Ruprecht-Karls-Universität
Institut für Informatik
Sommersemester 2009
Fortgeschrittenenpraktikum
Betreuung: Prof. Dr. Barbara Paech

Praktikumsbericht zum Fortgeschrittenenpraktikum

Konzept für die Verwendung von GameMaker zur Vermittlung von Programmierkompetenz im Schulunterricht

David Grünbeck
Studiengang Lehramt
Informatik (4. Fachsemester)
E-Mail: gruenbeck@stud.uni-heidelberg.de

Inhalt

Legende

EA Einzelarbeit
F Frontalunterricht
PA Partnerarbeit
UG Unterrichtsgespräch
OO Objektorientierung

1. Einleitung

Ziel des in diesem Dokument vorgestellten Praktikums war die Entwicklung mehrerer Lehreinheiten zur Vermittlung von Programmierkompetenz, die statt einer konkreten Programmiersprache das Programm Game Maker (YoYo Games) verwenden. Dazu wurde zunächst in einer Literaturrecherche zusammengestellt, was wichtige Programmierkonzepte sind und wie sie typischerweise vermittelt werden. Daraufhin wurde eine Auswahl einiger Konzepte, auf denen die Lehreinheiten basieren sollen, getroffen. Schließlich wurde die Klassenstufe festgelegt, in der diese Konzepte vermittelt werden sollen. Die Ergebnisse dieser Recherche- und Auswahltätigkeit finden sich im ersten Teil der Arbeit.

Im weiteren Verlauf des Praktikums wurde dann ein Vorschlag für mehrere aufeinander abgestimmte Lehreinheiten erarbeitet, die die zuvor ausgewählten Programmierkonzepte mithilfe von Game Maker vermitteln sollen. Diese finden sich im zweiten Teil der Arbeit. Bei der Ausarbeitung wurde darauf geachtet, die Darstellung des Vorgehens im Unterricht detailliert zu gestalten, um auch Lehrern mit einem geringen Einarbeitungsgrad in die Entwicklungsumgebung Game Maker die Durchführung des hier vorgestellten Konzepts zu ermöglichen. Um dies weiterhin zu unterstützen, finden sich zudem zwei Anhänge: Der erste Anhang beschreibt mögliche technische Stolperfallen in der Benutzung von Game Maker und wie diese umgangen werden können. Der zweite Anhang beschreibt die im Zuge dieses Praktikums erstellten und gesammelten Materialen, die Schülern wie Lehrern sowohl das Verständnis des hier vorgestellten Konzepts als auch die Erweiterung ihrer eigenen Projekte erleichtern sollen.

Innerhalb der nächsten Wochen wird das hier vorgestellte Unterrichtskonzept noch praktisch evaluiert werden und die Ergebnisse der praktischen Evaluation in dieser Arbeit vermerkt.

1.1 Kerndaten des Konzepts

Die Kerndaten des entwickelten Konzepts finden sich in untenstehender Tabelle.

Klassenstufen	• 6 • 7
Dauer	• 6 x 90 Minuten
Benötigte Vorkenntnisse	• Keine Programmierkenntnisse: Die Lehreinheiten richten sich an Programmiereinsteiger ohne Vorkenntnisse. • Bedienerkenntnisse: Kenntnisse in der Bedienung von grafischen Benutzeroberflächen (z.B. Benutzung von Menüs, Eingabemasken, Drag and Drop) sind von Vorteil, können aber auch im Rahmen der Lehreinheiten vermittelt werden.
Lerninhalte	• Verzweigungen (Bedingte Anweisungen) • Objekt/Objekt-Instanz • Bezeichner (Name), Attribut (Eigenschaft), Methode (Fähigkeit) • Klasse/Objekt-Klasse (Objekttyp) • Variablen
Lernziele	• Siehe 1.2

1.2 Lerninhalte, Vermittlungsansätze und Bildungsplanbezug

Zu Beginn des Praktikums wurde zunächst in einer Literaturrecherche zusammengestellt, was wichtige Programmierkonzepte sind, wie diese typischerweise vermittelt werden und inwiefern sie in den Bildungsplänen von Baden-Württemberg und Bayern verankert sind. Die Rechercheergebnisse wurden zum einen in kompakter Form in untenstehender Tabelle und zum anderen ausführlicher reflektiert in den sich der Tabelle anschließenden Überlegungen festgehalten.

Stufe	Konzept
6-7 BY 11-12 BW	**Objektorientierung: Konzept** • Objekt • Bezeichner (Name), Attribut (Eigenschaft), Methode (Fähigkeit) • Attributzuweisung: Bezeichner.Attribut = Wert • Methodenaufruf: Bezeichner.Methode(Argument, Argument, …) • Klasse (Objekttyp) • Beziehungen zwischen Objekten (Enthält-Beziehung) • Teilobjektzugriff: Hauptobjekt.Teilobjekt (Boettcher, Grabowsky und Humbert) (Brichzin, Freiberger und Reinold 13f, 18-20, 24) (Diethelm) (Diethelm, Geiger und Zündorf 164) (Hubwieser 29f) (Ministerium für Kultus, Jugend und Sport des Landes Baden-Württemberg 440)
6-7 BY 11-12 BW	**Objektorientierung: Dokumentation** • UML-Objektsymbol • UML-Klassensymbol • UML-Klassendiagramm mit benannten Beziehungen (Brichzin, Freiberger und Reinold 44-48) (Diethelm, Geiger und Zündorf 164) (Hubwieser 29f) (Ministerium für Kultus, Jugend und Sport des Landes Baden-Württemberg 440)
6-7 BY 10-12 BW	**Algorithmen: Konzept** • Abfolge von Anweisungen (Sequenz) • Kapselung in Methoden (Brichzin, Freiberger und Reinold 127) (Ministerium für Kultus, Jugend und Sport des Landes Baden-Württemberg 313, 439)
6-7 BY 10-12 BW	**Algorithmen: Kontrollstrukturen** • Schleifen (Wiederholungen) mit fester Anzahl von Durchläufen • Schleifen (Wiederholungen) mit variabler Anzahl von Durchläufen und Bedingungen • Verzweigungen (Bedingte Anweisungen) (Brichzin, Freiberger und Reinold 117, 120, 124) (Ministerium für Kultus, Jugend und Sport des Landes Baden-Württemberg 313, 439)

Insgesamt ergibt sich bei der Auswahl der zu unterrichtenden Inhalte ein sehr homogenes Bild; es scheint Einigkeit darüber zu bestehen, welche Inhalte zum Erwerb von Programmierkompetenz im Informatikunterricht behandelt werden sollen. Uneinigkeit besteht hingegen darüber, welche Inhalte in welcher Klassenstufe behandelt werden sollen und insbesondere darüber, wie diese Inhalte vermittelt werden sollen und ob sich das objektorientierte Paradigma zum Einstieg in die Programmierung eignet.

Bei der Frage nach der geeigneten Klassenstufe scheint sich zunehmend die Meinung zu etablieren, dass bereits in der Unterstufe mit der Vermittlung einer objektorientierten Sicht begonnen werden kann, wenn auch in einer vereinfachten Form. So vertritt beispielsweise Frey die Ansicht, dass die objektorientierte Sichtweise der menschlichen Denkweise entgegenkommt, mahnt aber, dass aber die Modelle, die sie in Jahrgangsstufe 6 verwendet sehen möchte, vereinfacht sein sollen auf ein Abstraktionsniveau, das der Altersstufe angemessen ist. Auch Hubwieser ist der Meinung, dass Informatikunterricht möglichst früh einsetzen sollte, nicht erst in der gymnasialen Oberstufe. Der Modellierung wird bei ihm ein hoher Bildungswert zugesprochen; sie solle daher einen hohen Stellenwert im Unterricht einnehmen. Programmierung wird bei Hubwieser verstanden als die Umsetzung von vorher erarbeiteten Modellen, denn diese schulen Hubwieser zufolge ein tiefgreifendes, abstraktes Verständnis, das wichtiger ist als die „Benutzerfertigkeit“ eines konkreten Systems. Nichtsdestotrotz solle das abstrakte Verständnis an konkreten Systemen und Werkzeugen umgesetzt und dadurch geschärft werden, so Hubwieser, und dazu sollten Lernvorgänge wenn möglich projektorientiert durchgeführt werden, während Lehrervorträge nur punktuell eingesetzt werden sollten. Auch Hubwieser ist für eine „milde“ Form der Objektorientierung, die er objektorientierte Sichtweise nennt. Ihr ist ihm zufolge der Vorzug zu gewähren gegenüber tatsächlicher objektorientierter Programmierung, da bei dieser Programmierung eine Akkumulation vieler neuer, teils schwieriger Konzepte (Variable, Funktion, Zeiger/Referenz…). die Schüler leicht überfordern kann. Hubwieser schlägt merkt an, eine Black-Box-Sicht könne zur Vermeidung dieser Überforderung beitragen.

Auch Diethelm sieht Probleme, die sich für Einsteiger in die Objektorientierung stellen und die Lernhürden mit sich bringen. Dennoch habe die Objektorientierung lerntheoretische Vorteile, so Diethelm. Auf der Suche nach der Ursache dieses Widerspruchs kommt sie zu dem Schluss, die Vermittlung von Objektorientierung sei didaktisch schwierig und fehlerträchtig. Diethelm entwickelt deshalb ein Konzept, das zunächst die Modellierung betont, dann Objekte und anschließend Klassen einführt, daraufhin Vererbung diskutiert und schließlich Methoden behandelt. Diese von Diethelm vorgeschlagene Reihenfolge wird durch das in diesem Praktikum entwickelte Konzept fast genau eingehalten, nur werden Methoden (in Form mehrerer kombinierter Actions) behandelt, bevor der Thematik der Vererbung behandelt wird. Ein weiterer wichtiger Punkt Diethelms ist Gegenstand des hier vorgestellten Konzepts, und zwar die Ausführbarkeit: Für Diethelm ist die Anwendbarkeit der Modelle ein hoher Motivationsfaktor für die Lernenden: „Etwas zu schaffen, das man evtl. sogar mit nach Hause nehmen kann, um es seinen Eltern oder Freunden oder evtl. über das Internet der ganzen Welt zu zeigen […], ist eine der großen Chancen des Informatikunterrichts.“ (Diethelm 48) Diese Produktorientierung wird konsequent durch das hier präsentierte Konzept vertreten.

Genau wie Diethelm kommt auch Börstler auf Basis mehrerer Studien zu dem Schluss, dass die Probleme mit der frühen Einführung der Objektorientierung mit einem Mangel an geeigneten Werkzeugen und geeigneten pädagogischen Konzepten in Zusammenhang stehen. Seiner Ansicht nach ist an der frühen Einführung problematisch, dass viele grundlegende Konzepte miteinander verwoben sind und somit nicht mehr Schritt für Schritt aufeinander aufbauend gelehrt werden können wie bei der prozeduralen Programmierung. Börstler argumentiert ähnlich wie Diethelm für ein überarbeitetes didaktisches Konzept bei der Einführung der Objektorientierung; allerdings plädiert er nicht für eine besondere Abfolge von Inhalten, sondern für einfachere Sprachen und Werkzeuge im Zusammenhang mit der Objektorientierung. Seiner Ansicht zufolge helfen diese, eine kognitive Überbelastung der Lernenden zu vermeiden, indem sie die Komplexität des zu lernenden Stoffes reduziert. Das in dem hier vorgestellten Konzept verwendete Werkzeug Game Maker mit seinem Ansatz einer

visuellen Programmierung kann durchaus als solch ein Komplexität reduzierendes Werkzeug angesehen werden.

Eine solche visuelle Programmiersprache hilft laut Kohl Syntaxfehler zu vermeiden und reduziert somit Hürden und Verunsicherungen beim Einstieg in die Programmierung. Visuelle Programmiersprachen, so Kohl, helfen, auf einfache Weise allgemein gültige Konstrukte kennenzulernen, welche dann später auf textuelle Programmiersprachen übertragen und dort anwendet werden können. Sie helfen damit, dem Problem zu begegnen, dass professionelle Programmiersprachen insbesondere bei Anwendung des objektorientierten Paradigmas häufig komplex sind und viel Zeit zum Erlernen benötigen, weshalb sie zum Einstieg nur beschränkt geeignet sind, wie Brändle et al. bemerken. Ihre Forderungs lautet deshalb: „Die Schüler müssen mit wenig Aufwand in kurzer Zeit erste lauffähige Programme erstellen können." (Brändle, Hartmann und Nievergelt 201). Auch dieser Forderung kommt das hier vorgestellte Unterrichtskonzept nach. Bereits innerhalb der ersten zwei Lehreinheiten haben die Schüler ein ausführbares Programm vorliegen, das dann Zug um Zug erweitert wird.

Da das in diesem Dokument vorgestellte Unterrichtskonzept nur 6 Doppelstunden umfasst, musste eine Auswahl aus den oben erwähnten Lerninhalten getroffen werden. Da sich das Unterrichtskonzept an Programmiereinsteiger richtet, handelt es sich um die grundlegendsten Eigenschaften der Objektorientierung. Im Detail sollen folgende Lerninhalte innerhalb der 6 Doppelstunden vermittelt werden:

- Objekt/Objekt-Instanz;
- Bezeichner (Name), Attribut (Eigenschaft), Methode (Fähigkeit);
- Klasse/Objekt-Klasse (Objekttyp);
- Verzweigungen (bedingte Anweisungen);
- Variablen.

1.3 Konkrete Lernziele

Nach der Festlegung der Lerninhalte wurden diese operationalisiert, d.h. die Lerninhalte wurden in konkrete Lernziele überführt, die als beobachtbares Verhalten, erlangte Fähigkeiten und gewonnenes Wissen der Schülerinnen und Schüler messbar sind. Die angestrebten Lernziele sind die im Folgenden aufgezählten.

Die Schülerinnen und Schüler sollen nach Abschluss der Lehreinheiten in der Lage sein:

- die zentralen Ziele eines Programmierprojekts festzulegen;
- die zugrundeliegenden Regeln eines Spiels zu dokumentieren;
- ein Spiel in einzelne kleinere Elemente (Objekte) zu zerlegen;
- eine Online-Hilfe zur Informationssuche zu benutzen;
- eine Modellierung (Game Design Document) in eine Implementierung zu überführen (Objekte zu definieren und deren Eigenschaften /Attribute festzulegen);
- Objektklassen von Objektinstanzen zu unterscheiden;
- eine 2D-Grafik auf dem Bildschirm darzustellen;
- eine 2D-Grafik auf dem Bildschirm zu positionieren;
- Fähigkeiten (Methoden) von Objekten zu programmieren;
- das Verhalten von Objekten zu kontrollieren;
- Vererbungsstrukturen zu nutzen, um gleichartige Objekte einfacher zu programmieren;
- abstrakte Kontrollobjekte zu programmieren;
- die Lebensdauer von Objektinstanzen zu begrenzen;
- Variablen zu benutzen um Daten zu speichern;
- bestimmte Anweisungen nur unter bestimmten Bedingungen auszuführen (bedingte Anweisung und Verzweigung).

