Einführung in Game Development mit Gamemaker Studio 2

Inhalt

1. Über diese Lehr-Lerneinheit

In dieser Lehr-Lerneinheit sollen die Schüler_innen im Laufe von zwei bis drei Doppelstunden mithilfe von Gamemaker Studio 2 (kurz GMS) ein kleines Computerspiel erstellen. Dabei lernen die Schüler_innen wie eine Gameengine funktioniert, sowie grundlegende Konzepte des Programmierens, wie Variablen und Verzweigungen, bzw. können diese Konzepte in einem ansprechenden Beispiel anwenden.

Das Spiel heißt Brickout und ist eine vereinfachte Version des Klassikers Breakout. Die Schüler_innen sollen dieses anhand eines Tutorials und/oder durch Anleitung der Lehrperson erstellen.

Eckdaten

Umfang	ca. 4-6 Unterrichtsstunden
Schulstufe[1]	8-12
Benötigtes Vorwissen SuS[2]	keines
Benötigtes Vorwissen Lehrperson	Grundlegende Kenntnisse der Objektorientierten Programmierung, sowie Gamedevelopment und Gamedesign Gute Kenntnisse in GMS
Verortung im Lehrplan	5. Klasse AHS: *Informatiksysteme* – Die Funktionsweise von Informatiksystemen erklären können *Praktische Informatik* – Begriffe und Konzepte der Informatik verstehen und Methoden und Arbeitsweisen anwenden können – Algorithmen erklären, entwerfen, darstellen und in einer Programmiersprache implementieren können
Benötigte Materialien	PCs mit Gamemaker Studio 2 installiert und einen YoYo-Games Account (Mehr dazu in Kapitel 2)

Ziele

Die Schüler_innen …

- … erstellen ein eigenes Computerspiel
- … kennen Sprites und ihre Bedeutung in 2D-Spielen
- … kennen die grundlegenden Funktionen einer Gameengine (Gameloop, Rendering, Collisiondetection
- … verstehen die Begriffe Variablen und Verzweigungen und können die Konzepte dahinter anwenden

2. Gamemaker Studio 2

Gamemaker Studio 2 ist eine Gameengine und Entwicklungsumgebung von YoYo-Games. Es handelt sich dabei um eine einsteigerfreundliche Entwicklungsumgebung, mit der man allerdings durchaus

[1] Aufgrund der einfachen Handhabung und benutzerfreundlichen Oberfläche von Gamemaker kann diese Einheit ohne Probleme im regulären Informatikunterricht in der 5. Klasse AHS, oder auch schon in der Unterstufe ohne Vorwissen durchgeführt werden. Da Gamedevelopment aber auch ein sehr komplexes Thema ist, eignet sie sich durchaus auch für einen späteren Einsatz. Man kann dann mehr in die Tiefe gehen.

[2] Falls die Klasse schon über Programmierkenntnisse in einer Textbasierten Programmiersprache verfügt, empfiehlt sich die in Kapitel *6. Alternative Option mit GML statt Drag'n'Drop* beschriebene Version der LL-Einheit.

auch professionelle Spiele erstellen kann. Ein paar bekannte Gamemaker Spiele sind z.B. *Hotline Miami*, *Spelunky* und *Undertale*.

Vorteile	**Nachteile**
Sehr flache Lernkurve	Proprietär und nicht Gratis
Sowohl Block- als auch Textbasiertes Programmieren möglich	Trial Version sehr eingeschränkt
In vielen unterschiedlichen Sprachen verfügbar	
Einfaches und übersichtliches Userinterface	
Exportmöglichkeiten für alle gängigen Plattformen (mit der passenden Lizenz)	

GMS ist leider proprietäre Software und auch nicht gratis. Allerdings gibt es spezielle Education-Lizenzen, bzw. eine gratis Trial Version, welche für diese LL-Einheit und auch generell für kleine Projekte ausreichend ist. Man benötigt auch für die Trial-Version einen registrierten Account. Allerdings kann ein Trial-Account auf mehreren PCs gleichzeitig verwendet werden, sodass sich die einzelnen Schüler_innen nicht unbedingt einen Account anlegen müssen.[3]

3. Grundlagen der Gameengine

Gameloop

Basis einer jeden Gameengine ist diese sogenannte Gameloop. Eine Schleife die während des Spielens endlos läuft und grob gesagt in jedem Durchlauf einen neuen Spielframe berechnet und diesen auf den Bildschirm zeichnet. In jedem Durchlauf werden u.a. alle Spielobjekte aktualisiert (z.B. die Position anhand der letzten Position und der momentanen Geschwindigkeit berechnet), Kollisionen erkannt und verarbeitet und zum Schluss das neue Bild gerendert.

