

Einleitung

Was war das Projekt?

Das Ziel dieser Arbeit war es, ein Game zu erstellen. Das heisst, es zu erfinden, zu programmieren und die Grafiken zu erstellen. Da es ein eher grosses Projekt ist, habe ich mit Noah Jäggi zusammengearbeitet. Das Produkt sollte unsere gemeinsame Arbeit sein, wir haben uns zusammen überlegt was wir genau machen wollen. Ich sollte programmieren und er sollte die Grafiken erstellen. Meine Arbeit war es unsere Ideen umzusetzen, in Code zu verwandeln und seine Grafiken auf den Bildschirm malen.

Wie ist dieser Text organisiert?

In dieser Arbeit möchte ich meine Vorkenntnisse erklären, wie wir das Projekt geplant und es umgesetzt haben. Ich werde auch einige Beispiele nennen, jedoch nicht alles im Detail ausführen, da der Umfang zu gross wäre und nicht alles spannend ist.

Vorgeschichte, wie ich zu BMX kam (soll das in den Anhang?)

Schon als Kind war es mein Traum Welten zu erstellen. Das drückte ich dadurch aus indem ich mit Bleistift Menschen, Schiffe, Raketen, Burgen und Bunker zeichnete und dann daraus eine Geschichte entstehen liess. Ich nahm den Radiergummi und veränderte die Situation. Ich wollte auch schon immer mal ein Strategiespiel erstellen. So nahm ich Papier und Stifte, Schere, Karton und Leim und bastelte mir eins. Ich musste nun jemanden finden der mir beim Testen half. In den Ferien hatte mein Vater Zeit dafür und wir beginnen an zu Spielen. Nach den ersten paar Runden aber stellten wir fest, dass mein Spiel sehr rechenaufwendig war und mussten stundenlang Zahlen zusammenzählen um auch nur die einfachste Aktion durchzuführen.

Heute kann ich diesen Traum ein Stück besser umsetzen. Mithilfe der Rechenkraft des Computers werden auch die längsten Rechnungen in Sekundenbruchstücken erledigt und man kann sich um das wesentliche kümmern, nämlich die Mechanik des Spiels an sich. Dabei geht es darum die Spielwelt so natürlich wie möglich zu gestalten, dh. mit physikalischen Regeln die Bewegungen der verschiedenen Objekte bestimmen. Falls ein Gegenstand fällt muss das gleich passieren wie in der realen Welt. Zudem sollte man als Spieler nicht durch den Boden hindurchzufallen sondern die Kollision muss entsprechend berechnet werden.

Damit man bisschen besser verstehen kann, wie ich lernte zu programmieren möchte ich hier meine Geschichte erzählen.

Gestartet hat alles nachdem ein Freund von mir aus einem Ferienlager kam, in welchem er lernte wie man eine Website erstellt. Bis zu diesem Punkt hatte ich noch keine Erfahrungen mit Computern ausser den Stunden die ich mit Gamen verbrachte, was jedoch verglichen mit gleichaltrigen wenig war. Dieser Freund gab mir nun eine CD, worauf alles Material war,

welches er während des Lagers bekam. Ich war fasziniert und klickte mich durch die Ordner, schaute mir alles Genau an. Dann nahm ich die Tools die auf der CD waren und startete meine ersten versuche mit HTML zu programmieren. Sehr erfolgreich war ich nicht.

Was ich aber entdeckte war die Commandline von Windows (CMD). Ich erstellte meine ersten Batchdateien und sogar ein kleines simples Autospiele resultierte daraus. Keine Bilder nur Buchstaben. Ein I für den Strassenrand und ein X für das Auto. Aber das war für mich nicht genug, ich wollte mehr lernen.

Eines Tages stiess ich auf Quickbasic. Es ist eine recht alte Sprache und liess sich auch ohne Windows-Betriebssystem brauchen. Auf die Diskette kopiert mit einem Simplen DOS darauf reichte völlig aus. Dies war meine erste Sprache aus der Basic-Familie und ich programmierte auch einige kleine Spiele damit. In einem musste man zum Beispiel mit einem Panzer Gegner Auslöschen ohne selber von ihren Kanonen getroffen zu werden. Nach einer gewissen Zeit wurde mir aber auch das zu langweilig, ich wollte höhere Bildauflösungen und zudem war Qbasic einfach zu langsam.