2. Unterrichtsaufbau

2.1 Einheit A: Administratives, Ableitung des Game Designs

Kerndaten Einheit A

Dauer: 90 Minuten

Inhalte

- A1 Administratives | F | 15 min.
- A2 Durchspielen des Prototyps | EA/PA | 10 min.
- A3 Ableitung des Game Design Documents | EA/PA | 45 min.
- A4 Konsolidierung des Game Design Documents | UG | 30 min.

Hilfsmittel

- Mehrere Schüler-PCs mit Microsoft Windows, auf denen der Prototyp des Spiels als ausführbare Datei vorhanden ist
- Für jeden Schüler 1 Konzeptblatt Game Design Document
- Für jeden Schüler 1 Projektmappe (Reinschrift)
- Tafel

Lernziele

„In dieser Einheit lernt ihr…

- … die zentralen Ziele eines Programmierprojekts festzulegen
- … die zugrundeliegenden Regeln eines Spiels zu dokumentieren
- … ein Spiel in einzelne kleinere Elemente (Objekte) zu zerlegen“

A1 Administratives | F | 15 min.

Zunächst sind 15 Minuten Unterrichtszeit für allgemeine administrative Aufgaben eingeplant wie z.B.:

- Begrüßung
- Vorstellung Lehrer
- Vorstellung Schüler
- Vorkenntnisse Schüler
- Erwartungen Schüler
- Vorstellung Fahrplan, Projektmappe

Ein Beispiel für die Vorstellung der Projektmappe könnte sein:
L: In der Projektmappe sammeln wir Informationen, die wir uns schrittweise erarbeitet haben und auf die wir später wieder zurückgreifen müssen, falls wir sie vergessen haben. Sie dient zur Dokumentation unserer Arbeit für uns. Bei großen Spielen ist eine solche Dokumentation wichtig, weil viele Mitarbeiter gleichzeitig an einem Spiele-Projekt arbeiten und Mitarbeiter möglicherweise hin und wieder auch wechseln und sich deshalb in das Spiel einarbeiten müssen.

- Regeln Computerraum, Essen und Trinken

A2 Durchspielen des Prototyps | EA/PA | 10 min.

Nach den administrativen Aufgaben beginnt die eigentliche Projektarbeit. Hierzu wird den Schülern ein fertiges Spiel vorgestellt, das vom gleichen Typ ist wie das zu erstellende Spiel (Labyrinth-Spiel). Dieses Spiel findet sich in den Materialien im gleichen Verzeichnis wie dieses Dokument; es hat den Dateinamen `RP.exe`. (RP steht für „Rettet Percy“, den Titel des Spiels.)

Die Schüler werden gebeten, das Spiel auf eigene Faust für einige Minuten zu spielen und es zu erkunden. Wenn nötig, kann der Lehrer das Spiel mit Hilfe eines Lehrer-PCs und eines Projektors erklären (nicht erwartet).

L: Auf euren PCs ist ein Prototyp unseres Spiels installiert. Das heißt es handelt sich um eine Version, die so ähnlich aussieht wie das Spiel, das wir am Ende des Projekts programmiert haben wollen. Der Prototyp ist aber noch nicht fertig. Das ist gut, denn das heißt ihr dürft im Laufe des Projekts später noch eigene Ideen einbringen und den Prototyp erweitern und verbessern.

L: Damit ihr diese Art von Spiel besser kennen lernt, schlage ich vor, dass wir es einfach einmal alle einige Minuten lang spielen. Dabei lernt ihr die Regeln des Spiels kennen und einige der Objekte, aus denen sich das Spiel zusammensetzt. Nachher schauen wir uns das dann noch einmal im Detail an.

Didaktische Überlegungen zu A1

Ein komplett eigenständiges Game Design, also der Entwurf eines eigenen neuartigen Typs von Spiel, wird im Rahmen des Projekts nicht angestrebt. Diese Entscheidung wurde in dieser Form getroffen, weil sich das Projekt an Programmieranfänger richtet und demzufolge in den ersten Stunden nicht das Game Design, sondern das Erlernen von Programmiertechniken zur Erstellung eines Spiels im Vordergrund stehen soll. Raum für Kreativität im Game Design wird in den späteren Stunden eingeräumt, wenn das Spiel auf eigene Faust um weitere Eigenschaften erweitert werden darf.

Den Schülern wird durch die Demonstration des Spiels ein Produkt vor Augen geführt, dessen Erstellung als Ziel des Projekts angestrebt wird. Durch die Demonstration wird also eine konkrete, praktische, klar umrissene Zielvorgabe gegeben, die die Schüler motivieren soll. Die Erstellung eines Spiels wird durch diese Art der Einleitung zum zentralen Problem des Projekts, das zu lösen ist. Die zur Lösung des Problems benötigten theoretischen Programmiertechniken und -konzepte (Algorithmenstrukturen, Objektorientierung) werden also vom Fokus der Wahrnehmung in die Peripherie gedrängt. Sie sind dann nur noch ein Nebenprodukt bzw. ein Mittel zur Lösung des Problems „Wie erzeuge ich ein Spiel“. Dieser Effekt ist gewünscht, denn er bewirkt bei den Schülern eine realistische Wahrnehmung von Programmierung, denn Programmierung ist außerhalb von Unterricht fast nie Selbstzweck, sondern dient in der Regel der Lösung eines Problems der realen Welt mit Hilfe eines Computers.

Die Schüler lernen durch die Demonstration des Spiels zudem die Regeln des Spiels kennen. Sie bekommen ein Gefühl für die Ereignisse und Bedingungen, die sie später definieren müssen und lernen die möglichen Eingaben des Benutzers an das Spiel kennen. Sie bekommen eine erste Vorstellung davon, welche Objekte an dem Spiel beteiligt sind.

A3 Ableitung des Game Design Documents | EA/PA | 45 min.

Die Schüler werden gebeten, auf einem Konzeptblatt (das vom Lehrer verteilt wird) in Einzelarbeit oder Partnerarbeit das Game Design Document aus der Basis des gespielten Spiels ableiten. Das Game Design Document ist aufgeteilt in verschiedene Bereiche. Diese Bereiche können im Anhang in der Projektmappe eingesehen werden. Erwartete Schüler-Ergebnisse des Entwurfs sind im nächsten Punkt aufgeführt.

L: Ihr habt jetzt eine bessere Vorstellung davon, wie das Spiel funktioniert: Welche Regeln es hat, welche Figuren vorkommen und wie man die Figuren steuert. Bevor wir uns aber auf das Programmieren stürzen, sollten wir alle sicherstellen, dass wir die gleiche Vorstellung davon haben, wie das Spiel funktionieren soll und welche Elemente wir programmieren müssen. Das legen wir mit dem sogenannten Game Design Document fest, dem Spielentwurfsdokument. Ihr findet einen Vordruck in eurer Projektmappe. Bevor wir den Vordruck in der Projektmappe ausfüllen, möchte ich euch bitten, erst einmal selbst / in Zweiergruppen ein Konzeptblatt auszufüllen – bitte notiert euch nur Stichworte zu jedem Punkt. Wir sammeln dann später eure Ideen, führen sie zusammen und füllen dann gemeinsam den Vordruck in der Projektmappe aus, so dass wir dort alle das gleiche stehen haben.

A4 Konsolidierung des Game Design Documents | UG | 30 min.

Die unten genannten erwarteten Schüler-Ergebnisse aus der vorherigen Phase sowie alle sich darüber hinaus im Unterrichtsgespräch entwickelnden Antworten werden zunächst an der Tafel festgehalten und nach der Entwicklung des Tafelanschriebs von den Schülern in die Projektmappe übertragen.

Objekte (beschreibt Objekte des Spiels, ihre Eigenschaften, ihr Verhalten zueinander):

Raum:	Aufenthaltsort der Figuren, begrenzt Spielfläche
Stein:	Bildet mit anderen Steinen zusammen Wände/Mauern, undurchdringlich für alle beweglichen Figuren
Schlüssel:	Beendet Level/Spiel, wenn Spieler ihn einsammelt
Tresorstein:	Verschließt Zugang zu Zielflagge, verschwindet nach Einsammeln des Tresoröffners
Tresoröffner:	Verschwindet bei Berührung mit Spieler, lässt Tresorsteine verschwinden
Diamant:	Verschwindet bei Berührung mit Spieler, ergibt Punkte
Monster:	Laufen nach links/rechts durch den Raum, verletzen Spieler bei Berührung, geben Punktabzug, starten Raum neu
Spieler:	Läuft nach links/rechts/oben/unten durch den Raum, gesteuert durch Tastatureingabe von Spieler.

Klänge (beschreibt Klänge des Spiels, ihr Einsetzen bei bestimmten Ereignissen):

Ton 1:	Ton beim Einsammeln eines Diamanten
Ton 2:	Ton beim Einsammeln eines Schlüssels
Ton 3:	Ton beim Berühren eines Steins
Ton 4:	Ton beim Öffnen des Tresors
Ton 5:	Ton beim Berühren eines Monsters

Steuerung (beschreibt Eingabemöglichkeiten des Spielers, Reaktion der Objekte darauf):

Rechts-Taste:	Spielerfigur bewegt sich nach rechts
Links-Taste:	Spielerfigur bewegt sich nach links

Oben-Taste:	Spielerfigur bewegt sich nach oben
Unten-Taste:	Spielerfigur bewegt sich nach unten
Keine Taste:	Spielerfigur bleibt stehen

Spielfluss (beschreibt Startzustand, Endzustand des Spiels, Punktevergabe):

Spielstart:	Spielerfigur befindet sich an festgelegter Stelle im Raum / Monster (falls im Raum vorhanden), bewegen sich umher
Punkte:	Für jeden gesammelten Diamanten gibt es +10 Punkte / für jedes berührte Monster gibt es -20 Punkte
Spielende:	Spielerfigur erreicht letzte Zielflagge im letzten Raum

Didaktische Überlegungen zu A4

Aufgrund dieser hohen Schreibintensität dieser Tätigkeit sind für diesen Teil der Einheit 30 Minuten veranschlagt. Es wird erwartet, dass der hohe Aufwand für diese Tätigkeit sich insofern rechnet, als die Schüler mit einem gestärkten Wissen über die zu programmierenden Objekte und die Regeln des Spiels aus der Übung gehen werden und sich somit zeitintensive Fragen zu diesen Feldern später nicht oder nur in geringem Maß ergeben werden.

Nach der Entwicklung des Game Design Documents ist der Rahmen gesteckt für die Entwicklung des Spiels. Die Schüler haben den Prototyp des Spiels analysiert und beschrieben. Sie sind vertraut mit den Objekten und den Regeln des Spiels sowie den Eingabemöglichkeiten des Benutzers. Sie haben ihre Kenntnisse in der Projektmappe dokumentiert, so dass ein Rückgriff auf die Erkenntnisse in den nächsten Sitzungen möglich ist. In den folgenden Einheiten Computergrafik und Programmiergrundlagen soll technisches Grundlagenwissen geschaffen werden in Hinblick auf Grafikdarstellung am Computer und den grundlegenden Strukturen, die Algorithmen bereitstellen müssen. Dieses technische Grundlagenwissen wird benötigt, damit die Schüler später Probleme wie Figurenpositionierung in Räumen und die Koordination von Events und Actions effektiv in Angriff nehmen zu können.

Erfahrungen aus der Unterrichtspraxis zu Einheit A

(Folgen nach Durchführung)

2.2 Einheit B: Einführung in Game Maker, Objekte, Objekt-Eigenschaften (Attribute)

Kerndaten Einheit B

Dauer: 90 Minuten

Inhalte

- B1 Einführung in die Game Maker-Oberfläche | F | 5 min.
- B2 Implementierung Modellierung, OO: Eigenschaften/Attribute | UG, EA/PA | 55 min.
- B3 Erstellen & Möblieren des ersten Raums, OO: Instanzen | UG, EA/PA | 30 min.

Hilfsmittel

- Mehrere Schüler-PCs mit Microsoft Windows, auf denen Game Maker 7 installiert ist
- Lehrer-PC, der an Projektor angeschlossen ist
- Tafel

Lernziele

„In dieser Einheit lernt ihr…

- … eine Online-Hilfe zur Informationssuche zu benutzen
- … eine Modellierung (Game Design Document) in eine Implementierung zu überführen (Objekte zu definieren und deren Eigenschaften /Attribute festzulegen)
- … Objektklassen von Objektinstanzen zu unterscheiden“

B1 Einführung in die Game Maker-Oberfläche | F | 5 min.

- Lehrer öffnet Game Maker 7 auf dem Lehrer-PC, der an den Projektor angeschlossen ist.
- Lehrer verbalisiert jeden Schritt, den er als nächstes durchführt.
- Lehrer öffnet die Datei `RP1.gmk`, damit Ressourcen in die Oberfläche geladen werden und die Erklärungen anschaulicher erscheinen.
- Lehrer erklärt die beiden wichtigsten Bereiche der Oberfläche (siehe Abbildung 1: Die Oberfläche von Game Maker):
    - Bereich 1: Die Seitenleiste/Ressourcenliste. Resourcen sind sämtliche Bestandteile, aus denen das Spiel besteht: Grafiken, Töne, Objekte, Räume usw.
        - Rooms: Hier werden die Räume/Levels verwaltet, in denen sich die Figuren bewegen.
        - Objects: Hier werden die Objekte (Figuren, Gegenstände, aber auch unsichtbare Wächterobjekte) verwaltet, die sich in den Räumen befinden, bewegen oder sie kontrollieren.
        - Sprites: Hier werden die Grafiken verwaltet, die Objekte am Bildschirm darstellen.
        - Backgrounds: Hier werden die Grafiken verwaltet, die den Hintergrund von Räumen darstellen.
        - Sounds: Hier werden sämtliche Musik und alle Toneffekte des Spiels verwaltet.
        - Game Information: Hier kann ein Hilfesystem für den Spieler erstellt werden.

- Global Game Settings: Hier wird festgelegt, wie das Spiel mit dem Computer/Betriebssystem kommuniziert (Dateisystem, Fenstereigenschaften, Prozesspriorität, Fehlermeldungen, Startbildschirm).