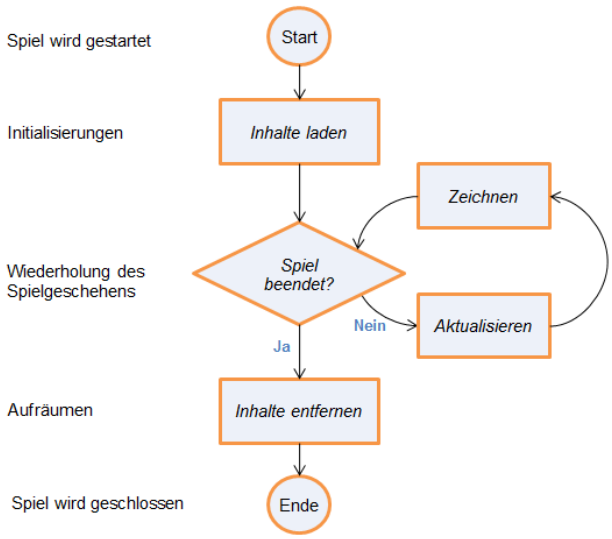

Abbildung 1 Gameloop

[3] Ich konnte in der License-Agreement für die Trial Lizenz nichts finden, was gegen die gleichzeitige Verwendung von einem Account auf mehreren PCs spricht, allerdings bin ich kein Jurist und es kann sein, dass ich etwas übersehen habe. Ich empfehle trotzdem um sicherzugehen die Erstellung eines eigenen Accounts für jedes Endgerät.

Sprites

Bewegliche Spielgrafiken in GMS werden Sprites genannt. Ein Sprite in GMS kann dabei aus mehreren einzelnen Bildern (Frames) bestehen. Wenn ein Sprite aus mehreren Grafiken besteht, wird es automatisch animiert, indem einfach zwischen den Einzelbildern gewechselt wird.

Kollisionsmaske

Eine wichtige Einstellung eines Sprites ist, die Kollisionsmaske. Wenn sich die Kollisionsmasken von zwei Objekten überschneiden, wird eine Kollision zwischen beiden erkannt. Es gibt hierbei verschiedene Arten:

Art	Geschwindigkeit
Rechteck ohne Drehung	Sehr schnell
Rechteck	Schnell
Ellipse	Mittel
Diamant	Mittel
Präzise	Langsam
Präzise pro Frame	Sehr langsam

Man sollte sich immer gut überlegen, welche Kollisionsmaske man einem Sprite gibt. In den meisten Fällen ist eine Kollisionsmaske, die das Sprite nur ungefähr abdeckt völlig ausreichend. Nimmt man hingegen für alle Sprites *Präzise*, kann sich das sehr stark auf die Performance des Spiels auswirken und zu Ruckeln führen.

