchen ist vielleicht aufgefallen, dass es sich bei der Variable isDoubleScore möglicherweise um eine Boolesche Variable handelt, was man am is am Anfang des Namens erkennen kann. Schon mit solchen Kleinigkeiten kann man viele Missverständnisse im Code vermeiden.

4 Ressourcenbezeichnungen

Das Thema kennt jeder, der in GM etwas gemacht hat, betrifft aber auch andere Entwicklungsumgebungen. Wie benenne ich meine Ressourcen?

Auch hier setze ich voraus, dass wir die sprachliche Hürde genommen haben. Unabhängig vom eigentlichen Namen stellt sich hier wie auch schon bei den Variablen die Frage, ob ich Worte mit Unterstrichen oder Großbuchstaben voneinander trenne. Ich selber habe es mir zur Gewohnheit gemacht in GML Variablen mit Großbuchstaben und Ressourcen mit Unterstrichen zu trennen. Das hat den Vorteil, dass man gleich sieht, ob es sich um eine Ressource oder eine Variable handelt.

Bei den Ressourcen kommt es aber gerne (eigentlich ungerne) vor, dass man sie doppelt verwendet, beispielsweise Sprites und Objekte. Man hat einen Sprite mit dem Player und macht daraus ein Objekt mit dem Player. In GML ist es üblich, Ressourcen ein Präfix voran zu stellen. Gebräuchlich sind folgende Präfixe:

spr_ für Sprite

obj_ für Objekt

scr_ für Script

fnt_ für Font

bg_ für Background

snd_ für Soundeffekt

Bei Räumen gibt es unterschiedliche Präfixe: rm_, r_, room_

Ich verwende letzteres, dann gibt es keine Missverständnisse.

Anschließend folgt der eigentliche Name, den man, wie oben beschrieben, durch Unterstriche trennen kann. Bei Level und anderen Ressourcen kann es passieren, dass man viele mit dem

gleichen Namen hat und nur hoch zählt. Meiner Erfahrung nach eignet sich dabei eine dreistellige Zahl ganz gut, beispielsweise room_level_001. Das System hilft auch bei der Benennung von Sprites, wenn man beispielsweise Bilder rendert, weil viele Programme die Zahlen nicht als ganze Zahl sortieren sondern nach den einzelnen Ziffern. Dann kommt nach 1 nicht die 2 sondern die 10, weshalb es angebracht ist, der 1 eine oder besser zwei Nullen voran zu stellen.

Im GM hat man nicht nur die Möglichkeit die Ressourcen nach dem Alphabet zu sortieren, sondern auch die Möglichkeit Ordner anzulegen, was man schon bei mittelgroßen Projekten tun sollte. Allerdings sollte man sich auch Ordnernamen und Sortierung gut überlegen, bevor man hier im Chaos versinkt und am Ende viel Zeit damit verliert, Ressourcen zu suchen. Die Ordner in GM sind nur virtuell und nicht physisch, weshalb ich Ressourcen die ich gerade brauche, gerne in den Hauptordner verschiebe und sie am Ende meiner Arbeit wieder einsortiere.

5 Codekommentare

Code kommentieren, dass klingt so simpel wie überflüssig. Warum sollte man auch etwas, das ohnehin schon dran steht, noch einmal beschreiben? Man könnte tatsächlich meinen, wer viel Code kommentiert, hat einfach zu viel Zeit, oder schlicht keine Ahnung vom programmieren. Wer das behauptet, ist entweder ein Genie, oder hat selber keine Ahnung.

Wie bereits gesagt, ist Programmcode nicht gerade etwas, dass der Mensch intuitiv versteht. Selbst Programmierer haben ab und an Mühe, längeren Code zu lesen, selbst den eigenen. Das liegt an vielen Dingen, vor allem aber daran, dass man sich beim lesen ein technisches, teils recht abstraktes Bild eines Programms machen muss, um zu verstehen, was da passiert. Man muss also wie ein Computer denken, in Bildern, die Menschen verstehen.