- Bereich 2: Die Symbolleiste.
  - Spiel laden.
  - Spiel speichern.
  - Spiel ausführen ( ). Kompiliert das Spiel im Hintergrund (macht eine ausführbare Datei daraus), speichert die Datei temporär ab und führt sie aus. Die temporäre Datei wird nach Ende des Spiels wieder gelöscht. Diese Funktion ist dazu gedacht, das Spiel in der Entwicklungsphase zu testen.
  - Spiel in eine ausführbare Datei kompilieren ( ). Kompiliert das Spiel und speichert es als ausführbare Datei ab. Das Spiel wird nicht gestartet.
  - Neue Ressourcen erstellen (Icons von bis). Mit diesen Symbolen können neue Ressourcen zum Spiel hinzugefügt werden.
    - Alternative 1: Rechtsklick auf einen der Ordner in der Ressourcenliste > `Create`....
    - Alternative 2: Menü `Resources` > `Create`....
    - Alternative 3: Tastenkürzel wie im Menü `Resources` angegeben.

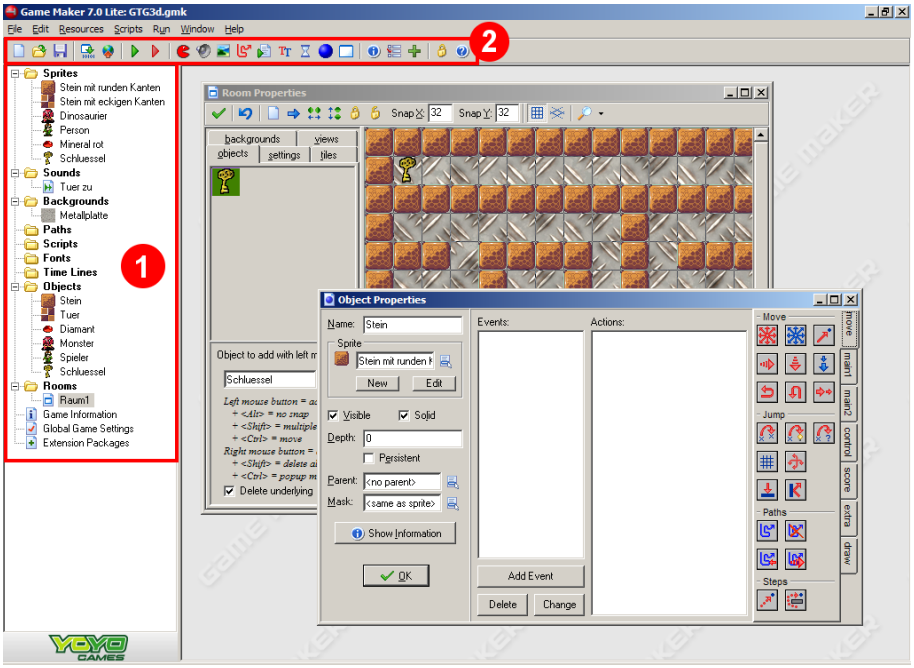


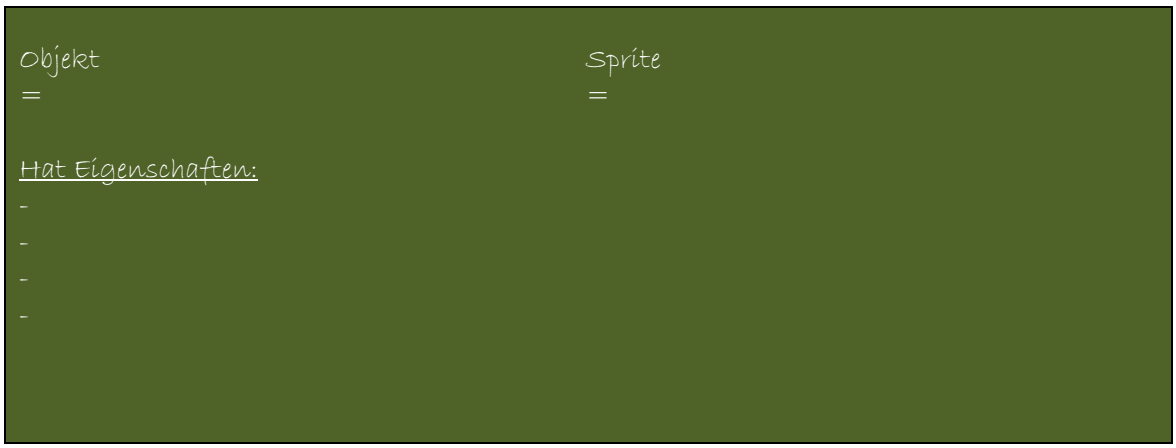
Abbildung 1: Die Oberfläche von Game Maker

Didaktische Überlegungen zu B1

Mit der Einführung in die Oberfläche wird bei den Schülern das Grundwissen sichergestellt, das nötig ist, um die Software im Folgenden zügig und problemlos nutzen zu können. Durch Frontalunterricht wird sichergestellt, dass hierbei keine Verunsicherungen bei den Schülern entstehen, dass keine wesentlichen Elemente ausgelassen werden und dass die Einführung ohne große Verzögerungen für den folgenden Unterrichtsverlauf durchgeführt werden kann.

B2 Implementierung Modellierung, OO: Eigenschaften/Attribute | UG, EA/PA | 55 min.

- Lehrer schließt geöffnetes Spiel in Game Maker.
- Lehrer bittet Schüler, das Game Design Document aus A4 zur Hand zu nehmen.
- Lehrer merkt an, dass jetzt die Objekte aus dem Game Design Document in Game Maker erstellt werden sollen.
- Lehrer fragt, welche Arten von Resourcen für die im Game Design Document unter „Objekte“ genannten Objekte in Frage kommt.
- Schüler-Antworten (Zielsetzung):
  - „Sprites“,
  - „Objects“,
  - „Rooms“.
- Lehrer erstellt am Lehrer-PC ein neues leeres Objekt
  - Hauptfenster > Symbolleiste >  .
  - Neues Fenster öffnet sich, in Ressourcenliste erscheint neues Objekt unter „Objects“.
- Lehrer merkt an, dass eine Vielzahl von Einstellungen für ein Objekt vorgenommen werden können und dass in den Einstellungen sicherlich viele den Schülern unbekannte Begriffe und Eigenschaften vorkommen.
- Lehrer bittet Schüler, die Online-Hilfe des Programms zu benutzen, um unbekannte Begriffe und Eigenschaften, die im Objekt-Fenster angezeigt werden, nachzuschlagen.
  - Menü `Help` > `Contents` > *`Index`* oder Taste *F1* > *`Index`*.
  - Bei Problemen mit der englischen Beschreibung kann den Schülern auch die semi-offizielle deutsche Übersetzung der Hilfedatei aus der älteren Game Maker-Version 6.1 angeboten werden, siehe beigefügtes Material.
  - Es steht dem Lehrer frei, den Schülern entweder den Suchbegriff direkt zu nennen oder die Findung des richtigen Suchbegriffs den Schülern selbst zu überlassen. Unter dem Plural „Objects“ (englischsprachige Hilfe) bzw. „Objekte“ (deutschsprachige Hilfe) ist der entsprechende Eintrag zu finden.
- Lehrer bittet Schüler, Ausschnitte aus dem Artikel zu „Objects“ in der Online-Hilfe zu lesen, und zwar die ersten beiden Absätze sowie die drei letzten Absätze. (10 min.) Die Schüler sollen dabei auf folgende Dinge achten:
  - Was ist ein Objekt in Game Maker?
  - Was ist ein Sprite?
  - Welche vier Eigenschaften kann ein Objekt in Game Maker haben?
- Lehrer bereitet während der Recherche der Schüler folgenden Tafelanschrieb vor:



- Schüler-Antworten(Zielsetzung), an Tafel festgehalten:

Objekt	Sprite
= Ding/Gebilde/Einheit im Spiel	= Darstellung/Grafik von Objekt
Hat Eigenschaften:	
- Name	
- Sprite	
- ± Sichtbar	
- ± Massiv	
Hat Verhalten	

- Zusätzlich sollten folgende Informationen festgehalten werden:

Objekt	Sprite
= Ding/Gebilde/Einheit im Spiel	= Darstellung/Grafik von Objekt
Hat Eigenschaften:	Hat Eigenschaften:
- Name	- ± Transparent
- Sprite	…
- ± Sichtbar	
- ± Massiv	
Hat Verhalten	Hat KEIN Verhalten

- Lehrer klärt mit Schülern die Bedeutung der Objekt-Eigenschaften „sichtbar“ („visible“) und „massiv“ („solid“).
    - Die Sichtbarkeit regelt, ob das Sprite eines Objekts bei der Platzierung in einem Raum angezeigt wird oder unsichtbar bleibt.
    - Die Massivität regelt, ob ein Objekt für andere Objekte durchdringbar sein soll. Dies ist z.B. wichtig bei der Planung von Bewegungen durch Kollisionen.
- Lehrer merkt an, dass von diesen an der Tafel festgehaltenen Informationen nun einige in die Projektmappe übernommen werden sollen, und dass diese im weiteren Verlauf vervollständigt werden.
    - Die Informationen sollen bei „Objekt-Klassen, Objekt-Instanzen, Sprites“ eingetragen werden.
    - Der Begriff „Objekt“ heißt in der Projektmappe „Objekt-Klasse“.
- Lehrer bittet Schüler, in die Projektmappe folgende Informationen zu übernehmen:
    - Bei „Objekt-Klasse“ soll ein Haken bei „Hat Eigenschaften“ und „Hat Verhalten“ gemacht werden.
    - Bei „Objekt-Klasse“ sollen als Beispiele für Eigenschaften eingetragen werden: „± sichtbar (visible)“ und „± massiv (solid)“.
    - Bei „Sprite“ soll ein Haken bei „Hat Eigenschaften“ gemacht werden.
    - Bei „Sprite“ soll als Beispiel für Eigenschaften eingetragen werden: „± transparent“.
    - Es steht dem Lehrer frei, diese Informationen selbst zu nennen oder die Schüler sie selbstständig finden zu lassen.

- Lehrer kündigt an, dass nun die Objekte aus dem Game Design Document erstellt werden sollen und dass dazu noch eine Anmerkung zur Benennung von Sprites gemacht werden soll
- Lehrer öffnet nacheinander die folgenden Dateien und macht dazu folgende Anmerkungen:
  - `RP1.gmk`
    Sprites können z.B. genau wie die dazugehörigen Objekte genannt werden. Das ist allerdings problematisch, falls ein Sprite einmal zu mehreren Objekten gehören sollte.
  - `RP1b.gmk`
    Sprites können z.B. auch eine beliebige Bezeichnung haben. Das ist allerdings unübersichtlich.
  - `RP1c.gmk`
    Sprites können z.B. auch eine Bezeichnung haben, die allgemein benennt, was das Sprite darstellt, gefolgt von z.B. einer Nummerierung, wenn mehrere ähnliche Sprites existieren (z.B. „Stein1“, „Stein2“, …).
  - `RP1d.gmk`
    Sprites können z.B. aber auch eine Bezeichnung haben, die speziell benennt, was das Sprite darstellt (z.B. „Stein mit runden Kanten“, „Stein mit eckigen Kanten“, …). Dies ist die hier bevorzugte Darstellung, wie sie im Rahmen dieses Projektes weiter verwendet werden wird.
- Lehrer bittet Schüler, die Objekte aus dem Game Design Document mit Ausnahme des Raums zu erstellen und ihnen Sprites zuzuweisen. Dabei soll der Import von Sprites von innerhalb des Fensters, in dem die Objekteigenschaften definiert werden, ausgelöst werden (siehe Abbildung 2: Erstellung eines Sprites für ein Objekt).
  - Stein – mit Eigenschaft „solid“,
  - Schlüssel,
  - Tür – mit Eigenschaft „solid“,
  - Diamant,
  - Monster,
  - Spieler.
- Lehrer geht an den einzelnen Schülerrechnern vorbei und hilft bei eventuellen Problemen.

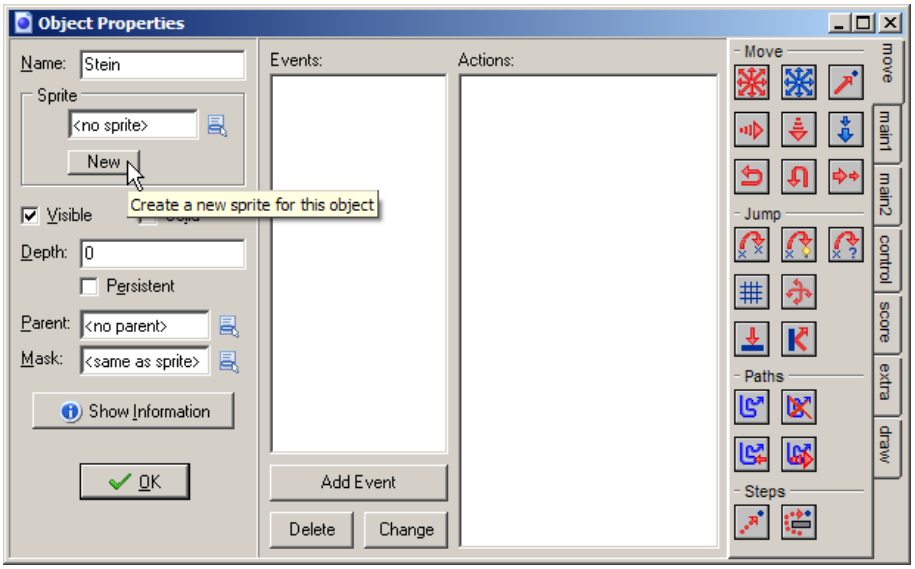


Abbildung 2: Erstellung eines Sprites für ein Objekt

Didaktische Überlegungen zu B2

Selbständige Suche in der Onlinehilfe
Die selbständige Suche der Schüler in der Onlinehilfe soll den Anstoß geben zum „Lernen lernen“. Die Schüler sollen selbständig benötigte Informationen suchen, statt das Programm vom Lehrer erklärt bekommen. Dies hat den Vorteil, dass die Schüler dadurch in die Lage versetzt werden, sich selbst in dieses und andere unbekannte Programme einzuarbeiten und künftig in diesem und anderen Programmen selbständig Hilfe zu suchen. Den Schülern wird somit Hilfe zur Selbsthilfe gegeben. Da Online-Hilfen in nahezu allen Programmen vorhanden sind, handelte sich hierbei um einen zeitstabilen Lerninhalt. Problematisch dabei ist allerdings, dass die Online-Hilfe für Game Maker ausschließlich in Englisch vorhanden ist und der Lernstand der Schüler in Englisch nicht Gegenstand von Informatik sein darf und dass der Erfolg der Schüler im Projekt und/oder bei einer Bewertung hier nicht von ihren Englisch-Kenntnissen abhängen sollte. Als Abhilfe ist möglich, ein gemeinsames Lesen der Hilfe mit Übersetzungshilfe anzudenken oder die deutsche Onlinehilfe für die alte Game Maker-Version 6.1 zur Verfügung stellen, die von einer Privatpersonen in einer Community angefertigt wurde und vom Hersteller von Game Maker zwar wohlwollend beachtet, aber nicht offiziell ins Programm integriert wird. Es ist anzumerken, dass die deutsche Onlinehilfe nicht durchgängig gleich gut übersetzt ist. Teilweise finden sich in ihr Umgangssprache, Grammatik- und Rechtschreibfehler oder mehrdeutige Begriffswahl (z.B. „entities“ übersetzt mit „Einträge“ statt mit „Dinge/Gebilde/Einheiten“).