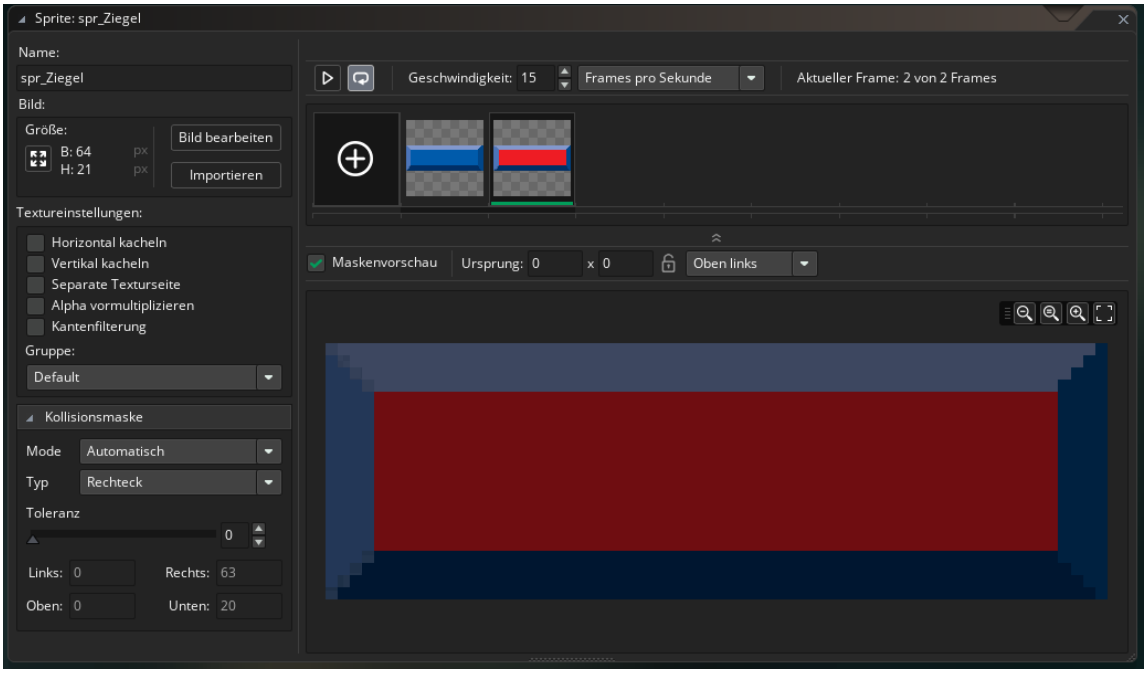

Abbildung 2 Spritefenster

Objekte

Um ein Spiel zu erstellen müssen wir Objekte definieren. In unserem Beispiel, ein Objekt für den Balken, die Ziegel und den Ball. Von diesen Objekten kann man dann Instanzen in das Spiel direkt einfügen, indem man sie einfach in einen *Raum* einfügt. Jedem Spielobjekt kann man ein Sprite zuweisen.

Jede Instanz eines Objekts verfügt auch über eigene Variablen, welche man unter *Variablendefinitionen* deklarieren kann. Viele wichtige Variablen sind schon per Default in allen Objekten definiert.

Ein paar wichtige vordeklarierten Variablen:

Name	Beschreibung
mouse_x, mouse_y[4]	x-, bzw. y- Koordinate des Mauszeigers
x,y	Die Positionskoordinaten des Objekts
speed	Die Geschwindigkeit des Objekts
vspeed, hspeed	Die vertikale bzw. horizontale Geschwindigkeit des Objekts. Der Wert dieser Variable gibt an, um wie viele Pixel sich das Objekt im nächsten Frame vertikal bzw. horizontal verschiebt.