Auf der Website von Quickbasic war ein Link auf die Seite von Blitzbasic. Das ist eine Sprache für Anfänger, die jedoch recht viel bietet. Man kann sowohl 2D als auch 3D Programme und Games erstellen. Die Kompilierten EXE-Dateien lassen sich aber nur auf Windows-Computern ausführen. Nun konnte ich meine Pläne endlich so richtig umsetzen und viel Erfahrung sammeln. Falls ich Probleme oder Fragen hatte konnte ich fast immer eine Lösung auf dem Forum finden.

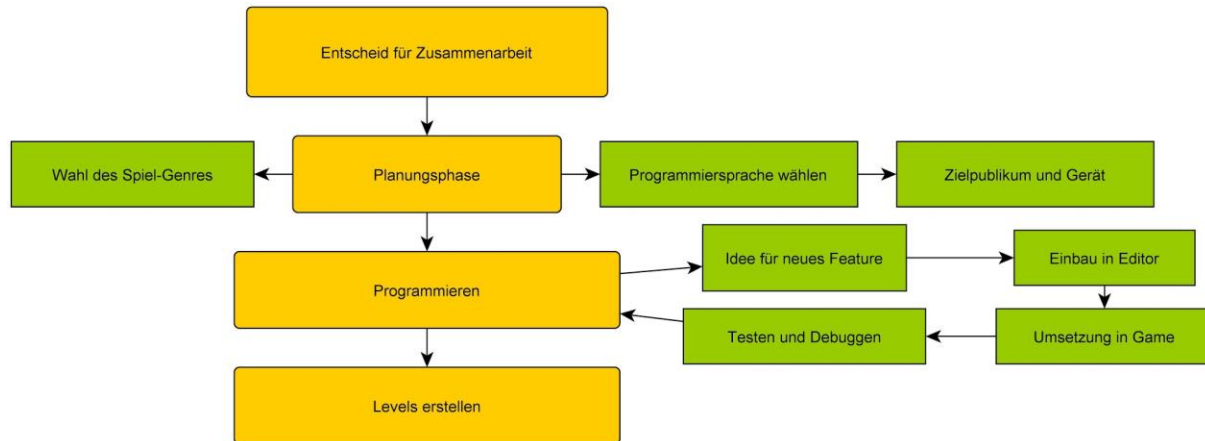
Mit der Zeit stellte ich fest, dass die meisten aus dem BlitzBasic-Forum aber schon den Nachfolger von Blitzbasic nämlich BlitzMax nutzten. Diese Sprache lässt sich mit vielen modernen Sprachen wie C++ und Java Vergleichen. Sie unterstützt Object-Oriented-Programming (OOP) und hat neben DirectX (Grafik-Engine von Windows) auch OpenGL (Cross-Platform). Somit ist die Sprache auch auf Linux und Mac nutzbar. Insgesamt ist BlitzMax sehr brauchbar für Spielentwickler und auch Applikationen lassen sich erstellen. Es hat eine diverse Module für verschiedenstes, zB. Grafik, Audio, Netzwerk-Datenaustausch, Luascript etc. Mit BlitzMax habe ich nun schon diverse Simulationen, Games und Applikationen programmiert und möchte die Vorteile dieser Sprache nicht missen.

Die Phasen des Projekts

Das Projekt lässt sich in 4 Phasen zerlegen:

1. Entscheiden für die zusammenarbeit, Motivation
2. Entscheiden für gerne, Publikum, Entscheiden für grobe Story, ungefähres Spielziel
3. Engine erstellen, testen
4. Levels erstellen / Story ausarbeiten

[bild 4 phasen]



Darstellung der verschiedenen Projekt-Phasen

Motivation (Phase 1: Entscheid für die Zusammenarbeit)

Da es ja mein Hobby ist Games zu programmieren und ich nicht der begabteste Künstler bin was Grafiken angeht bot es sich an diese Maturarbeit mit Noah zu machen, der sich für Pixelgrafiken interessierte. Zudem ist es eine Gelegenheit und natürlich auch eine Herausforderung etwas im Team zu machen, man trainiert dabei aufeinander einzugehen und sich einzubringen.

Planungsphase (Phase 2: Was wollen wir produzieren/erreichen und wie?)