Erst Objekte, dann Sprites
Der Unterschied zwischen Sprites und Objekten in Game Maker ist möglicherweise für Anfänger nicht auf Anhieb einfach zu verstehen. Anzunehmen ist, dass einige Schüler intuitiv das Sprite als das Objekt wahrnehmen könnten, also die grafische Repräsentation des Objekts mit dem eigentlichen Objekt gleichsetzen könnten. Wichtig ist aber zu verstehen, dass das eigentliche Objekt in Game Maker eine abstrakte Einheit ohne Grafik ist, der dann eine Grafik als Eigenschaft zur Repräsentation am Bildschirm zugewiesen wird. Aus diesem Grund soll zuerst das Objekt erstellt werden und ihm dann ein Sprite zugewiesen werden und nicht umgekehrt.

Namensgebung Sprites
Außerdem wird Wert darauf gelegt, dass die Sprites einen anderen Namen besitzen sollen als die Objekte, um den Unterschied zwischen beiden zu verdeutlichen. Deshalb wird in diesem Projekt auch vorgeschlagen, dass Sprites einen Namen besitzen sollten, der sich stärker auf ihr Aussehen als auf ihre Funktion bezieht (z.B. die Sprite-Bezeichnung „Dinosaurier“ für das Sprite des Objekts „Monster“ oder die Sprite-Bezeichnung „Stein mit eckigen Kanten“ für das Sprite des Objekts „Tür“).

Game Maker-Objekte als Klassen
Game Maker-„Objects“ werden in diesem Projekt als Klassen angesehen und behandelt, auch wenn sie selbst in der Innensicht bzw. der Binnenverwaltung des Programms Game Maker höchstwahrscheinlich eher Instanzen einer dem Benutzer verborgenen Klasse sind, die alle Einträge im Ordner „Objects“ definiert (diese Vermutung liegt z.B. allein schon deshalb nahe, weil innerhalb eines Spiels mehrere Objekte vom gleichen Namen existieren können). Dies betrifft jedoch nur die Binnenmodellierung von Game Maker und nicht die Modellierung der Spiele, die mit Game Maker erstellt werden. Für den Benutzer erscheinen die „Objects“, die er mit Game Maker erstellen kann als etwas, das Klassen sehr ähnlich ist. Einschränkend muss zwar erwähnt werden, dass ein Benutzer mithilfe der grafischen Benutzeroberfläche solch einer Klasse bzw. „Object“ keine weiteren Attribute hinzufügen kann. Allerdings kann der Benutzer jeder Klasse bzw. jedem „Object“ mithilfe der grafischen Benutzeroberfläche neue Actions und damit etwas Ähnliches wie Methoden hinzugefügen. Durch die Betrachtung von Game Maker-„Objects“ als Klassen kann der

Unterschied zwischen Klassen und Instanzen relativ anschaulich vermittelt werden: Die Game Maker-Objekte, die als Resource definiert werden und in der Seitenleiste im Ordner „Objects“ aufgelistet werden, sollen als Klassen verstanden werden, also als *Beschreibung/Modell* eines Steins, einer Tür, eines Schlüssels, etc. in der Mini-Welt des Spieles, das programmiert wird. Diese Beschreibungen/Modelle werden dann durch (ggfs. mehrfache) Platzierung in Räumen (Steine, Monster) zu einer *Realisierung* eines Steins, einer Tür, eines Schlüssels, etc., also quasi in der Welt des Spieles „materialisiert“ und somit für den Spieler, der das Spiel später spielen wird, wahrnehmbar. Ein kombiniertes Klassen- und Objektdiagramm, das dies darstellt, findet sich untenstehend (Abbildung 3: Objektdiagramm für einige Klassen und Instanzen des Spiels).

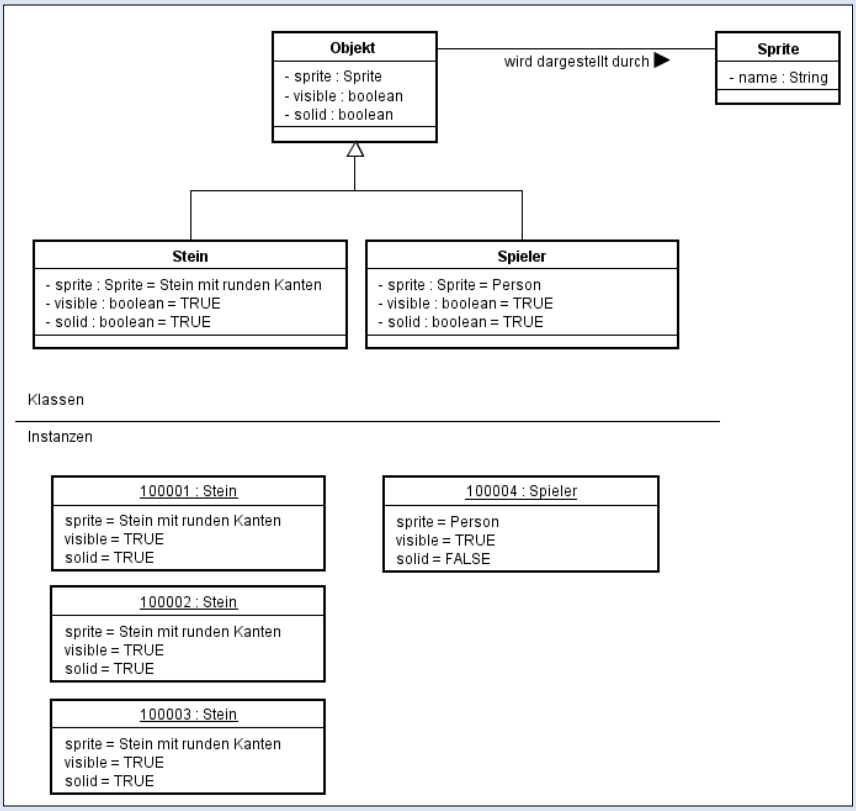


Abbildung 3: Objektdiagramm für einige Klassen und Instanzen des Spiels

Zu „Abbildung 3: Objektdiagramm für einige Klassen und Instanzen des Spiels“ noch ein Hinweis: Die Vergabe von eindeutigen ganzzahligen IDs für Instanzen, die bei der Platzierung in einem Raum erstellt werden, erfolgt automatisch und inkrementell durch Game Maker und beginnt mit der Zahl 100001. Mithilfe dieser eindeutigen IDs ist jede Instanz z.B. durch die Game Maker Language (GML) individuell ansprechbar.

B3 Erstellen und Möblieren des ersten Raums, OO: Instanzen | UG, EA/PA | 30 min.

- Schüler werden gebeten, einen neuen Raum zu erstellen mit folgenden Eigenschaften (festgelegt im Fenster `Room Properties`, das sich nach dem Erstellen eines neuen Objekts öffnet):
  - Snap X: *32*,
  - Snap Y: *32*,
  - Registerkarte `settings` > `Name`: *Raum1*,
  - Registerkarte `settings` > `Width`: *640*,
  - Registerkarte `settings` > `Height`: *512*,
  - Registerkarte `backgrounds` > *Sandstein*,
  - Registerkarte `backgrounds` > *[ ] Draw background color*,
  - Registerkarte `objects` > `Object to add...`: *Stein*.
- Schüler werden gebeten, durch Drücken von *Shift + linke Maustaste* eine Mauer um die Kanten des Raums zu erstellen, die den Raum begrenzen sollen.
- Schüler werden gebeten, durch Drücken der linken Maustaste weitere Steine so hinzuzufügen, dass daraus ein Labyrinth im Raum entsteht (15 Minuten). Ein Beispiel hierfür findet sich in `RP2.gmk` sowie in „Abbildung 4: Fertig möblierter erster Raum“.
  - Lehrer macht darauf aufmerksam, dass die Schüler dabei häufig speichern sollen, um bei eventuellen Abstürzen nicht ihre Arbeit zu verlieren.
- Schüler werden gebeten, eine Spielerfigur und einen Schlüssel im Raum zu platzieren.
  - Registerkarte “objects” > Object to add...: Spielerfigur.
  - *Linke Maustaste*: Spielerfigur in eine Ecke des Labyrinths platzieren.
  - Registerkarte “objects” > Object to add...: Schlüssel.
  - *Linke Maustaste*: Schlüssel in eine von Spielerfigur weit entfernte Ecke des Labyrinths platzieren.

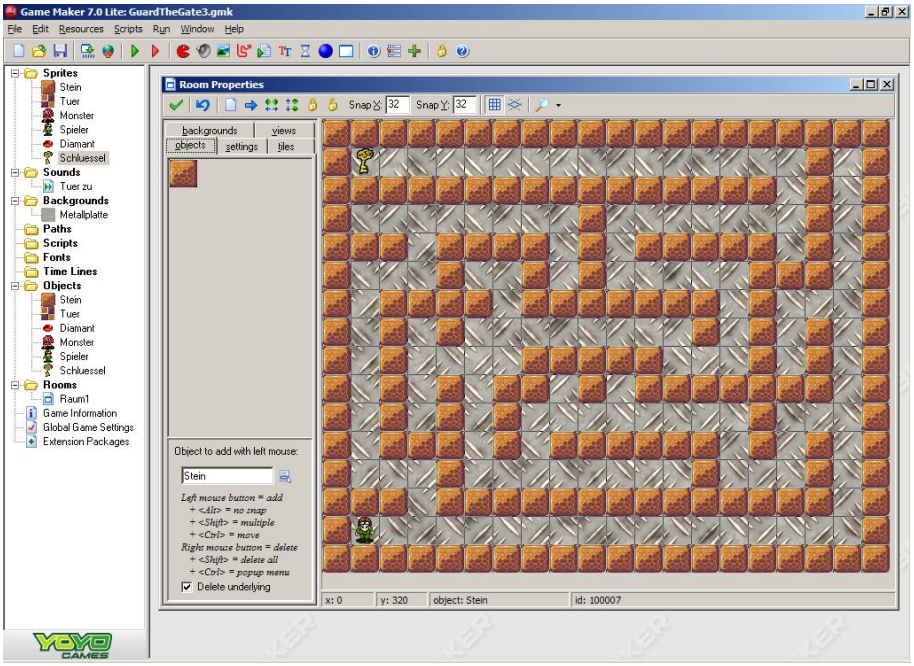


Abbildung 4: Fertig möblierter erster Raum

- Der Zwischenstand aus „Abbildung 4: Fertig möblierter erster Raum“ findet sich auch in der beigefügten Datei `RP2.gmk`.
- Lehrer macht darauf aufmerksam, dass nun viele Steine im Raum vorhanden sind, die alle gleich aussehen und alle die Eigenschaften haben, die vorhin definiert wurden.

- Lehrer bittet Schüler, in der Projektmappe das Bild mit den Plätzchen und das darunter befindliche Bildschirmfoto von Game Maker anzuschauen.
- Schüler werden gefragt, ob sie eine Ähnlichkeit zwischen dem Plätzchen-Bild und dem darunter befindlichen Bildschirmfoto erkennen können und wenn ja, welche.
- Schüler-Antworten (Zielsetzung):
    - Plätzchenform/Objekt-Klasse geben Form vor,
    - Plätzchen/Instanzen sind Kopien/Anfertigungen/Realisierungen der vorgegeben Form.
- Lehrer bittet Schüler, die Erkenntnis in der Projektmappe festzuhalten.
    - Bei „Objekt-Klasse“ wird eingetragen: „Objekt-Klasse = Beschreibung eines Dinges/Gebildes…“.
    - Bei „Objekt-Instanz“ wird eingetragen: „Objekt-Instanz = Realisierung eines Dinges/Gebildes…“.
    - Des Weiteren sollen bei „Objekt-Instanz“ nun angekreuzt werden „Hat Eigenschaften“ und „Hat Verhalten“, weil Objekt-Instanzen diese Merkmale von den Objekt-Klassen übernehmen. Dem Lehrer steht es frei, diese Tatsache von den Schülern selbst herleiten zu lassen. Bei den Beispielen für Eigenschaften und Verhalten bei „Objekt-Instanz“ wird dementsprechend eingetragen „genau wie Objekt-Klasse“.
    - Bei dem Beispiel für eine „Objekt-Klasse“ kann z.B. eingetragen werden „ein Stein“. Es bietet sich an, hier den unbestimmten Artikel zu verwenden.
    - Bei dem Beispiel für eine „Objekt-Klasse“ kann z.B. eingetragen werden „der Stein, der im ersten Raum in der linken unteren Ecke liegt“. Es bietet sich an, hier den bestimmten Artikel zu verwenden. Dem Lehrer steht es frei, diese Beschreibung von den Schülern herleiten zu lassen.

Erfahrungen aus der Unterrichtspraxis zu Einheit B
(Folgen nach Durchführung)

2.3 Einheit C: Einführung in 2D-Computergrafik

Einheiten C und D ergeben zusammen eine Sitzung mit 90 Minuten.

Kerndaten Einheit C

Dauer: 30 Minuten

Inhalte

- C1 Aufbau von Grafiken aus Pixeln | F, EA/PA | 15 min.
- C2 Aufbau von Grafiken aus Pixeln | EA | 15 min.

Hilfsmittel

- Mehrere Schüler-PCs mit Microsoft Windows, auf denen Game Maker 7 installiert ist
- LEGO-Set:
  - LEGO-Grundplatte,
  - LEGO-Steine Größe 2x2 und/oder
  - LEGO-Mosaic-Set
    - Transparente Grundplatte
    - 1x1 große Bausteine, die so gesteckt sind, dass sie eine Figur ergeben
- Tafel

Lernziele

„In dieser Einheit lernt ihr…

- … eine 2D-Grafik auf dem Bildschirm darzustellen
- … eine 2D-Grafik auf dem Bildschirm zu positionieren“

C1 Aufbau von Grafiken aus Pixeln | F, EA/PA | 15 min.