Wer gut Kommentiert, hat es nicht nur im Team leichter. Kommentare helfen einem schnell dabei, den nachfolgenden Code zu verstehen, ohne ihn zu lesen. Vorausgesetzt der Kommentar ist richtig, verständlich und aktuell. Kommentare können aber auch mehr verwirren als helfen. Beispielsweise, wenn er so ausfällt:

```
// TODO: Das muss noch angepasst werden.
```

Missverständlich ist der Kommentar, weil man nicht weiß, was angepasst werden muss und warum. Deshalb sollten solche Kommentare ruhig etwas länger sein, damit man auch in ein paar Wochen, also meist viele 100 oder 1000 Zeilen später, noch weiß, was man sich dabei gedacht hat.

Kommentare sind also meistens Nachrichten an sich selber in der Zukunft. Im schlimmsten Fall sind es Stolpersteine für sich selber in der Zukunft, oder für ein ganzes Team.

Wie das Beispiel oben schon zeigt, kann man mit Kommentaren auch Stellen markieren, an denen man noch schrauben will. TODO oder FIXME sind dabei sehr hilfreich. Selbst mit Notepad++ kann man dann alle offenen Dateien danach absuchen und abarbeiten, das ist sehr nützlich!

Kommentare sollten ehrlich sein. Wenn man eine Sache nicht hin bekommt oder nicht versteht, gehört das in den Kommentar. Beispiel:

```
// Hier wird geschaut, wie die Benutzergruppe heißt.
// TODO: Ggf. würde man die Datenbank weniger belasten, wenn man das vorher ausliest und
in einem
// Array speichert. Allerdings hatte ich keine Zeit das zu versuchen.
```

Auch nützlich sind Kommentare, wenn man Code aus einem Fremden Forum oder einer Webseite verwendet, ggf. Hinweise auf Referenzen, Hilfen etc. kann man gerne mit rein nehmen. Wichtig ist vor allem ein Link zur Quelle, damit man immer wieder dort hin kann, wenn man ein Problem hat.

Am nützlichsten finde ich aber Kommentare mittlerweile, bevor ich etwas programmiere. Anfänger coden einfach mal darauf los, pfuschen herum, ändern ggf. das Konzept und kommentieren am Ende nicht einmal den Knäul an Code. Viel besser ist es vorher zu beschreiben, was man programmieren will. Dann hat man schon einen Leitfaden und kann am Ende kontrollieren, ob man wirklich das erhalten hat, was man sich vorgenommen hat. Beispiel:

```
/* Login Abfragen starten. Das "Verhalten" steuert dabei, was unter bestimmten
*  Bedingungen passiert. 0 ist nichts, 1 ist, er wird angemeldet und 2 ergibt einen
*  Fehler auf der Login-Seite.
*/
```

Diese Technik hat mir schon manchmal geholfen, den Code recht entspannt herunter zu tippen. Wer dies noch nicht verwendet, sollte es auf jeden Fall mal probieren.

6 Einrückung

Ein beliebtes Streitthema unter Paaren war früher die Zahnpastatube im Badezimmer, welche so funktionierte wie die Senftube. Streitpunkt ist oft, dass ein Partner irrsinnigen Wert darauf legt, dass die Tube stets aufgerollt ist, während der andere Partner dazu eher ein entspanntes Verhältnis hat. Ähnlich ist es beim Thema Einrückung unter Programmierern. Wenn man es geschickt anstellt, kann man hier unter Programmierern sogar eine Massenschlägerei anzetteln. ;)

Worum geht es eigentlich? Am besten zeige ich das an einem Stück Code, den man auf verschiedene Arten formatieren kann. Hier sind zwei Beispiele:

Version 1

```
for(i=0; i<360; i+=30){
draw_line_width(xx+lengthdir_x(innen-15,i),yy+lengthdir_y(innen-
15,i),xx+lengthdir_x(innen,i),yy+lengthdir_y(innen,i),4)
draw_text(xx+lengthdir_x(innen-32,i),yy+lengthdir_y(innen-32,i),string(a))
if(a=1){a=12}else{a--}
}
```