Nachdem wir uns für dieses Projekt entschieden ging es darum unseren wilden Ideen in konkrete Pläne zu verwandeln. Dabei mussten wir uns überlegen auf welchen das Spiel gespielt werden können sollte. Und natürlich die Spielidee an sich. Soll es ein Strategiespiel werden oder lieber ein Adventure oder doch ein Jump'n'run? Soll es highscorebasiert sein oder levelbasiert? Erstellen wir die Welten selber oder lassen wir sie per Zufall vom Computer erstellen? Soll man das Spiel auf dem Handy (Smartphone) oder auf dem Computer spielen können? Welche Mittel sind geeignet für dieses Projekt?

Wahl der Sprache:

Zuerst machte ich mich auf die Suche nach einer Programmiersprache mit der man Games für Smartphones machen kann. Das war aber gar nicht so einfach, denn entweder war die Programmierumgebung nicht genügend Professionell, zu kompliziert, zu teuer oder hatte Einschränkungen, sodass man zum Beispiel nicht eine Hintergrundmusik und zugleich einen anderen Soundeffekt abspielen konnte. Ich konnte nichts geeignetes finden und somit musste ich auf BlitzMax zurückgreifen. Das war keine schlechte Entscheidung, denn so konnte ich meine Erfahrungen damit einbringen und musste mich nicht in eine neue Sprache einarbeiten und somit Zeit sparen. Jetzt wird unser Game nicht für Smartphones sein, was natürlich seine Nachteile mit sich bringt, denn ein Spiel für den Computer muss einiges besser sein und darf nicht gleich simpel sein wie ein Handyspiel. Gamer haben bei Computergames einfach höhere Ansprüche als bei Smartphonegames.

Wahl des Genres:

Dann ging es auch um die Spielidee an sich. Wir hatten verschiedene vage Ideen. Wir wussten teilweise auch was wir zwar gerne gemacht hätten, jedoch nicht genug Zeit im Rahmen der Naturarbeit hatten oder was schlicht zu kompliziert geworden wäre.

Ein Strategiespiel wäre schon immer mein Traum gewesen, jedoch sind meine früheren Versuche eins zu programmieren gescheitert. Meistens lag dies daran, dass man eine Unmenge an verschiedenen Einheiten erstellen muss. Es ist einfach zu kompliziert und damit das es auch gebalanced ist viel zu zeitaufwendig.

Ein Adventure-Game, bei welchem man herausfinden muss was kaputt ist und dann das Problem beheben muss, wäre auch eine Idee gewesen. Wir verwarfen diese aber, da wir für jedes einzelne Puzzle Grafiken und Code hätten produzieren müssen.

Wir wollten eine Geschichte mit kniffligen Aufgaben aber auch Spielfluss ins Game bringen. Somit entschieden wir uns für das Genre Jump'n'run. Bekannt aus diesem Genre ist Super-Mario. Man geht mit seinem Hauptcharakter durch die Welt, überspringt Gräben, weicht gefährlichen Gegnern aus und hat irgendeine Mission zu erfüllen.

Nun ging es darum, ob das Spiel levelbasiert oder highscoreorientiert ist. Wollen wir die Levels von Hand erstellen oder lieber ein Algorithmus programmieren welcher uns eine zufällige Gamewelt erstellt? Da wir eine Geschichte ins Spiel bringen wollten und es uns wesentlich einfacher erschien die Levels selber zu erstellen, entschieden wir uns dafür, eine lineare Geschichte zu haben. Das heisst, dass ein Level nach dem andern kommt und man eines nach dem anderen in einer festen Reihenfolge durchspielt. Das eigentliche Spiel besteht also aus einer Serie von Leveln.

Finden von etwas was original ist:

Jump'n'runs gibt es schon lange und somit gibt es auch schon sehr viele verschiedene Varianten und Versionen. Wir sahen uns Spiele und Videos über Spiele an um zu sehen was schon existiert und Ideen zu sammeln. Unser Ziel war es nicht einfach ein bestehendes Spiel zu kopieren. Wir wollten etwas was es noch nicht so gab, etwas originales. Oder zumindest sollte man nicht sagen können es sei gleich wie ein anderes bekanntes Spiel. Somit suchten wir für spezielle, neuartige Spielelemente.

Uns faszinierte die Idee, dass man sich als Spieler während des Spiels verändert und man je nach dem aktuellen Stadium andere Fähigkeiten hat. Als die verschiedenen Stadien wählten wir Elemente wie Wasser, Stein, Pflanze, Feuer, Dampf und einige andere. Wir hatten noch nicht für alles eine konkrete Vorstellung aber die Feinheiten wollten wir ausarbeiten sobald wir soweit waren sie zu implementieren.