- Lehrer merkt an, dass als nächstes folgende Fragen beantwortet werden sollen:
  - Wie werden die Figuren vom Computer am Bildschirm dargestellt?
  - Wie wird eine 2D-Grafik (zweidimensionale Grafik) am Bildschirm aufgebaut?
- Schüler werden gebeten, in Game Maker den letzten Stand ihres Projekts zu laden, das Bild (Sprite) eines Steins aus dem Spiel zu öffnen und dieses im Sprite Editor stark zu vergrößern
  - Sprite durch Doppelklick aus der Ressourcenliste öffnen.
  - In der linken Hälfte des Fensters, das sich öffnet, `Edit Sprite` auswählen.
  - Doppelt mit der linken Maustaste auf `image0` klicken. Der Image Editor von Game Maker öffnet sich.
  - Im Image Editor so lange auf das Lupen-Symbol () klicken, bis das Bild sich nicht weiter vergrößert.
- Schüler werden gebeten, in der rechten Fensterhälfte in der Farbwahlfläche eine Signalfarbe (rot, magenta) auszuwählen und damit einige Male willkürlich in das Bild zu klicken. Es entstehen pixelgroße Farbflecken im Bild.
- Lehrer fragt, ob den Schülern hieraus ersichtlich wird, wie der Computer das Bild erzeugt.
- Schüler-Antwort (Zielsetzung):
  - Viele sehr kleine Bildpunkte (Pixel) ergeben, wenn sie sehr klein sind und sehr dicht beieinander liegen, für das Auge ein Bild.

C2 Aufbau von Grafiken aus Pixeln | EA | 15 min.

- Lehrer merkt an, dass als nächstes die folgende Frage beantwortet werden soll:
  - Wie kann die Position einer Figur in einem Raum bestimmt werden?
- Schüler werden gebeten, Lego-Grundplatte anzuschauen
  - Die Grundplatte soll einen Raum in Game Maker symbolisieren.
  - Die Grundplatte besteht aus einzelnen Noppen. Jede Noppe entspricht in unserem Versuch einem Bildschirmpunkt (Pixel).
  - Ein 2x2 großer LEGO-Stein oder ein LEGO-Mosaic auf transparenter Grundplatte symbolisiert ein Sprite.
  - Linke obere Ecke entspricht dem Nullpunkt/Koordinatenursprung
  - Vom Nullpunkt ab Pixel/Noppen nach rechts zählen = x-Wert.
  - Vom Nullpunkt ab Pixel/Noppen nach unten zählen = y-Wert.
  - Position der Figur = (x,y).
- Lehrer spricht Problem an, zu diskutieren: Wo soll zur Positionsbestimmung an der Figur gemessen werden? Links oben, Mitte, rechts unten? Lehrer verweist auf die Origin-Eigenschaft eines Sprites. Prinzipiell ist egal, wo gemessen wird, aber es muss in der Sprite-Eigenschaft festgelegt worden sein!

Erfahrungen aus der Unterrichtspraxis zu Einheit C

(Folgen nach Durchführung)

2.4 Einheit D: Eventhandling, Objekt-Fähigkeiten (Methoden/Actions)

Kerndaten Einheit D

Dauer: 60 Minuten

Inhalte

- D1 Eventhandling, OO: Fähigkeiten/Methoden/Actions | F, UG, EA/PA | 30min.
- D2 Kollisionen | F, UG, EA/PA | 30 min.

Hilfsmittel

- Mehrere Schüler-PCs mit Microsoft Windows, auf denen Game Maker 7 installiert ist
- Lehrer-PC, der an Projektor angeschlossen ist

Lernziele

„In dieser Einheit lernt ihr…

- … Fähigkeiten (Methoden) von Objekten zu programmieren
- … das Verhalten von Objekten zu kontrollieren“

D1 Eventhandling, OO: Fähigkeiten/Methoden/Actions | F, UG, EA/PA | 30min.

- Lehrer bittet Schüler, das Game Design Document zur Hand zu nehmen.
- Schüler werden gebeten, bereits realisierte Elemente des Design Documents zu nennen.
- Schüler-Antworten (Zielsetzung):
  - Objekte,
  - Raum,
  - Stein,
  - Schlüssel,
  - Spieler.
- Lehrer merkt an, dass für den ersten Raum noch keine Monster, Türen oder Diamanten geplant sind, da die Schwierigkeit mit jedem weiteren Raum steigen soll, um das Spiel interessant zu gestalten.
- Schüler werden gefragt, welche Elemente aus dem Game Design Document für den ersten Raum noch fehlen.
- Schüler-Antworten (Zielsetzung):
  - Klänge,
  - Steuerung,
  - Spielfluss.
- Lehrer merkt an, dass zunächst die Steuerung der Spielerfigur programmiert werden soll, und zwar als Reaktion auf den Tastendruck des Benutzers, wie im Game Design Document formuliert.
- Lehrer zeigt am Lehrer-PC, der an Projektor angeschlossen ist, exemplarisch, wie die Bewegung nach rechts durch Events und Actions programmiert werden kann.
  - Objekt „Spieler“ öffnen.
  - Lehrer erklärt, was Events und Actions sind:
    - Das Spiel schickt an alle Objekt-Instanzen Nachrichten, wenn etwas passiert / ein Ereignis (Event) eintritt. Dieses Ereignis kann z.B. ein Tastendruck sein.

Die Nachricht enthält eine Information darüber, was genau das Ereignis war, das eingetreten ist. Die Objekt-Instanzen haben dann die Möglichkeit, auf dieses Ereignis zu reagieren, und zwar mit Actions (Handlungen).

  - `Add Event` > `Keyboard` > `<Right>`.
  - Vorstellung der von Game Maker angebotenen Actions (siehe „Abbildung 5: Vordefinierte Actions“):
    - Thematische Gliederung der Actions in Registerkarten von `Move` bis `Draw`.
    - Einfügen der gewünschten Action durch Drag and Drop in die Liste „Actions“.
  - Action `Move` > `Move` > `Move Fixed` per Drag and Drop ins die Liste “Actions” ziehen.
  - In der Liste “Actions” steht dann “Start moving in a direction”, ein neues Fenster mit Eigenschaften der Action öffnet sich.
  - Im neuen Fenster den Pfeil anklicken, der nach rechts zeigt und als Geschwindigkeit “4” angeben.
  - Fenster schließen.
- Lehrer führt Spiel an Lehrer-PC aus.
- Lehrer drückt die Pfeil rechts-Taste; die Spielerfigur reagiert auf Tasteneingabe, hört aber nicht auf zu laufen und läuft über den Bildschirmrand hinaus.
- Schüler werden gefragt, wie man dieses Problem lösen kann.
- Schüler-Antwort (Zielsetzung):
  - Definition eines neuen Events, der ausgelöst wird, wenn keine Taste gedrückt ist (`<no key>`) und der als Action das Stoppen der Figur hat.
  - Mögliche Hilfestellungen: Blick in Game Design Document, Blick in Liste mögicher Keyboard-Events.
- Lehrer fügt Event und entsprechende Action an Lehrer-PC ein.
  - `Add Event` > `Keyboard` > `<no key>`.
  - Action `Move` > `Move` > `Move Fixed` per Drag and Drop in die Liste “Actions” ziehen.
  - Im neuen Fenster die Mittelstellung anklicken und als Geschwindigkeit “0” angeben.
- Schüler werden gebeten, die weiteren Tasten Links, Oben, Unten analog zu vorgestelltem Beispiel zu programmieren.
- ZWISCHENSTAND, siehe Materialien: `RP3.gmk`.

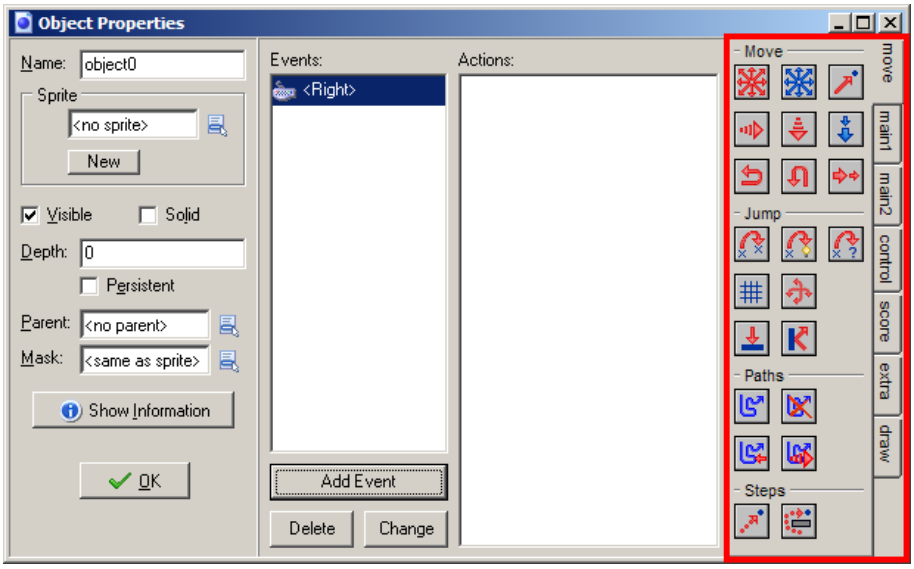


Abbildung 5: Vordefinierte Actions

D2 Kollisionen | F, UG, EA/PA | 30 min.

- Schüler werden gebeten ihr Spiel zu testen.
- Lehrer fragt, ob unerwartetes Verhalten aufgetreten ist.
- Schüler-Antworten (Zielsetzung):
  - Spielerfigur bewegt sich durch/unter Steine/n hindurch.
- Lehrer fragt, welche Action notwendig ist, um Figur vor Stein zu stoppen.
- Schüler-Antworten (Zielsetzung):
  - Gleiche Action wie bei <no key>-Event.
- Lehrer bittet Schüler nachzuschauen, welche Kategorie von Events, die der Event Selector anbietet, wohl am besten geeignet sein könnte um Figur vor einem Stein zu stoppen.
  - Intuitives Ausprobieren: Öffnen des Event Selectors durch `Add Event`.
  - Systematisches Suchen: Öffnen der Online-Hilfe, Suche nach Eintrag „Events“, Erklärungen zu den einzelnen Events in unterer Hälfte des Eintrags.
  - Lehrer gibt Hilfestellung bei Problemen mit den englischen Begriffen.
- Schüler-Antworten (Zielsetzung):
  - „Collision“-Event.
- Schüler werden gebeten, den Event zu programmieren.
  - Event: `Add Event` > `Collision` > `Stein`.
  - Action: Wie bei <no key>-Event.
- ZWISCHENSTAND, siehe Materialien: `RP3b.gmk`.
- Schüler werden gebeten ihr Spiel erneut zu testen.
- Lehrer fragt, ob weiteres unerwartetes Verhalten aufgetreten ist.
- Schüler-Antworten (Zielsetzung):
  - Spielerfigur bewegt sich durch/unter Steine/n hindurch.
- Lehrer fragt, welche Action notwendig ist, um Figur vor Stein zu stoppen.
- Schüler-Antworten (Zielsetzung):
  - Spielerfigur bleibt manchmal an einigen Steinen „hängen“.
  - Grund: Spielerfigur muss in engen Passagen genau im 32x32-Pixel-Raster ausgerichtet sein, um Lücke zwischen zwei Steinen passieren zu können.
- Lehrer merkt an, dass Figur an 32x32-Raster ausgerichtet werden muss.
- Lehrer zeigt Ausrichtung exemplarisch an Lehrer-PC, der an Projektor angeschlossen ist.
  - Bei allen Keyboard-Events neue Action hinzufügen: `Move` > `Jump` > `Align to Grid` > `Snap hor`: *32*, `Snap vert`: *32*.
  - Dabei Geschwindigkeit so einstellen, dass ein ganzzahliges Vielfaches der Geschwindigkeit die Gittergröße 32 ergibt, also, z.B. 4 (8x4=32), 6.4 (5x6.4=32), 8 (4x8=32) oder 16 (2x16=32).
- ZWISCHENSTAND, siehe Materialien: `RP3c.gmk`.

Didaktische Überlegungen zu D2

Die für die weitere Tätigkeit benötigte Information, wie mithilfe von Actions die Spielerfigur bewegt werden kann, könnte von den von Schülern auch eigenständig in der Online-Hilfe nachgeschlagen werden, wie dies für die Informationssuche zu Objekteigenschaften in B2 eingeplant wurde. In diesem Fall wurde hierauf allerdings verzichtet, da in dieser Einheit des Projektes viele Schritte vorzunehmen sind und die Geschwindigkeit des Fortschreitens nicht zu stark gebremst werden soll. Die kognitive Forderung der Schüler in dieser Einheit besteht in der Bildung von Analogien (Events für Kollisionen), Recherchetätigkeit (Informationen zu Events in der Online-Hilfe) und in Transferleistungen (Finden einer alternativen Lösung durch Jumps).

Erfahrungen aus der Unterrichtspraxis zu Einheit D
(Folgen nach Durchführung)

2.5 Einheit E: Kontrollstrukturen, Vererbung, Lebensdauer von Instanzen

Kerndaten Einheit E

Dauer: 90 Minuten

Inhalte

- E1 Kontrollstrukturen: Übergänge zwischen Räumen | UG, EA/PA | 15 min.
- E2 Vererbung: Anlegen einer Oberklasse | UG, EA/PA | 10 min.
- E3 Kontrollstrukturen: Wächterobjekt | UG, EA/PA | 15 min.
- E4 Vererbung: Vererbung von Eigenschaften | UG, EA/PA | 30 min.
- E5 Lebensdauer von Instanzen | UG, EA/PA | 20 min.

Hilfsmittel

- Mehrere Schüler-PCs mit Microsoft Windows, auf denen Game Maker 7 installiert ist
- Lehrer-PC, der an Projektor angeschlossen ist

Lernziele

„In dieser Einheit lernt ihr…

- Vererbungsstrukturen zu nutzen, um gleichartige Objekte einfacher zu programmieren
- abstrakte Kontrollobjekte zu programmieren
- die Lebensdauer von Objektinstanzen zu begrenzen“

E1 Kontrollstrukturen: Übergänge zwischen Räumen | UG, EA/PA | 15 min.

- Lehrer richtet folgende Aufgabe an Schüler: Kollision mit Schlüssel soll nach Abspielen eines Tons zu neuem Raum führen (ggfs. Ableitung aus Game Design Document). Lehrer diskutiert durchzuführende Teilschritte mit Schülern:
  - Ton importieren.
  - Neuen Raum erstellen.
  - Neues Hintergrundbild importieren und in Raum einbinden.
  - Überleitung zu neuem Raum implementieren.
    - Gegebenenfalls Hinweis auf Event „Kollision Spielerfigur mit Schlüssel“ und die entsprechende Action `Main1` > `Go to next room`.
    - Vor der Action `Go to next room` soll noch eine Action `Main1` > `Play sound` > `Tuer zu` eingefügt werden.
    - Damit der Ton vollständig abgespielt wird vor dem Wechsel in den neuen Raum soll noch eine Action `Main2` > `Sleep` eingefügt werden mit der Einstellung 1000 Millisekunden.
    - Gegebenenfalls kann noch ein Hinweis gegeben werden auf mögliche Überblendeffekte zwischen den Räumen, die als Eigenschaft der Action `Go to next room` definiert werden können.