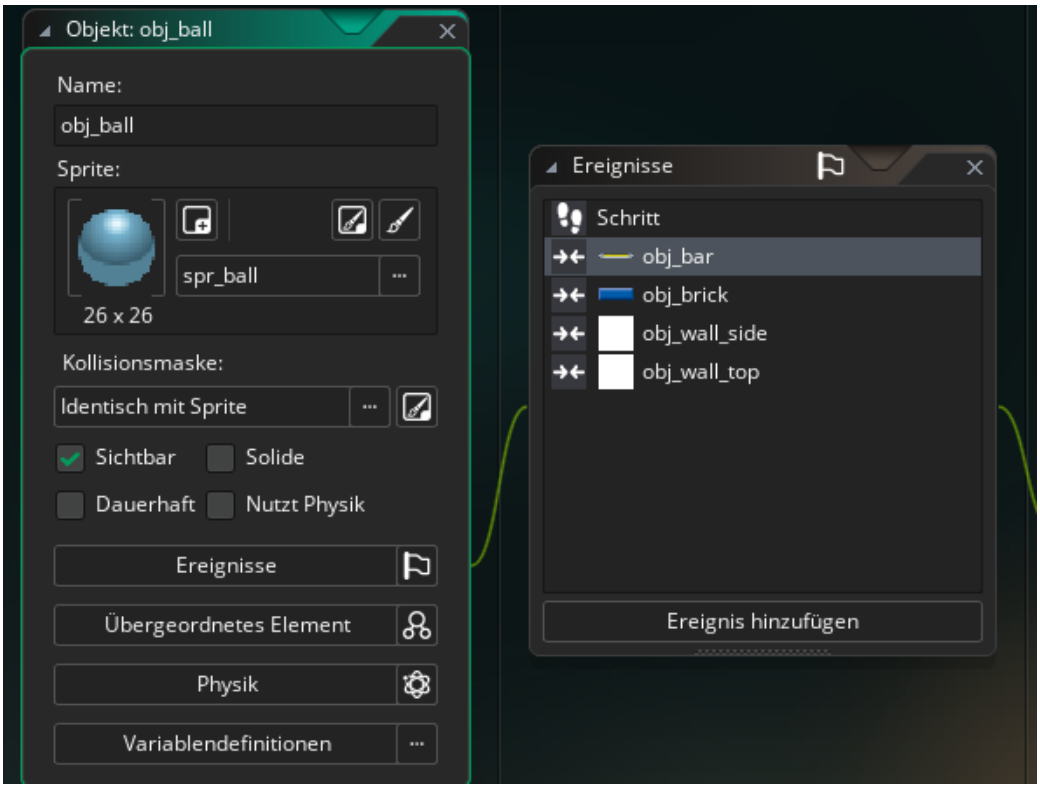

Abbildung 3 Objektfenster

Ereignisse

Über Ereignisse wird das Verhalten von Objekten geregelt. Wenn man z.B. festlegen will, was passiert wenn der Ball mit der Wand kollidiert, kann man das Ereignis *Kollision mit Wand* verwenden.

Ein sehr wichtiges Ereignis ist Schritt. Dieses Ereignis wird einfach in jedem Durchlauf der Gameloop für alle Objekte aufgerufen.

Weiters gibt noch das Zeichnen-Ereignis. Dieses wird am Ende eines Durchlaufs der Gameloop aufgerufen. WICHTIG: Nur in diesem Ereignis kann man Dinge (z.B. einen Text, oder eine Anzeige) direkt auf den Bildschirm zeichnen!

[4] Ist genaugenommen keine Objektvariable sondern eine globale Variable

4. Das Tutorial

Das Tutorial kann als Website unter https://quwetz.github.io/GameMaker-Tutorial/ aufgerufen werden. Die Website wurde mit *mkdocs* erstellt. Die Sourcefiles dazu stehen und der CC0-Lizenz und befinden sich unter https://github.com/quwetz/GameMaker-Tutorial

Das Tutorial wurde so konzipiert, dass man es grundsätzlich ohne jegliches Vorwissen und auch ohne Anweisung durch eine Lehrperson durchführen kann. Dennoch empfehle ich nicht, die Schüler_innen einfach mit dem Tutorial drauf los arbeiten zu lassen, sondern würde ihnen vorher das Programm bzw. das fertige Spiel vorführen, die Benutzeroberfläche zeigen und die Grundlagen der Gameengine erklären.

5. Alternative Option mit GML statt Drag'n'Drop

Sollten die Lernenden bereits über Programmierkenntnisse verfügen, dann empfiehlt es sich meines Erachtens nicht mit Drag'n'Drop, sondern textbasiert mit GML (Gamemaker Language) zu programmieren. Das Drag'n'Drop Interface ist zwar ansprechend und für einfache Dinge völlig ausreichend, allerdings kann man damit nicht alle Tasks ohne weiteres Durchführen, der Workflow ist um einiges mühsamer und Dinge, die zu einem tieferen Verständnis der Engine beitragen können, werden eher versteckt.

Die Quellcodes für die Skripte des Spiels aus dem Tutorial befindet sich unter https://github.com/quwetz/GameMaker-Tutorial im Ordner gml_sources. Das Tutorial selbst ist jedoch nur für den Einsatz mit blockbasierter Programmierung gedacht.

Blöcke in Code übersetzen

Eine sehr praktische Funktion in GMS, die auch einen nahtlosen Wechsel von der Blockbasierten zur textbasierten Programmierung ermöglicht, ist es die Blöcke direkt in GML Code übersetzen zu lassen. Man kann den Code, der hinter den programmierten Blöcken steht in einer Live-Vorschau ansehen, oder die Blöcke direkt in GML-Scripts übersetzen lassen. (ACHTUNG: Dies lässt sich nicht mehr rückgängig machen!)