Erstellen der Engine(Phase 3: Programmieren, Erstellen alle Features)

Nachdem wir uns für die Programmiersprache, das Spielgenre und auch gewisse detailliertere

Ideen hatten ging es darum das Projekt umzusetzen. Das Ziel des Projekt ist es Levels zu haben und diese spielen zu können. Zuerst jedoch muss die Arbeitstruktur festgelegt werden.

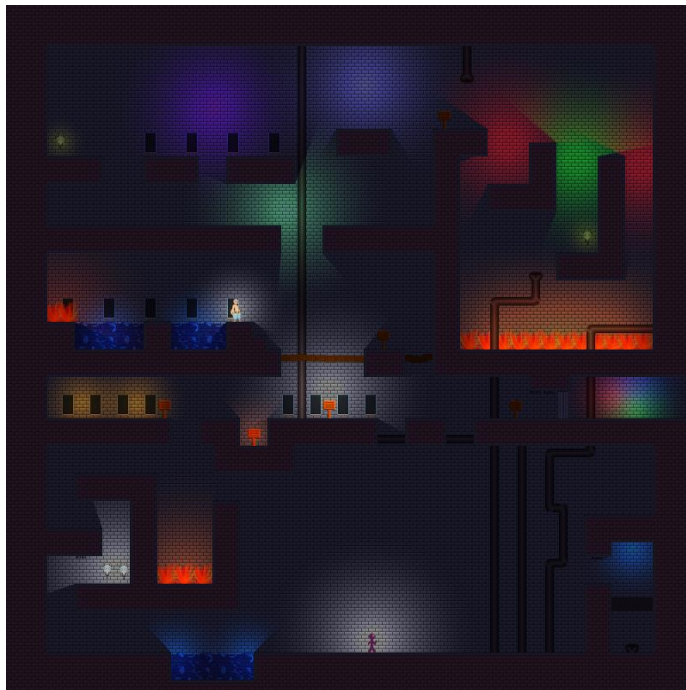
1. Editor, Level und Game

Wir haben uns für ein Levelbasiertes Spiel entschieden, dh. der Spieler wird ein Level nach dem andern Spielen, in jedem Level soll es Herausforderungen geben und am Ende zu einem Ziel kommen. Das bedeutet, dass sich die ganze Arbeit um die Levels dreht. Es geht darum die Levels für den Spieler zu erstellen. Irgendwie muss man diese Levels aber erstellen. Um das Leveldesign zu vereinfachen braucht es einen Level-Editor. In diesem hat man die Möglichkeit alles einzurichten und einzustellen, Objekte zu Platzieren und zu entfernen. Schlussendlich wird das Level gespeichert in einem Dokument. Um das Level zu spielen braucht es ein anderes, separates Programm, eine Interpretations-Maschine, die das Level liest und dann spielbar macht. Dieses Programm nenne ich das Game.

[Bild: Editor, Level, Game]



Zusammenhang zwischen Editor, Level und Game





Zwei Beispiele für Levels, das eine mit, das andere ohne Lichter. In beiden ist es das Ziel von links nach rechts zu kommen.

In erster Linie geht es mir nicht darum die Levels zu erstellen sondern die Vielfalt an Objekten und Funktionen für die Levels zu erstellen. Das heisst eigentlich, dass ich zuerst die Sprache für die Levels erstelle um dann mit dieser die Levels zu designen. Irgendwie müssen die Levels ja definiert werden. Um möglichst viele Möglichkeiten zu haben, muss zuerst diese Sprache ausgebaut werden und damit auch der Editor der die Levels schreibt und das Game, welches das Level danach liest, interpretiert und ausführt.

Die Idee ist es zuerst alle Bausteine (Features, Objekte, Funktionen) zu erstellen um danach damit das Haus (Levels) zu bauen. Natürlich müssen aber die Bausteine getestet werden bevor sie dann verwendet werden für das Haus.

2. Parallele Arbeit an Game und Editor:

Diese Trennung von Game und Editor bedeutet aber, dass man zwei Programme parallel programmieren muss. Wenn man etwas neues zur Gamemechanik hinzufügt muss es möglich sein diese mit dem Editor ins Level zu bringen und das Game muss dieses neue Feature auch umsetzen können. Teile des Codes können natürlich einfach kopiert werden. Jedoch muss man beim Editor die Oberfläche programmieren mit der man die Objekte verwaltet und modifiziert. Der Editor muss aber nicht berechnen wie der Spieler fällt, das muss aber auf der Seite des Games implementiert werden.