E2 OO: Vererbung: Anlegen einer Oberklasse | UG, EA/PA | 10 min.

- Lehrer richtet folgende Aufgabe an Schüler: Steine in neuem Raum sollen ein anderes Aussehen bekommen als im Steine im ersten Raum, aber genau die gleichen Eigenschaften haben wie die Steine im ersten Raum. Lehrer diskutiert durchzuführende Teilschritte mit Schülern:
  - Lehrer weist darauf hin, dass den Steinen weitere Eigenschaften hinzugefügt werden sollen (z.B. Töne bei Berührung) und dass nicht für jeden Stein das Ereignis neu hinzugefügt werden müssen soll (hoher Zeitaufwand, schlechte Wartbarkeit und Fehleranfälligkeit).
  - Neues Sprite erstellen und Grafik importieren.
  - Neues Objekt erstellen, z.B. `Stein2` mit Eigenschaft `solid`.
  - Alternative Lösung wäre: Objekt ein neues Sprite zuweisen. Dies wird hier aber nicht bevorzugt; Begründung siehe didaktische Überlegungen.
  - Bestehendes Objekt `Stein` umbenennen in `Stein1`.
  - Lehrer weist darauf hin, dass jetzt ein „Elternstein“ (alternativ auch „Mutterstein“ oder „Vaterstein“) kreiert wird, d.h. ein Stein, dessen Eigenschaften alle „Kindersteine“ erben sollen.
  - Neues Objekt `Stein` wird erstellt, kein Sprite wird ihm zugewiesen.
  - Lehrer fragt, warum dem Objekt `Stein` kein Sprite zugewiesen wird.
  - Schüler-Antwort (Zielsetzung):
    - Das Objekt bleibt unsichtbar.
    - Das Objekt wird in keinem Raum platziert.
    - Das Objekt bekommt keine Instanz.
  - Game Maker wird die Eltern-Kind-Beziehung mitgeteilt, indem bei den Objekteigenschaften von `Stein1` und `Stein2` als `Parent` > `Stein` eingetragen wird.

Didaktische Überlegungen zu E2

Zuweisung eines neuen Sprites vs. Erstellung eines neuen Objekts
Alternative Lösung wäre für ein neues Aussehen der Steine im neuen Raum wäre, dem Objekt „Stein“ bei Betreten eines neuen Raums ein neues Sprite zuweisen. Dies ist z.B. mit einem Script in der Game Maker Language möglich und stellt eine ebenso valide Lösung dar. Allerdings ist dieses Vorgehen für das Lernziel „Vererbung“ dieser Einheit weniger geeignet und sollte deswegen als eine mögliche Lösung im Gespräch mit den Schülern anerkannt werden, falls es vorgeschlagen wird. Jedoch sollte mit Verweis auf die Thematik der Vererbung der oben vorgestellten Lösung der Vorrang gegeben werden.

„Elternstein“ und „Kindersteine“
Es wird erwartet, dass die Schüler das Konzept der Vererbung aufgrund der sprachlichen Analogie schnell verstehen werden, zumal sie eine Lösung für eine Problematik darstellt, die sich an diesem Punkt des Unterrichts gerade akut stellt und durch Vererbung gelöst werden kann. Sollten Schüler das Konzept dennoch nicht auf Anhieb durch die sprachliche Analogie verstehen, so ist dies nicht schlimm, da im Folgenden die Vererbung praktisch durchexerziert und somit greifbar wird.

E3 Kontrollstrukturen: Wächterobjekt | UG, EA/PA | 15 min.

- Lehrer erklärt, dass für die folgende Spielentwicklung und für Spieletester wichtig ist, (geheime) Tastenkombinationen zum Springen zwischen Leveln zu haben (Schummeltasten bzw. „Cheats“). In unserem Spiel könnte das z.B. die Taste F12 sein. Die Schüler sollen diese Schummeltasten-Funktion nun implementieren.
    - Event passt aus logischer Sicht nicht gut zu den bestehenden Figuren – es wäre möglich, z.B. einem Stein oder der Spielerfigur einen Event bei Betätigung der Schummeltaste hinzuzufügen.
    - Besser ist hingegen, ein unsichtbares Wächterobjekt zu krieren und einzufügen, das diese und andere allgemeine Aufgaben des Spielverlaufs kontrolliert, die keiner Figur zuzuordnen sind.
    - Aals Name bietet sich z.B. an es tatsächlich *Waechter* zu nennen, Eigenschaft `visible` deaktiviert, kein Sprite.

E4 OO: Vererbung: Vererbung von Eigenschaften | UG, EA/PA | 30 min.

- Lehrer richtet folgende Aufgabe an Schüler: Ein neuer Raum soll erstellt werden, in dem sich Diamanten befinden, Spielerfigur und Schlüssel sollen platziert werden (15 min.).
    - Lehrer weist darauf hin, dass Spielerfigur durch die Wände im zweiten Raum gehen wird.
    - Lehrer bittet Schüler, dafür eine Lösung zu finden.
        - Ursache des Problems ist, dass Spielerfigur nur auf Kollision mit Stein1 reagiert.
        - Lösung 1 ist, dass bei Spielerfigur der Kollisions-Event auf Kollision mit `Stein` geändert wird.
        - Lösung 2 ist, dass bei Spieler der Kollisions-Event entfernt wird und bei `Stein` ein Kollisions-Event mit dem Spieler eingerichtet wird, der Spielerfigur stoppt.
- Lehrer richtet an Schüler die Aufgabe, dass bei Berührung eines beliebigen Typ von Steins ein Klang ertönen soll.
    - Lösung: Kollisions-Event mit Spieler in Oberklasse `Stein`, Action `Main1` > `Play Sound`, zusätzlich danach weitere Action `Main2` > `Sleep` mit Einstellung 150 Millisekunden (um Ton nicht zu oft erklingen zu lassen).
- Lehrer fragt Schüler, warum Ton bei allen Steinen erklingt.
- Schüler-Antworten (Zielsetzung):
    - Eigenschaft wird an Kind-Objekte vererbt.

E5 Lebensdauer von Instanzen | UG, EA/PA | 20 min.

- Lehrer weist Schüler auf die Actions Main1 > Objects hin, mit denen z.B. eine Instanz entfernt werden kann (Destroy Instance).
- Lehrer weist Schüler auf die Action `Create Instance` hin. Schüler werden gefragt, was mit dieser Action und der Destroy-Action wohl prinzipiell alles im Spielverlauf noch machbar / möglich wäre.
- Schüler-Antwort (Zielsetzung):

    - Ein Objekt kann neue Instanzen erstellen, z.B. das Wächter-Objekt kann neue Steine erstellen.
    - Ein Objekt kann Instanzen löschen, z.B. kann die Spielerfigur bei Berührung ein anderes Objekt entfernen (Diamanten!).
- Schüler werden gebeten, der Spielerfigur einen neuen Event zu programmieren, der bei Berührung die Diamanten löscht und einen Ton erklingen lässt.
    - Lösung 1: `Collision`-Event bei Spielerfigur mit Diamant, Action `Destroy Instance`, Einstellung `Other` (da nicht die Spielerfigur, sondern der Diamant gelöscht werden soll).
    - Lösung 2: `Collision`-Event bei Diamant mit Spielerfigur, Action `Destroy Instance`, Einstellung `Self`.
- Die Lösung mit dem verschwindenden Diamanten soll auf die Schlüssel übertragen werden; diese sollen ebenfalls bei Berührung verschwinden, bevor zum nächsten Raum übergeleitet wird.

Erfahrungen aus der Unterrichtspraxis zu Einheit E

(Folgen nach Durchführung)

2.5 Einheit F: Variablen, bedingte Anweisung und Verzweigung

Kerndaten Einheit F

Dauer: 90 Minuten

Inhalte

- F1 Variablen | UG, EA/PA | 20 min.
- F2 Animierte Grafiken | UG, EA/PA | 20 min.
- F3 Kontrollstrukturen: Bedingte Anweisung und Verzweigung | UG, EA/PA | 40 min.
- F4 Vorbereitung eigener Erweiterungen | UG, EA/PA | 10 min.

Hilfsmittel

- Mehrere Schüler-PCs mit Microsoft Windows, auf denen Game Maker 7 installiert ist
- Lehrer-PC, der an Projektor angeschlossen ist

Lernziele

„In dieser Einheit lernt ihr…

- Variablen zu benutzen um Daten zu speichern
- bestimmte Anweisungen nur unter bestimmten Bedingungen auszuführen (bedingte Anweisung und Verzweigung)“

F1 Variablen | UG, EA/PA | 20 min.

- Schüler werden gebeten, innerhalb von 10 Minuten herauszufinden, wie ein Spielstand mit Game Maker verwaltet werden kann, aber die Verwaltung noch nicht zu programmieren.
  - Als Impuls wird ggfs. der englische Begriff „Score“ genannt, falls die englische Hilfedatei verwendet wird.
- Schüler-Verhalten (Zielsetzung):
  - Suche in der Online-Hilfe nach *Score*.
  - Auffinden zweier Artikel zum Thema.
  - Auswahl des passenden Artikels (`Score actions`) .
- Lehrer stellt sicher, dass die Funktion der Option `Relative` verstanden wurde, nämlich, dass zum aktuellen Spielstand die angegebene Anzahl Punkte addiert wird – ansonsten würde der Spielstand nämlich auf die angegebene Anzahl Punkte zurückgesetzt.
- Schüler werden gebeten, den Spielstand so zu implementieren, dass er zu Beginn des Spiels auf 0 steht und mit jedem gesammelten Diamanten um 10 Punkte steigt (ggfs. Ableitung aus Game Design Document).
  - Spielstand wird in Fenstertitel angezeigt.

F2 Animierte Grafiken | UG, EA/PA | 20 min.

- Lehrer weist darauf hin, dass die Spielerfigur grafisch interessanter gemacht werden kann.
- Schüler werden gebeten, das Sprite der Spielerfigur neu zu laden, und zwar die Datei `Spieler animiert - [tut].gif`. Lehrer weist darauf hin, dass es sich hierbei um ein animiertes GIF handelt, also eine Grafikdatei, die aus 4 Einzelbildern besteht, die zusammen

eine Animation ergeben könnten (sich drehende Spielerfigur). Es sollen aber jeweils nur die Einzelbilder verwendet werden, je nach Richtung, in der die Spielerfigur gerade läuft. Man könnte dazu auch 4 Einzelsprites importieren, was aber unübersichtlich ist.

- Schüler werden gebeten, sich die vier Einzelbilder der Figur anzuschauen mit den verschiedenen Ausrichtungen der Figur nach Laufrichtungen in `Sprite properties` > Links mittig `Show` > Pfeil-Buttons und dabei die Nummern der Unterbilder („subimages“) zu beachten.
- Lehrer macht Schüler aufmerksam auf die Action `Main1` > `Change Sprite` und deren Eigenschaft `Speed`, die bei Setzen von 0 das Anhalten einer Animation erlaubt.
- Schüler werden gebeten, die Bewegung der Figur so umzuprogrammieren, dass die Figur immer in Richtung ihres Laufens blickt.
    - Bei jedem der bisher definierten Key-Events muss der Figur ein neues, passendes Unterbild zugewiesen werden.
    - Beim `<no key>`-Event sollte das Sprite optimalerweise auf das Unterbild 0 zurückgesetzt werden.
    - Es muss ein neuer `Create`-Event erstellt werden, der die Animation stoppt, sonst dreht sich die Spielerfigur zu Beginn jedes Raumes.

F3 Kontrollstrukturen: Bedingte Anweisung und Verzweigung | UG, EA/PA | 40 min.

- Schüler werden auf das Problem aufmerksam gemacht, das nach Beeindigung des letzten Raums eine Fehlermeldung kommt.
- Lehrer erklärt, dass Räume in Game Maker in der Reihenfolge ihrer Auflistung in der Resourcenliste abgearbeitet werden und fragt, was Ursache für Fehlermeldung ist.
- Schüler-Antworten (Zielsetzung):
    - Es ist kein weiterer Raum vorhanden. (Warum ist das ein Problem für den Computer?)
    - Wir haben keine spezielle Anweisung festgelegt, wie nach dem letzten Raum zu verfahren ist.
    - Der Computer befolgt die Standard-Anweisung und versucht in den nächsten Raum zu springen, aber es existiert keiner.
- Lehrer fragt, welche umgangssprachliche Formulierung man wählen könnte, um dem Computer mitzuteilen was er bei jedem Sprung von Raum zu Raum machen soll. Antwort wird an Tafel geschrieben.

- Schüler-Antwort (Zielsetzung):

(1) Wenn ein weiterer Raum vorhanden ist, springe zu nächstem Raum.
Sonst: Beende Spiel. = Wenn kein weiterer Raum vorhanden ist, beende spiel.

```
Wenn (weiterer Raum vorhanden?)
        Springe zu nächstem Raum
Sonst
        Beende Spiel
```

```
Wenn (weiterer Raum vorhanden?)
        Springe zu nächstem Raum
Wenn NICHT (weiterer Raum vorhanden?)
        Beende Spiel
```

(2) Wenn der aktuelle Raum nicht der letzte ist, springe zu nächstem Raum.
Wenn der aktuelle Raum der letzte ist, beende Spiel.

```
Wenn NICHT (aktueller Raum der letzte?)
        Springe zu nächstem Raum
Sonst
        Beende Spiel
```

```
Wenn NICHT (aktueller Raum der letzte?)
        Springe zu nächstem Raum
Wenn (aktueller Raum der letzte?)
        Beende Spiel
```

- Lehrer merkt an, dass es sich hierbei um bedingte Anweisungen und Verzweigung handelt, und dass das feste und sehr wichtige Bestandteile jeder Programmiersprache sind.
    - Bedingte Anweisungen bestehen aus einer Frage, die mit Ja oder Nein beantwortet werden kann. Man könnte diese Frage auch einen Test nennen, der richtig/wahr oder falsch ergeben kann.
    - Bedingte Anweisungen bestehen außerdem aus Reaktionsmöglichkeiten, entweder einer oder zwei. Das ist logisch, weil der Test ja auch zwei Ergebnisse ergeben kann – richtig/wahr oder falsch. Oft will man nur eine Anweisung geben, wenn ein Test richtg/wahr ist. Manchmal möchte man auch noch eine alternative Anweisung geben, falls der Test falsch ist. Für jede der möglichen Antwort muss es eine unterschiediche Reaktionsmöglichkeit geben.
- Schüler werden gebeten, diejenigen Stellen (Objekte > Events) zu suchen, an denen zum nächsten Raum gesprungen wird
    - Zum Beispiel: `Schluessel` > Kollision mit `Spieler`, oder eventuell auch: `Spieler` > Kollision mit `Schlüssel`, je nach Implementierung.
    - `Waechter` > Tastendruck `F12`.
- Lehrer macht Schüler aufmerksam auf die folgenden Actions:
    - `Main1` > `Check Next` > `If next room exists`
        - Das ist der Test „(weiterer Raum vorhanden?)“ aus dem Beispiel an der Tafel.
    - `Control` > `Other` > `Start Block` / `End Block`
        - Das sind Anfang und Ende der Einrückungen aus dem Beispiel an der Tafel.
    - `Control` > `Other` > `Else`
        - Das ist der Hinweis „sonst” aus dem Beispiel an der Tafel.
- Schüler werden gebeten, um die oben gefundenen Stellen herum einen Schutzmechanismus in Form einer bedingten Anweisung zu bauen. Diese soll kontrollieren, ob ein weiterer Raum existiert und nur dann zu diesem Raum springen.
- Schüler-Lösung (Zielsetzung):
    - If next room exists
      Start of a block
      Go to next room
      End of a block

- Schüler werden gebeten, die Aufgabe „Bedingte Anweisungen und Verzweigungen“ in der Projektmappe zu bearbeiten (15 min.).
- Lehrer bespricht Ergebnisse der Aufgabe mit den Schülern.
- Schüler werden gebeten, eine alternative Anweisung bei der bedingten Anweisung hinzuzufügen, die nach Beendigung des letzten Raums eine Highscore-Tabelle anzeigt und danach das Spiel neu startet oder beendet.
- Schüler-Lösung (Zielsetzung):
  - `Control` > `Other` > `Else`
  - `Score` > `Score` > `Show Highscore`
  - `Main2` > `Game` > `Restart Game` oder `Main2` > `Game` > `End Game`
- ZWISCHENSTAND, siehe Materialien: `RP4.gmk`

F4 Vorbereitung eigener Erweiterungen | UG, EA/PA | 10 min.