Version 2

```
for (i = 0; i < 360; i += 30)
{
    draw_line_width(xx + lengthdir_x(innen - 15, i), yy + lengthdir_y(innen - 15, i), xx +
lengthdir_x(innen, i), yy + lengthdir_y(innen, i), 4);
    draw_text(xx + lengthdir_x(innen - 32, i), yy + lengthdir_y(innen - 32, i),
string(a));

    if (a = 1)
    {
        a = 12;
    }else{
        a--;
    }
}
```

Beide Versionen funktionieren identisch, aber ich bin der Meinung, dass man bei der zweiten Version viel besser versteht, was da eigentlich passiert. Dabei reden wir hier nur von ganz wenigen Zeilen. Mit jeder weiteren Zeile wird es immer schwieriger und die Formatierung des Codes immer wichtiger.

Je nach Editor / Programmiersprache ist die Einrücktiefe ein Thema. Da wir im Forum mit GM die selbe Entwicklungsumgebung nutzen, möchte ich an der Stelle kein Fass aufmachen, aber darauf

hinweisen, dass bei anderen Projekten es im Team ein Thema werden kann, um wie viele Zeichen der Courser springt, wenn man die Tab-Taste drückt. Im Zweifelsfall sollte man schauen, welchen Standard es bei der verwendeten Programmiersprache gibt.

Ein zweites Thema sind Zeilenumbrüche in Verbindung mit geschweiften Klammern. Da gibt es verschiedene Wege, im Team sollte man dieses Thema aber vor Projektbeginn abhandeln, damit der Code einheitlich aussieht. Hier ein paar Beispiele:

Version 1

```
if (bedingung){

}else{

}
```

Version 2

```
if (bedingung)
{

}else{

}
```

Version 3

```
if (bedingung)
{

} else {

}
```

Version 4

```
if (bedingung)
```

```
{

}
else
{

}
```

Ich persönlich favorisiere die Version 2, weil ich in den Leerzeichen beim else nicht viel Sinn erkennen kann, aber die geschweifte Klammer gerne in einer eigenen Zeile habe.

An dieser Stelle noch zwei Hinweise. In Tutorials und Fachbüchern wird darauf hingewiesen, dass man auf geschweifte Klammern verzichten kann, wenn danach nur eine Zeile kommt. Von dieser Praxis rate ich vor allem Anfängern ab, weil man sich einerseits an einen einheitlichen Aufbau gewöhnen sollte, andererseits man vor allem als Anfänger seinen Code immer wieder erweitern muss und man dann auch gerne vergisst die Klammern nachträglich zu setzen.

Das zweite Thema ist die Frage, wann man Code in nur eine Zeile schreibt und wann nicht. Ich persönlich bin der Auffassung, dass man möglichst immer nach demselben Aufbau arbeiten sollte, ein if Konstrukt also IMMER mehrere Zeilen hat. Nach meiner oben beschriebenen Methode wären das also immer mindestens drei Zeilen. Für mich habe ich zwei Ausnahmen festgemacht. Angenommen, wir haben wirklich nur eine sehr kurze Abfrage:

```
if (a = 1)
{
   a = 12;
}
```

Das kann man auch so darstellen:

```
if (a = 1){a = 12;}
```

Ich mache dies in zwei Fällen. Erstens, wenn ich nur schnell etwas rein schreibe um zu testen, wie es aussieht. In dem Fall wird der Code entweder ohnehin gelöscht oder nachträglich formatiert. Die zweite Ausnahme ist, dass man viele solcher sehr kurzen Abfragen hat und es übersichtlicher ist, jeweils nur eine Zeile zu verwenden.