Nun stellt sich die Frage was soll zuerst kommen, der Editor oder das Game. Sicher nicht das Game, denn das Game ohne ein Level bringt nichts. Somit muss zuerst der Editor erstellt werden. Jedoch nur mit den wichtigsten Grundbausteinen, ohne denen nicht funktionieren würde, wie zum Beispiel Kollisionsboxen und der Spieler. Sobald dies gemacht ist muss man testen ob auch alles funktioniert, man will ja nicht nach Fertigstellung des Editors feststellen müssen, dass man etwas nicht bedacht hat und dass man nochmals von vorne beginnen muss. Deshalb muss man sobald wie möglich das Game erstellen, welches genau denselben Umfang

an Funktionen hat, sodass die erstellten Testlevels gespielt werden können. Danach fügt man dem Editor einige Funktion hinzu und danach dem Game, testet erneut. Dieser Prozess zieht sich nun durch bis das Schlussendliche Produkt fertiggestellt ist mit allem was benötigt wird.

In diesem Repetitiven Prozess des Hinzufügens neuer Objekte und Funktionen fand auch ein grosser Teil Zusammenarbeit zwischen mir und Noah statt. Wir hatten neue Ideen oder auch solche die wir schon immer umsetzen wollten. Diese mussten dann in eine Form gebracht werden, welche klar definiert ist damit sie auch umsetzbar ist. Dann wurden sie in den Editor integriert und danach ins Game. Darauf testeten wir die neuen Features bis aufs Detail. Gab es Fehler oder etwas das uns nicht gefiel, so musste ich die Idee umformulieren und dann im Editor und danach im Game einsetzen.

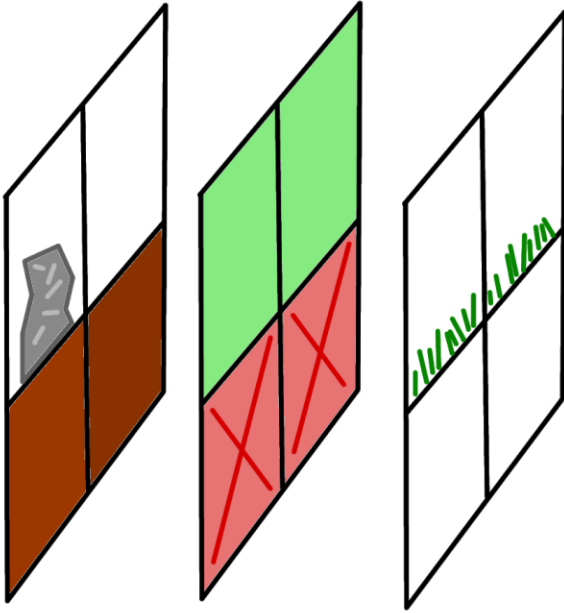
3. Basisstruktur der Gamemechanik:

Nachdem ich nun die allgemeine Vorgehensweise beschrieben habe, geht es mir diesem Abschnitt darum, die Struktur der Gamemechanik zu erklären und zu schildern wie die einzelnen Bausteine zusammengesetzt sind.

3.1 Das Raster, die Grundlage des Levels

Die Grundlage eines Levels ist ein schachbrettartiges Raster. Jedes Feld dieses Rasters besteht aus zwei Bildern. Eines ist im Hintergrund, also hinter dem Spieler und eines ist im Vordergrund, also vor dem Spieler. Dies ermöglicht, dass der Spieler beispielsweise vor einem Stein aber gleichzeitig hinter Gräsern durchläuft. Zudem sind gewisse dieser Felder massiv, haben also eine Kollisionsbox und andere sind frei. Somit lassen sich Böden, Wände und Decken gestalten. Dieses Raster ist fix, dh. es kann nicht verändert werden während dem man das Level durchspielt.

[BILD]



2x2 Feld zur darstellung des Rasters: links (hinten, Hintergrund), mitte (Kollisionsebene), rechts (vorne, Vordergrund)

3.2 Objekte, die bringen die Dynamik ins Spiel

Das Raster ist statisch, ein Spiel muss aber dynamisch sein, es muss sich etwas bewegen. Deshalb besteht ein Level aus dem Raster und aus einer Menge an Objekten welche sich bewegen, handeln und somit das Spiel interessant machen.