- Zur Vorbereitung eigener Erweiterungen werden die Schüler gebeten, einen neuen Raum zu erstellen und darin neue Objekte vom Typ Monster zu platzieren, die vorher erstellt wurden und die diese Eigenschaften haben:
  - Nach Erstellung eigenständige Bewegung nach links/rechts bis Kollision mit Stein, dann Umkehrung der Laufrichtung,
  - Anpassung des Monster-Sprites an Laufrichtung,
  - Bei Kollision mit Spieler Punkteabzug und Neustart des Raumes.
- Die Erstellung der Monster-Objekte wird in der folgenden Sitzung weitergeführt. Danach besteht Raum für eigene kreative Erweiterungen.
- ENDSTAND: `RP5.gmk`.

Erfahrungen aus der Unterrichtspraxis zu Einheit F

(Folgen nach Durchführung)

2.6 Einheit G: Eigene Erweiterungen und Abschluss

Die letzte Sitzung soll genutzt werden, um den Schülern Raum einzuräumen für eigene Erweiterungen des Spiels, wobei die Akzente völlig frei und offen gesetzt werden dürfen. Sollten sich wider Erwarten bei einigen Schülern keine einigen Ideen zur Erweiterung finden, finden sich im Folgenden einige Anregungen, wofür die Stunde beispielsweise genutzt werden könnte:

- Entwurf weiterer Räume mit steigenden Schwierigkeitsgraden.
- Einbau versteckter Geheimtüren in weitere Räume (Bonuspunkte!).
- Einführung von Objekten, die verschoben werden können, z.B. Spezialsteine, die wie normale Steine aussehen, aber verschoben werden können um Wege abzukürzen.
- Einführung weiterer Typen von Monstern.
- Einführung eines Modus in dem Monster außer Gefecht gesetzt werden.
- Programmierung einer Intro-/Outro-Sequenz, in der Figuren nicht vom Spieler gesteuert werden, sondern einer vom Programmierer definierten Choreografie folgen.

Die Schüler sollten soweit möglich angehalten werden, ihre eigenen Erweiterungen im Game Design zu dokumentieren.

Am Ende des Projekts bietet es sich an, alle Schüler ihre Erweiterungen kurz vorstellen zu lassen. Die Lerneinheiten sollten anschließend mit einem kurzen Fazit-Gespräch beschlossen werden, in dem die Schüler die Inhalte und Fähigkeiten, die sie sich im Projekt angeeignet haben, bündig zusammenfassen. Ebenso sollte zur Qualitätssicherung eine Evalution des Projektes von Schülerseite erfolgen.

Literaturverzeichnis

Boettcher, Daniel, et al. "Ada – dieser Zug hat Verspätung." Schubert, Sigrid. Didaktik der Informatik in Theorie und Praxis. Bonn: Gesellschaft für Informatik, 2007. 217-228.

Börstler, Jürgen. "Objektorientiertes Programmieren – Machen wir irgendwas falsch?" Schubert, Sigrid. Didaktik der Informatik in Theorie und Praxis. Bonn: Gesellschaft für Informatik, 2007. 9-20.

Brändle, Markus, et al. "Kara: Ein theoriebasierter Ansatz für Lernumgebungen zu fundamentalen Konzepten der Informatik." Hubwieser, Peter. Informatische Fachkonzepte im Unterricht. Bonn: Gesellschaft für Informatik, 2003. 201-210.

Brichzin, Peter, et al. Ikarus. Natur und Technik. Schwerpunkt: Informatik 6/7. München: Oldenbourg, 2004.

Diethelm, Ira. "Stricly models and objects first – Unterrichtskonzept für die objektorientierte Modellierung." Schubert, Sigrid. Didaktik der Informatik in Theorie und Praxis. Bonn: Gesellschaft für Informatik, 2007. 45-56.

Diethelm, Ira, Leif Geiger and Albert Zündorf. "Rettet Prinzessin Ada: Am leichtesten objektorientiert." Friedrich, Steffen. Unterrichtskonzepte für informatische Bildung. Bonn: Gesellschaft für Informatik, 2005. 161-172.

Frey, Elke. "Informatik in der Jahrgangsstufe 6 – ein Bericht aus der Praxis." Hubwieser, Peter. Informatische Fachkonzepte im Unterricht. Bonn: Gesellschaft für Informatik, 2003. 33.

Hubwieser, Peter. "Von der Funktion zum Objekt – Informatik für die Sekundarstufe 1." Friedrich, Steffen. Unterrichtskonzepte für informatische Bildung. Bonn: Gesellschaft für Informatik, 2005. 27-41.

Kohl, Lutz. "Puck – eine visuelle Programmiersprache für die Schule." Friedrich, Steffen. Unterrichtskonzepte für informatische Bildung. Bonn: Gesellschaft für Informatik, 2005. 309-318.

Ministerium für Kultus, Jugend und Sport des Landes Baden-Württemberg. "Bildungsplan 2004. Allgemein bildendes Gymnasium." 2004. Bildung stärkt Menschen — Bildungsplan 2004. 31 5 2009 <http://www.bildung-staerkt-menschen.de/service/downloads/Bildungsplaene/Gymnasium/Gymnasium_Bildungsplan_Gesamt.pdf>.

YoYo Games. YoYo Games Game Maker. 2008. 5 7 2009 <http://www.yoyogames.com/gamemaker/>.

Anhang 1: Mögliche Stolperfallen und ihre Umgehung

Unerwartete Freezes
Im Rahmen der Vorbereitung dieses Projekts habe ich auf mehreren Systemen, auf denen ich mit Game Maker 7 gearbeitet habe, bemerkt, dass der Game Maker-Prozess insbesondere beim Verlassen des Sprite Editors in einigen Fällen nicht mehr reagierte. Der Fehler trat allerdings nur sporadisch auf und war nicht reproduzierbar. Die Fehlerquelle war somit nicht aufzufinden. Das Programm lässt sich in solchen Fällen problemlos mithilfe des Betriebssystems beenden und kann danach neu gestartet werden. Allerdings kann dann nur der zuletzt gesicherte Datenstand aufgerufen werden. Ich möchte daher mit großem Nachdruck darauf aufmerksam machen, die Schüler im Rahmen des Projekts zu regelmäßigem Speichern anzuregen (es empfehlen sich Zeitabstände von wenigen Minuten). Dadurch dürften sich Datenverluste durch Programmfehler in vertretbaren Grenzen halten lassen.

Problematische Sprites
Im Rahmen der Vorbereitung dieses Projekts habe ich zudem auf mehreren System bei Game Maker 7 die Tendenz festgestellt, dass einige Grafik-Dateiformate Probleme beim Start des Spiels verursacht haben. In den von mir beobachteten Fällen handelte es sich um GIF-Dateien für Sprites, die diese Probleme ausgelöst hatten. Problematisch war, dass das Spiel nicht mehr gestartet werden konnte, sobald die auslösende Grafik für das Sprite geladen worden war. Dies äußerte sich darin, dass der erste Raum nicht geladen wurde. Stattdessen blieb der Bildschirm leer und nach einem kurzen Moment Pause erschien die Game Maker-Oberfläche wieder. Dieser Fehler trat wohlgemerkt auf, sobald die Grafik geladen war. Das Sprite musste weder mit einem Objekt verknüpft, noch in Verknüpfung mit einem Objekt in einem Raum platziert sein. Als Workaround empfehle ich zwei Methoden. Zum einen kann in einem solchen Fall das Sprite gelöscht oder die Grafik des Sprites gegen eine andere Grafik eingetauscht werden. Dadurch ließ sich bei mir in allen Fällen das Spiel wieder starten. Sollte die problematische Grafik hingegen beibehalten werden müssen, empfehle ich folgenden Versuch als Workaround: Sprite Editor für die problematische Grafik laden (Sprite Properties > Edit Sprite) und versuchen, die Leinwandgröße des Bildes (Canvas Size) zu verändern. Ich habe gute Erfahrungen damit gemacht, die Leinwandgröße auf ein symmetrisches Maß zu setzen, das der der anderen Sprites entspricht.

Anhang 2: Materialien zur Erweiterung der Projektinhalte

Ein integraler Bestandteil des hier vorgestellten Unterrichtskonzepts ist eine Zusammenstellung von Materialien, mit denen Schüler ihre Projekte selbständig erweitern können. Es handelt sich hierbei um Hintergrundbilder, Sprites, Töne, Hintergrundmusik, Dateisymbole, Infotafeln und Game IDs. Zwar existiert eine Vielzahl solcher Materialien im Web, die die Schüler eigenständig herunterladen und benutzen könnten, allerdings ist der urheberrechtliche Status dieser Materialien in der Mehrzahl der Fälle schlecht dokumentiert und/oder zweifelhaft. Die für dieses Unterrichtskonzept zusammengestellten Materialien sind hingegen alle urheberrechtlich dokumentiert und für den Gebrauch in nicht-kommerziellen Projekten freigegeben. Die Quellen dieser Materialien, sind dieser Arbeit beigefügt; die Quellenangaben finden sich in einer Datei, die den Materialien beiliegt. Eine Übersicht über die Materialien findet sich untenstehend.

Musik

16 Musikstücke:

[bjincs002] - A Strange Night.mp3
[bjincs002] - Depression.mp3
[bjincs002] - Sidewalk.mp3
[flopdog99] - Song of the Tiger.mp3
[flopdog99] - Up da beat.mp3
[lateksi2] - music 00.mid
[lateksi2] - music 01.mid
[lateksi2] - music 02.mid
[lateksi2] - music 03.mid
[lateksi2] - music 04.mid
[lateksi2] - music 05.mid
[lateksi2] - music 06.mid
[lateksi2] - music 07.mid
[lateksi2] - music 11.mid
[sphere_software2] - ambienze.mp3
[sphere_software] - electro_jazz.mp3

Dateisymbole

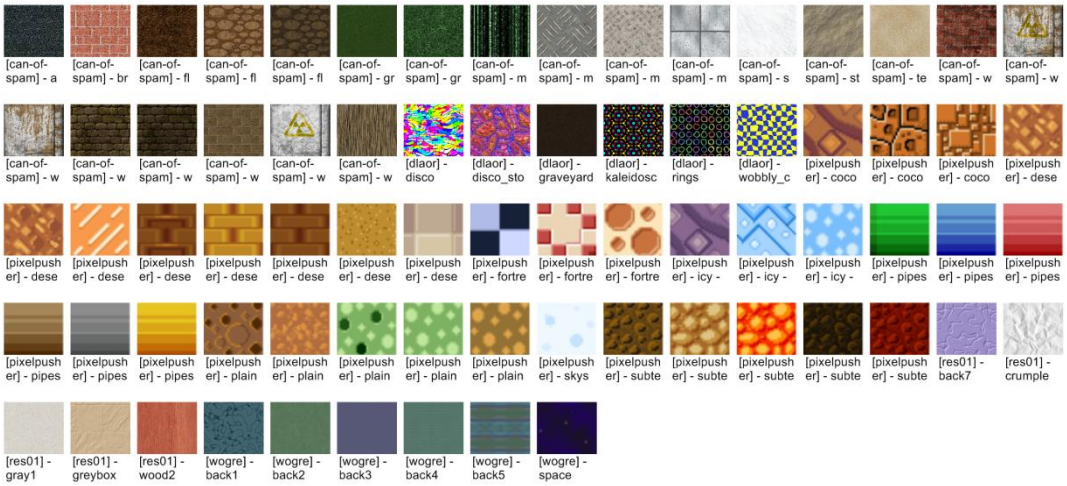
55 Objekte:

Teile und Sprites

280 Objekte:

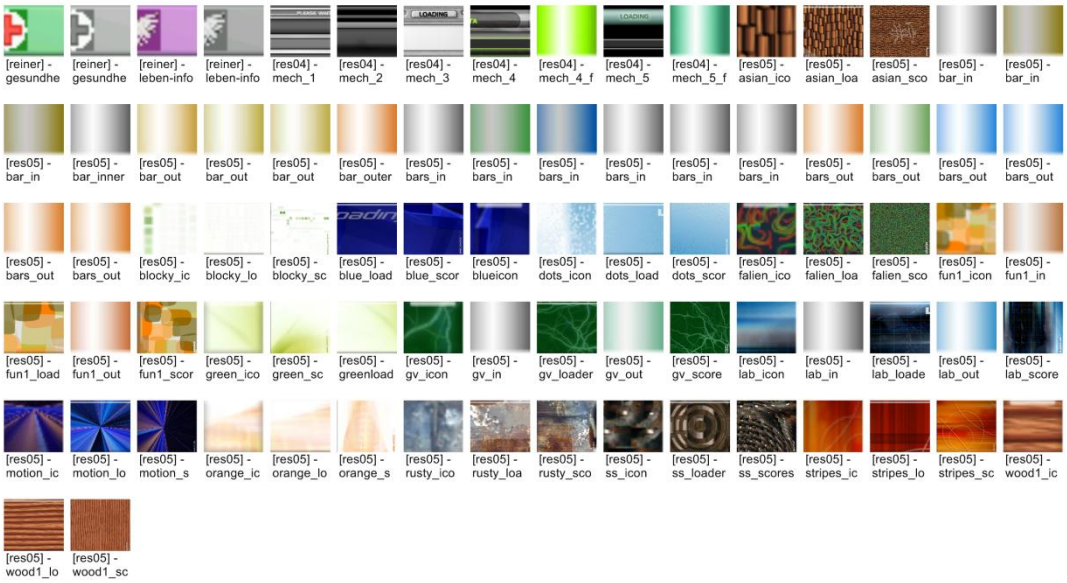
Hintergründe

73 Objekte:



Infotafeln und Game IDs

98 Objekte, z.T. in Unterordnern:



Töne

222 Töne:

[gotxbrains2] - Balloon Pop.wav
[gotxbrains2] - Bomb Fall 01.wav
[gotxbrains2] - Bomb Fall 02.wav
[gotxbrains2] - Comical Descent.wav
[gotxbrains2] - Comical Metal Gong.wav
[gotxbrains2] - Comical Pop and Swirl.wav
[gotxbrains2] - Swirl 01.wav
[gotxbrains2] - Swirl 02.wav
[gotxbrains2] - Swirl 03.wav
[gotxbrains2] - Swirl 04.wav
[gotxbrains3] - Hand Scanner.wav
[gotxbrains3] - Laser Gun.wav
[gotxbrains3] - Pnuematic Flange.wav
[gotxbrains3] - Rocket Launcher Sci Fi 01.wav
[gotxbrains3] - Rocket Launcher Sci Fi 02.wav
[gotxbrains3] - Rocket Launcher Sci Fi 03.wav
[gotxbrains3] - Sci Fi Beep 01.wav
[gotxbrains3] - Sci Fi Beep 02.wav
[gotxbrains3] - Sci Fi Beep 03.wav
[gotxbrains3] - Sci Fi Beep 04.wav
[gotxbrains3] - Sci Fi Beep 05.wav
[gotxbrains3] - Sci Fi Beep 06.wav
[gotxbrains3] - Sci Fi Beep 07.wav
[gotxbrains3] - Sci Fi Beep 08.wav
[gotxbrains3] - Sci Fi Beep 09.wav
[gotxbrains3] - Sci Fi Beep 10.wav
[gotxbrains3] - Sci Fi Beep 11.wav
[gotxbrains3] - Sci Fi Beep 12.wav
[gotxbrains3] - Sci Fi Beep 13.wav
[gotxbrains3] - Sci Fi Beep 14.wav
[gotxbrains3] - Sci Fi Beep 15.wav
[gotxbrains3] - Space Gun 01.wav
[gotxbrains3] - Space Gun 02.wav
[gotxbrains3] - Space Gun 03.wav
[gotxbrains3] - Space Gun 04.wav
[gotxbrains3] - Space Gun 05.wav
[gotxbrains3] - Space Gun 06.wav
[gotxbrains3] - Space Gun 07.wav
[gotxbrains3] - Space Gun 08.wav
[gotxbrains3] - Space Gun 09.wav
[gotxbrains3] - Space Gun 10.wav
[gotxbrains3] - Space Gun 11.wav
[gotxbrains3] - Space Gun 12.wav
[gotxbrains3] - Space Gun 13.wav
[gotxbrains3] - Space Gun 14.wav
[gotxbrains3] - Space Gun 15.wav
[gotxbrains3] - Space Gun 16.wav
[gotxbrains] - Arcade Action 01.wav
[gotxbrains] - Arcade Action 02.wav
[gotxbrains] - Arcade Action 03.wav
[gotxbrains] - Arcade Action 04.wav
[gotxbrains] - Arcade Action 05.wav
[gotxbrains] - Arcade Action 06.wav
[gotxbrains] - Arcade Alarm 01.wav
[gotxbrains] - Arcade Alarm 02.wav
[gotxbrains] - Arcade Alarm Loop 01.wav
[gotxbrains] - Arcade Alarm Loop 02.wav
[gotxbrains] - Arcade Beep 01.wav
[gotxbrains] - Arcade Beep 02.wav
[gotxbrains] - Arcade Beep 03.wav
[gotxbrains] - Arcade Beep 04.wav
[gotxbrains] - Arcade Beep 05.wav
[gotxbrains] - Arcade Chirp 01.wav
[gotxbrains] - Arcade Chirp 02.wav
[gotxbrains] - Arcade Chirp 03.wav
[gotxbrains] - Arcade Chirp 04.wav
[gotxbrains] - Arcade Chirp 05.wav
[gotxbrains] - Arcade Chirp 06.wav
[gotxbrains] - Arcade Chirp 07.wav
[gotxbrains] - Arcade Chirp 08.wav
[gotxbrains] - Arcade Chirp Descend 01.wav
[gotxbrains] - Arcade Chirp Descend 02.wav
[gotxbrains] - Arcade Power Up 01.wav
[gotxbrains] - Arcade Power Up 02.wav
[gotxbrains] - Arcade Power Up 03.wav
[gotxbrains] - Game Over 01.wav
[gotxbrains] - Ping Pong Ball 01.wav
[gotxbrains] - Ping Pong Ball on Table 01.wav
[gotxbrains] - Ping Pong Ball on Table 02.wav
[gotxbrains] - Ping Pong Ball on Table 03.wav
[gotxbrains] - Ping Pong Ball on Table 04.wav
[gotxbrains] - Ping Pong Paddle 01.wav
[gotxbrains] - Ping Pong Paddle 02.wav
[gotxbrains] - Ping Pong Paddle 03.wav
[gotxbrains] - Ping Pong Paddle 04.wav
[gotxbrains] - Ping Pong Paddle Placed on Table 01.wav
[vorpal8b] - vpl_tropic_goodjob01.wav
[vorpal8c] - vpl.0.22500.ST.wav
[vorpal8c] - vpl.1.22500.ST.wav
[vorpal8c] - vpl.10.22500.ST.wav
[vorpal8c] - vpl.2.22500.ST.wav
[vorpal8c] - vpl.3.22500.ST.wav
[vorpal8c] - vpl.4.22500.ST.wav
[vorpal8c] - vpl.5.22500.ST.wav
[vorpal8c] - vpl.6.22500.ST.wav
[vorpal8c] - vpl.7.22500.ST.wav
[vorpal8c] - vpl.8.22500.ST.wav
[vorpal8c] - vpl.9.22500.ST.wav
[vorpal8c] - vpl.Animal_Cow.wav
[vorpal8c] - vpl.Animal_House.wav
[vorpal8c] - vpl.Animal_Lion.wav
[vorpal8c] - vpl.Animal_Rooster.wav
[vorpal8c] - vpl.Animal_Wolf.wav
[vorpal8c] - vpl.Arrow.22500.wav
[vorpal8c] - vpl.Arrow.Close.22500.ST.wav
[vorpal8c] - vpl.Arrow.Far.22500.ST.wav
[vorpal8c] - vpl.Battle_Decrease.Mono01.wav
[vorpal8c] - vpl.Battle_Decrease.Strereo01.wav
[vorpal8c] - vpl.Battle_Increase.Mono01.wav
[vorpal8c] - vpl.Battle_Increase.Stereo01.wav
[vorpal8c] - vpl.Bell.Mono01.wav
[vorpal8c] - vpl.BirdCall.StereoPan01.wav
[vorpal8c] - vpl.BirdCall.StereoPan02.wav
[vorpal8c] - vpl.Boomarang.StereoPan01.wav
[vorpal8c] - vpl.Bounce.Fade.01.ST.wav
[vorpal8c] - vpl.Bounce.Fade.01.wav
[vorpal8c] - vpl.Bounce.Fade.02.wav
[vorpal8c] - vpl.Bounce.REV.wav
[vorpal8c] - vpl.Bounce.wav
[vorpal8c] - vpl.Cancel.Mono01.wav
[vorpal8c] - vpl.Cancel.Mono02.wav
[vorpal8c] - vpl.Car.Crank.32khz.ST.wav
[vorpal8c] - vpl.Car.Uncrank.32khz.ST.wav
[vorpal8c] - vpl.ComeBack.StereoPan01.wav
[vorpal8c] - vpl.Confused.Mono01.wav
[vorpal8c] - vpl.Confused.Mono02.wav
[vorpal8c] - vpl.Cowagunga.wav
[vorpal8c] - vpl.Cursor.Mono01.wav
[vorpal8c] - vpl.Cursor.Mono02.wav
[vorpal8c] - vpl.Dice.x1.32khz.wav
[vorpal8c] - vpl.Door.Creek.R.32khz.wav
[vorpal8c] - vpl.Door.Open.22500.wav
[vorpal8c] - vpl.Door.Shut.22500.wav
[vorpal8c] - vpl.EnemyAttacks.Mono01.wav
[vorpal8c] - vpl.EnemyDamage.Mono01.wav
[vorpal8c] - vpl.EnemyDie.StereoPan01.wav

[vorpal8c] - vpl.Explosion.Stereo01.wav
[vorpal8c] - vpl.Explosion.Stereo02.wav
[vorpal8c] - vpl.FallingDown.Mono01.wav
[vorpal8c] - vpl.FallingDown.Short.Mono01.wav
[vorpal8c] - vpl.FallingUP.Mono01.wav
[vorpal8c] - vpl.FallingUP.Short.Mono01.wav
[vorpal8c] - vpl.Fall_01.wav
[vorpal8c] - vpl.Fall_02.wav
[vorpal8c] - vpl.Five.Hundred.wav
[vorpal8c] - vpl.FlyAway01.StereoPan.wav
[vorpal8c] - vpl.FlyBack01.StereoPan.wav
[vorpal8c] - vpl.Four.Hundred.wav
[vorpal8c] - vpl.Gallop.22500.ST.wav
[vorpal8c] - vpl.Game.Over.Man.Smurf.wav
[vorpal8c] - vpl.Game.Over.Man.wav
[vorpal8c] - vpl.Game.Over.My.Friend.Chipmink.wav
[vorpal8c] - vpl.Grass.Mono01.wav
[vorpal8c] - vpl.Hoooh.01.22500.ST.wav
[vorpal8c] - vpl.Hoooh.02.22500.ST.wav
[vorpal8c] - vpl.Hoooh.03.22500.ST.wav
[vorpal8c] - vpl.Hoooh.04.22500.ST.wav
[vorpal8c] - vpl.Incoming.StereoPan01.wav
[vorpal8c] - vpl.Item.StereoPan01.wav
[vorpal8c] - vpl.Item.StereoPan02.wav
[vorpal8c] - vpl.Knock.01.wav
[vorpal8c] - vpl.Knock.02.wav
[vorpal8c] - vpl.Knock.Mono01.wav
[vorpal8c] - vpl.Shot.LASER.32.khz.ST.wav
[vorpal8c] - vpl.Shot.MGun1.32.khz.ST.wav
[vorpal8c] - vpl.Shot.Single.01.22500.ST.wav
[vorpal8c] - vpl.Shot.Single.02.22500.ST.wav
[vorpal8c] - vpl.Shot.Single.03.22500.ST.wav
[vorpal8c] - vpl.Shotgun.DBarrel.One.wav
[vorpal8c] - vpl.Shotgun.DBarrel.wav
[vorpal8c] - vpl.Siren.Mono01.wav
[vorpal8c] - vpl.Siren.StereoPan01.wav
[vorpal8c] - vpl.Siren.StereoPan02.wav
[vorpal8c] - vpl.Space.Ship.wav
[vorpal8c] - vpl.SpookyRiff.StereoPan01.wav
[vorpal8c] - vpl.Spring_01.wav
[vorpal8c] - vpl.Take.It.wav
[vorpal8c] - vpl.Three.Hundred.wav
[vorpal8c] - vpl.Tink.1.wav
[vorpal8c] - vpl.Trip.Mono01.wav
[vorpal8c] - vpl.Trot.32khz.ST.wav
[vorpal8c] - vpl.Two.Hundred.wav
[vorpal8c] - vpl.UhhSpook.01.22500.wav
[vorpal8c] - vpl.UhhSpook.02.22500.wav
[vorpal8c] - vpl.Unsheath.22500.wav
[vorpal8c] - vpl.Uuuunhh.01.22500.wav
[vorpal8c] - vpl.Uuuunhh.02.22500.wav
[vorpal8c] - vpl.Warp.01.22500.wav
[vorpal8c] - vpl.Warp.02.22500.wav
[vorpal8c] - vpl.Warp.03.22500.wav
[vorpal8c] - vpl.Warp.04.22500.wav
[vorpal8c] - vpl.Warp.05.22500.wav
[vorpal8c] - vpl.Warp.Mono01.wav
[vorpal8c] - vpl.Warp.Stereo01.wav
[vorpal8c] - vpl.YOU.wav
[vorpal8c] - vpl.Knock.Mono02.wav
[vorpal8c] - vpl.Later.StereoPan01.wav
[vorpal8c] - vpl.Monster.Die.01.wav
[vorpal8c] - vpl.Monster.Die.02.wav
[vorpal8c] - vpl.Monster.Die.03.wav
[vorpal8c] - vpl.Monster.Die.04.wav
[vorpal8c] - vpl.Motor.Cycle.22500.ST.wav
[vorpal8c] - vpl.Motor.Cycle.Idle.22500.ST.wav
[vorpal8c] - vpl.Motor.Cycle.Pan.22500.ST.wav
[vorpal8c] - vpl.One.Hundred.wav
[vorpal8c] - vpl.Outgoing.StereoPan01.wav
[vorpal8c] - vpl.Phaser.Mono01.wav
[vorpal8c] - vpl.Phaser.Mono02.wav
[vorpal8c] - vpl.Pouring.Water.wav
[vorpal8c] - vpl.Punch.01.22500.wav
[vorpal8c] - vpl.Punch.02.22500.wav
[vorpal8c] - vpl.Punch.03.22500.wav
[vorpal8c] - vpl.Punch.04.Defeated.ST.22500.wav
[vorpal8c] - vpl.Punch.04.Final.22500.wav
[vorpal8c] - vpl.Sawing.Stereo01.wav
[vorpal8c] - vpl.Select.Mono01.wav
[vorpal8c] - vpl.Select.Mono02.wav
[vorpal8c] - vpl.Ship.Pass.Right.wav
[vorpal8c] - vpl.Ship_01.Pass.Left.wav
[vorpal8c] - vpl.Ship_02.Pass.Left.wav
[vorpal8c] - vpl.Ship_02.Pass.Right.wav
[vorpal8c] - vpl.Shot.9MM.32.khz.ST.wav