Beispiel:

```
if (a = 1){a = 12;}
if (a = 2){a = 15;}
if (a = 3){a = 24;}
if (a = 4){a = 125;}
if (a = 5){a = 999;}
```

Wer sich in das Thema ein wenig vertiefen will, kann gerne hier einen Blick riskieren:

HTTPS://DE.WIKIPEDIA.ORG/WIKI/EINR%C3%BCCKUNGSSTIL

Ein anderes, nicht unwichtiges Thema ist die Verwendung von Leerzeichen, Beispielsweise bei Funktionen.

Man kann es so schreiben:

```
draw_line_width(xx, yy, xx + lengthdir_x(s_laeng, seconds_dir), yy + lengthdir_y(s_laeng,
seconds_dir), s_breit);
```

oder so:

```
draw_line_width(xx,yy,xx+lengthdir_x(s_laeng,seconds_dir),
yy+lengthdir_y(s_laeng,seconds_dir),s_breit);
```

In GM-Kreisen gibt es leider die Unart, sowohl nach dem Komma als auch zwischen den Operatoren keine Leerzeichen zu verwenden. Viele schreiben x=32 statt x = 32. Bei wenig Code und kurzen Zeilen mag das nicht so ausschlaggebend sein, aber bei langen Codestücken, Funktionen mit vielen Argumenten und entsprechenden Parametern kann die Verwendung von Leerzeichen die Fehlersuche entscheidend erleichtern.

Das letzte Thema in diesem Kapitel ist die Verwendung vom Semikolon (dieses Zeichen: ;). GM verlangt es im Gegensatz zu einigen Programmiersprachen nicht, lässt es aber zu. Einerseits macht das GML einfacher, weil es einen nicht zwingt, am Ende einer Befehlszeile ein Semikolon zu setzen. GM zieht es so krass durch, dass es selbst bei einer for-Schleife kein Semikolon braucht. for (i = 0 i < 360 i += 30) geht also ebenso wie for (i = 0; i < 360; i += 30). Interessant ist, dass viele Tutorials auf ein Semikolon verzichten, außer bei einer for-Schleife. Der Grund liegt in der Lesbarkeit, da ich mit einem Semikolon sofort sehe, wo ein Bereich beginnt und wo ein anderer

endet. Soweit ich weiß, ist das Semikolon nur noch bei globalvar pflicht, früher war es das auch bei var. Hier liegt für mich ein wenig der Hase im Pfeffer: es gibt sowohl eine Ausnahme als auch eine nützliche Anwendung (for-Schleife) für ein Semikolon. Um konsequent zu sein, sollte man meiner Meinung nach eher dazu über gehen und alles mit Semikolon machen.

Durch andere Sprachen, vor allem durch PHP, habe ich mich an das Semikolon so extrem gewöhnt, dass ich es nicht mehr misse und mir ehrlich gesagt Code ohne Semikolon irgendwie falsch vor kommt. Wirklich Probleme hatte ich zu einer Zeit, als ich sowohl in PHP als auch in Excel-VBA viel gemacht habe und ich in VBA laufend ein Semikolon geschrieben habe, obwohl es VBA nicht zulässt. Dieses und andere Problemchen haben mich dazu veranlasst die Welt von Excel-VBA zu verlassen und nur noch auf Webprogrammierung zu setzen.

7 Richtig kürzen

Nicht nur beim Einrücken und beim Aufbau stellt man fest, dass es verschiedene Wege gibt, etwas zu programmieren. Als Anfänger neigt man dazu, manches umständlicher zu machen als es sein muss, als Fortgeschrittener neigt man dazu, zu viel zu kürzen. Ein Klassiker bei Anfängern ist folgende Abfrage:

```
if (zeug == true)
{
```

oder

```
if (zeug == false)
{
```

Einfacher geht:

```
if (zeug)
{
```

oder

```
if (!zeug)
```

```
{
```

Das spart nicht nur Arbeit und damit Zeit, sondern wird auch allgemein so gehandhabt, kann also im Prinzip jeder Programmierer lesen. Die zweite Frage die sich stellt ist die, ob man die Klammern um die Abfrage mit schreibt oder nicht, schließlich würde

```
if zeug
{
```

ebenso funktionieren. Es funktioniert sogar, wenn man mehrere Abfragen hintereinander bringt:

```
if !zeug || 8 + 2 = 10
{
```