3.2.1 Der Spieler

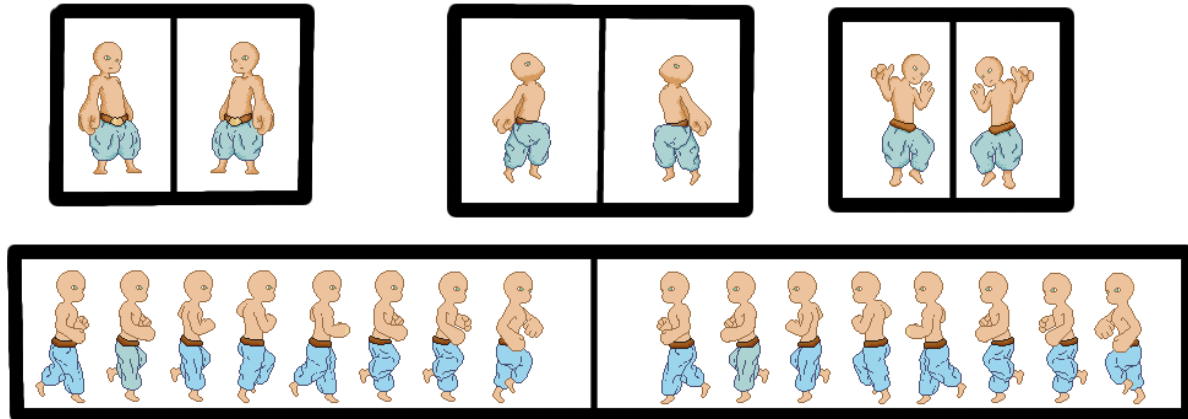
Der Spieler ist ein Objekt, denn er soll sich ja bewegen und handeln können. Er wird mit den Tasten gesteuert. Wir entschieden uns für die WASD-Steuerung. Anstelle der Pfeiltasten verwendet man die Tasten W, A, S und D um sich zu bewegen. Teilweise wird auch die Spacebar gebraucht um eine Spezialfunktion zu aktivieren.

Als ein Zentrales Spielelement haben wir uns entschieden, dass der Spieler sich in verschiedene Elemente verwandeln kann. Zu Beginn ist man normal, wird aber während dem Spiel zu Wasser, Feuer, Pflanze etc. Jeder dieser Modi hat seine eigenen Funktionen. So kann man als Wasser einen Charge-Sprung machen und den Wänden und der Decke entlang gehen. Als Feuer kann man hölzerne Gegenstände anzünden. Als Pflanze kann man zwar nicht springen aber hat dafür andere Vorteile.

Aus diesen Elementen könnte man schon fast ein Spiel erstellen. Der Spieler hüpfte durch die Welt und geht sie erkunden. Jedoch ist noch kein Ziel gegeben. Das ist jedoch ein sehr wichtiger Aspekt eines Spiels, man sollte ja eine Motivation haben und eine Herausforderung.

Sehr wichtig sind beim Hauptcharakter auch die verschiedenen Animationen. Schliesslich will man ja, dass er sich bewegt und nicht nur einfach über den Boden gleitet.

[Bild] Die Animation für den Normal-Modus: Stehen, Springen, Fallen und Gehen, je links und rechtsorientiert:



links-oben: stehen, mitte-oben: springen, rechts-oben: fallen, unten: gehen (Animations-Strip)

3.2.2 Boxen, sie bringen die Interaktion verschiedener Elemente ins Spiel

Somit kommen wir zu den Boxen. Sie übernehmen alles was eine Interaktion mit dem Spieler beinhaltet. Sie sind definiert durch eine Position auf dem Koordinatennetz auf dem Level, eine Instanz die den Block zeichnet und eine die ihn berechnet. Sie sind nicht mit den Boxen des Rasters zu verwechseln, welche keine Funktion haben als Kollision.

Diese Blöcke haben diverse Aufgaben. Zuerst dienen sie als zusätzliche Kollisionsboxen, welche sich nicht an das Quadratische Raster halten müssen. Zudem kann man sie abbrennen lassen. Die Boxen können aber auch Türen und Schalter, Hinweisschilder, Personen welche mit einem Sprechen, gefährliche Pflanzen die einen aufsaugen, Teleporter und vieles mehr sein. Auch Wasser, die Beleuchtung und die Musik und Soundeffekte werden von ihnen verwaltet. Man kann auch den Modus des Spielers anpassen, zum Beispiel mit einem Feuer den Spieler in den Feuer-Modus bringen. Für Gewisse Funktionen kann man auch verschiedene Boxen kombinieren.