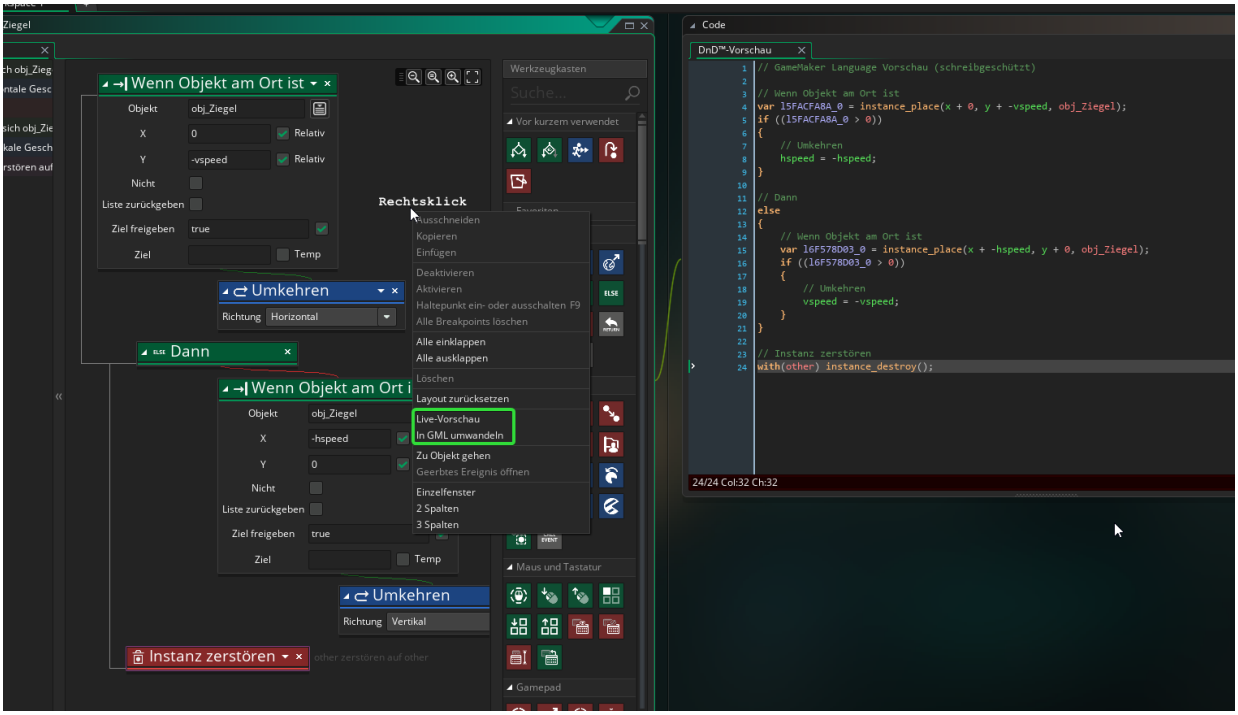

Abbildung 4 Übersetzung in GML

Der dabei generierte Code ist durchaus verständlich formatiert, mit der Ausnahme, dass Variablen automatisch benannt werden.

Möchte man im Unterricht nach dem Tutorial noch tiefer gehen und das Spiel erweitern, oder eigene Projekte in Angriff nehmen, dann empfehle ich spätestens nach Abschluss des Tutorials das gesamte Projekt in GML zu übersetzen, mit den Schüler_innen den generierten Code analysieren und später in GML weiterprogrammieren.

Gamemaker Language (GML)

GML ist eine imperative Scriptsprache, die sich syntaktisch an Javascript bzw. C-ähnlichen Sprachen orientiert. Allerdings gibt es für viele Befehle mehrere Alternativen, sodass man je nach Background im eigenen Stil programmieren kann. Beispielsweise können die logischen Verknüpfungen *Und*/*Oder* sowohl als &&/|| als auch im Stil von BASIC als *and*/*or* geschrieben werden.

Datentypen in GML

In GML gibt es im Grunde nur die Datentypen *double*, *String* und *Array*.[5]. Das heißt es gibt keinen eigenen Datentyp für Booleans. Dafür kann man jedoch Variablen vom Typ *double* direkt als boolsche Ausdrücke verwenden, wobei ein Wert kleiner gleich 0.5 *false* entspricht und darüber *true*. Man kann hierfür auch die Schlüsselwörter true und false verwenden, die *1* bzw. *0* entsprechen.

Variablen und Sichtbarkeit

Eine in meinen Augen sehr unpraktische Eigenheit in GML ist, dass Objektvariablen nicht explizit deklariert werden können. Die Deklaration von Objektvariablen erfolgt implizit bei der ersten Zuweisung einer noch nicht deklarierten Variable. Dies führt sehr leicht zu Fehlern, wenn man sich bei Variablennamen vertippt.

Lokale Variablen müssen mit dem Schlüsselwort *var* deklariert werden. Dieses Schlüsselwort wird sowohl für Strings als auch Zahlen verwendet.

Globale Variablen werden deklariert indem man an den Variablennamen vorne „global.“ anhängt.
z.B. `global.gameOver = false`

Für weitere Informationen gibt es eine ausführliche Dokumentation von GML unter
https://docs2.yoyogames.com/

[5] Genaugenommen gibt es auch noch Enumarationen und Pointer, auf die gehe ich aber hier nicht ein.