Allerdings erkennt man hier vielleicht schon das Problem: je mehr Abfragen es werden, umso unübersichtlicher wird es ohne die Verwendung von Klammern, auch wenn das obere Beispiel ein wenig schwachsinnig ist. ;)

Bei Berechnungen im Code rate ich eher dazu ab, möglichst wenig zu kürzen.

```
time = 60 * 60 * 24;
```

oder

```
time = 86400;
```

Das Ergebnis ist genau gleich, aber wer weiß noch genau, woher die Zahl 86400 kam? Wer die Meinung vertritt, dass das Spiel dadurch einen Bruchteil einer Nanosekunde schneller ist und das ganz wichtig sei, sollte dahinter wenigstens einen Kommentar lassen, damit man versteht, was sich dahinter verbirgt. Das kann auch dann nicht schaden, wenn man die Berechnung ausschreibt. Zur Veranschaulichung eine Zeile aus einem JavaScript-Code von mir:

```
var RGW = Math.round((((Math.pow(rRAD, 2) * Math.PI * Breite) - (Math.pow(rRID, 2) *
Math.PI * Breite))  * Dichte) / 1000000 * 100) / 100;
```

Unschwer zu erkennen sind zwei Dinge: die Rechnung lässt sich kürzen, aber auch ohne Kürzung lässt sich kaum nachvollziehen, was sich dahinter verbirgt. Einen Codekommentar gibt es dazu nicht, allerdings wird die Berechnung auf der Homepage mit einer anschaulichen Formel erklärt, was fast noch besser ist.

Wie ich oben bereits angedeutet habe, sind Kürzungen vor allem dann sinnvoll, wenn sie die Lesbarkeit des Codes verbessern und wenn sie üblichen Konventionen entsprechen. In solchen Fällen kann man auch Sachen in eine Zeile schrieben, vorausgesetzt, es erfüllt mindestens eine der beiden Bedingungen. Allerdings sollte man sich Kürzungen nicht angewöhnen, nur weil man es einmal bei jemand anderem gesehen hat. Bevor man Gefahr läuft sich etwas Schlechtes anzugewöhnen, sollte man sich umschauen, ob das andere ebenso tun, also ob diese Methode üblich ist.

Üblich, um nicht zu sagen Pflicht, ist die Verwendung von Schleifen. Wenn man Code mit einer Schleife verkürzen kann, sollte man das auch tun.

Ein streitbares GM-Element findet sich im Draw-Event wieder. Besonders bei der GUI werden an mehrere Stellen Schriften und Farben definiert und dann Texte erzeugt oder Elemente gezeichnet. Hier bietet es sich verlockend an, Elemente mit gleichen Farben und Schriften zu einem Block zusammen zu fassen, was einige Zeilen Code sparen und scheinbar die Übersicht verbessern kann. Diese Vorgehensweise funktioniert so lange gut, bis der Tag kommt an dem man den Code ändern will, weil einem einfällt, dass man bei einem GUI-Element doch eine andere Farbe und eine andere Schrift braucht. Dann beginnt man die Blöcke zu zerschneiden, was Arbeit macht und fehleranfällig ist. Hier ist es ratsam für jedes Element einen eigenen Abschnitt zu schaffen, bei dem die Farbe und die Schrift individuell bestimmt werden, auch wenn man fünf Mal hintereinander dieselbe Schrift und Farbe verwendet.

Kürzungen fallen meistens in den Bereich der Codeoptimierung. Wenn es tatsächlich Bedarf gibt den Code aus Performancegründen zu optimieren, sollte man dies erst am Ende machen und sich auf jeden Fall vom unoptimierten Code eine Kopie machen, ggf. als Kommentar vor oder nach dem optimierten Code. Wenn man Fehler sucht, ist die Fehlersuche dadurch meist um einiges angenehmer.

8 Copy & Paste

Copy & Paste sind Segen und Fluch zugleich. Ich bin mir nicht mehr sicher wann mir das gezeigt wurde oder ob ich das damals noch im DOS Editor selber heraus gefunden habe, aber an diesem Tag hat sich meine Einstellung der Welt ein wenig geändert. Ich bin mir ziemlich sicher, dass sich in diesem Augenblick der Himmel geöffnet und mein Gesicht ein Sonnenstrahl getroffen hat, welcher mich gen Himmel zog. Aber irgendwann war dieser Strahl aus und ich landete wieder auf dem harten Boden der Realität.

Einen Text, egal ob kurz oder lang, zu kopieren und einzufügen spart i.d.R. viel Arbeit. Da spielt es kaum eine Rolle ob wir in Excel, Word, in einer Mail oder in der Programmierung arbeiten. Der Fluch besteht vorwiegend aus zwei Punkten.

1. Fehler werden mit kopiert und man merkt es nur selten und dann meist zu spät.
2. Man lernt fast nichts außer dem Copy & Paste selber.

Eigentlich ist Copy & Paste sehr gut dazu geeignet Fehler zu vermeiden. Man schreibt etwas einmal richtig und kann den Text dann so oft kopieren, bis einem die Finger abfaulen. Meistens klopft man aber schon den ersten Text schlampig ein und kopiert den Fehler ungelesen weiter. Und wenn man dann in einer Datenbank die Spalte “GoupName” hat und diesen 1000mal im Code übernimmt, stellt man erst dann fest, dass da ein “r” fehlt, wenn man die Bezeichnung mal selber tippt. Entweder lässt man den Fehler so oder man wendet wieder Zeit auf um jede falsche Bezeichnung zu korrigieren.

Wenn man Glück hat, kann man mit suchen und ersetzten den Schaden sehr schnell beheben, allerdings tauchen hier gleich mehrere Probleme auf. Wenn, wie in PHP, man von einzelnen Dateien spricht die den Fehler enthalten, dann ist das noch eine halbwegs angenehme Sache. Wenn man aber im GM den ganzen Code suchen muss, ist das nicht mehr so lustig, vor allem dann nicht, wenn auch Ressourcen davon betroffen sind. Wenn man Glück hat, ist die falsche Bezeichnung individuell. Wenn man Pech hat, gibt es die falsche Bezeichnung auch in einer anderen Form, die aber richtig ist. Beispiel: Variable GroupName und Variable GroupNames. Irgendwann fällt einem auf, dass man im Code laufend durcheinander kommt und man entschließt sich, auf GroupName NameOfGroup zu machen. Wer das Problem schnell mit Suchen und Ersetzen lösen will, sollte sich für den Rest des Tages nichts vornehmen, vor allem wenn er vergisst nach der Variable ein Leerzeichen zu verwenden, damit GroupNames nicht ebenfalls verwandelt wird.

Solltet ihr mal vor der Entscheidung stehen: korrigiert es immer gleich. Ein “das mache ich später” bedeutet zu 99,9999752%, dass es nie gemacht wird. Auch nicht, wenn man eine Engine schreibt und diese für ein weiteres Projekt verwendet.

Fast noch schlimmer als diese mehr oder weniger kleinen Fehler ist aber, dass der Lerneffekt gleich Null ist. Um das wenigstens ein bisschen zu kompensieren, sollte man die kopierte Passage wenigstens noch einmal durchlesen um zu prüfen, ob man es wirklich verstanden hat. Das sollte man auf jeden Fall IMMER tun, wenn man sich mit irgendetwas nicht richtig auskennt, egal was man kopiert. Passagen aus einer Fremdsprache, Programmcode oder einen Abschnitt, der die interionische Wechselwirkung bei konzentrierten Flüssigkeiten beschreibt. IMMER!

Übrigens – und an dieser Stelle schweife ich etwas ab – sollte man sich auch gut überlegen, ob man für alles Tools einsetzt, die einem die Arbeit abnehmen oder ob man sich mal auf den Arsch setzt und den ganzen Mist selber macht um zu verstehen, was da eigentlich passiert. Seit ich das erste Mal mit HTML Editoren gearbeitet habe, habe ich Tabellen, Formulare und so gut wie alles mit einem Editor zusammen geklickt und nur kleine Korrekturen per Hand vorgenommen, oder den Code formatiert. Ja, dass kann man auch machen wenn man keine Ahnung hat.

Das geht schnell, einfach und man hat sein Ziel erreicht. Aber sobald man tiefer eintaucht und zum Beispiel per for-Schleife eine Tabelle erzeugen muss, bringt einem der Editor-Kram nichts. Absolut nichts! Mit etwas Glück kennt man die einzelnen Tags aber wenn man diese per C&P oft genug kopiert hat, nicht einmal das.

C&P zu intensiv eingesetzt ist der dunkle Pfad. Wer diesen in Verbindung mit einfachen Baukästen in Kombination verwendet, kann auch tolle Sachen machen, hat aber am Ende nichts verstanden.

GM zeichnet sich vor allem durch das einfache D&D-System aus. Anfänger erzielen damit sehr schnell kleine Erfolge, aber das Problem ist dabei, dass man sich daran gewöhnt und auch versucht komplexere Dinge damit zu realisieren. Auch hier kann ich nur raten, sich nach dem ersten D&D-Erfolg sich mit GML zu befassen, bevor man zu bequem wird.

9 Einzahl oder Mehrzahl?

Die Namensfindung von Variablen und Ressourcen haben wir bereits behandelt. Was manchmal ein Problem sein kann: verwende ich die Einzahl oder die Mehrzahl? Das Problem stellt sich bei mir vor allem bei Scripten und in PHP bei Funktionsnamen. Ein Script, in dem alle globale Variablen stecken (kann ich nur empfehlen um den Chaos Herr zu werden) kann ich scr_global_variable oder scr_global_variables nennen. Oder ein Script, welches das Aussehen von

bestimmten Wänden steuert (welches Sprite verwendet werden muss) kann ich scr_red_wall oder scr_red_walls nennen.

Auch über dieses Thema sollte man sich vor Projektstart genug Gedanken machen. Ich tue mich manchmal mit der Entscheidung ebenfalls schwer, habe mich aber darauf eingelassen, dass ich nur dann die Mehrzahl verwende, wenn es keinen Fall gibt, in dem keine Mehrzahl vorkommt. Um bei den oberen Beispielen zu bleiben: das Script für die Globale Variablen heißt bei mir scr_global_variables, weil ich immer mehrere habe. Das Script, welches das Aussehen der Wand steuert, heißt scr_red_wall, weil es zwar auch tausend Wände sein können, aber je nach Level kann es auch nur eine Wand sein. Außerdem bestimmt das Script das Aussehen eines einzelnen Steins und fragt nicht alle Steine in einer Schleife ab. Das kann übrigens ebenfalls ein Hinweis sein, ob man Plural verwendet oder nicht: hat das Script Schleifen? Wie bereist dargelegt sollte es nicht vorkommen, dass man beide Namen verwendet, weil man damit in Rekordzeit sein eigenes Grab schaufelt.

10 Große Pausen

Das hat nun nichts mit Programmierstil zu tun, aber ist dennoch ein wichtiger Tipp an Anfänger. Wenn ihr eine Entwicklungsumgebung mit Programmiersprache wie den GM erlernt, dann macht bei euren Projekten keine große Pausen. Programmiersprachen sind wie Fremdsprachen: wenn man sie nicht nutzt, vergisst man sie wieder und das umso mehr, je weniger man sie beherrscht hat. Wer sich also wirklich dazu entscheidet eine Sprache / Entwicklungsumgebung zu erlernen, sollte das eisern durch ziehen und nicht zwischendurch mehrere Wochen pausieren. Selbst wenn man nicht immer programmieren kann, sollte man zumindest immer wieder über die Programmiersprache Artikel lesen oder sich in Foren umsehen, damit man nicht zu vieles verlernt.