-Beispiel Pflanzenmonster:

Ein gutes Beispiel für die Verwendung von Boxen ist das Pflanzenmonster. Es besteht nicht nur aus einer Box sondern aus mehreren, nämlich aus einem Bild, einer Zone in der man aufgesogen wird wenn man aus Wasser ist, einer Kollisionsbox, damit man nicht hindurch kann und einer Soundbox, welche das Tier in regelmäßigen Abständen brummen lässt.

3.2.3 Feuervogel

Der Feuervogel sitzt irgendwo und der Spieler läuft unten durch. Der Vogel stürzt sich hinunter auf den Spieler und versucht ihn anzuzünden. Ist man aber im Pflanzenmodus sehen sie einen nicht.

3.2.4 Partikel:

In vielen Games hat es eine Unmenge an Partikeln, welche keine andere Funktion erfüllen als dem Spiel einen besseren Graphischen Eindruck verleihen. Je mehr Partikel umso besser. Jedoch sind Partikel in grossen Mengen sehr rechenintensiv. Das Spiel beginnt zu stocken und somit gilt es, die richtige Balance zu finden.

Ein gutes Beispiel für die Partikel ist beim Charge-Sprung zu finden. Während dem Sprung kommen unten aus dem Spieler Wasserpartikel raus. Wenn sie auf dem Boden aufprallen spritzen sie auf alle Seiten. Das ist zwar ein Detail, welches man nicht direkt wahrnimmt, es kreiert aber eine bessere Stimmung und verleiht physikalische Glaubwürdigkeit.

[BILD]



von links nach rechts: 1&2: Wasser-Jump hinterlässt Wassertropfen welche auf dem Boden aufprallen und zerspritzen, 3: Feuerpartikel welche aus dem Boden kommen.

Erstellen der Levels (Phase 4: Das Spiel mit Inhalt füllen)

Nachdem die Gamemechanik programmiert und ausführlich getestet ist und die Bilder zur Verfügung stehen, geht es darum aus diesen Bausteinen ein Haus zu bauen. Sollte irgend einem plötzlich auffallen, dass noch ein Baustein fehlt, kann man diesen natürlich noch erstellen.

Beim erstellen der eigentlichen Levels ist man seiner Kreativität überlassen. Für diese Arbeit steht dieser Teil aber nicht im Zentrum, zum jetzigen Zeitpunkt sind auch noch nicht alle Levels erstellt, jedoch eine kleine Anzahl besteht. Es handelt sich nur um Test- oder Demonstrations-Levels.

fortgeschrittener Teil

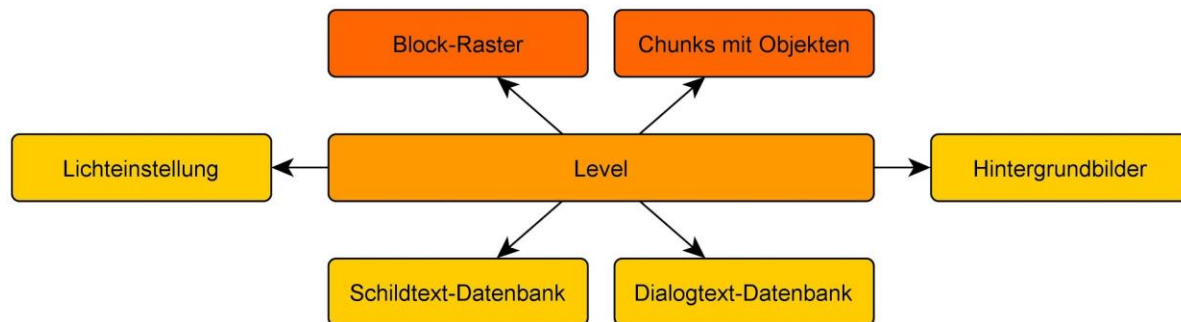
Dieser Teil ist gedacht um etwas mehr Details zu erklären. Er wird aber auch etwas komplizierter sein und ist somit nur für Leute mit Vorwissen in Programmieren und OOP geeignet. Gewisse Teile werden Wiederholung sein und etwas genauer ausgeführt sein.

1. Level

Das Level besteht aus dem quadratischen Raster und Objekten. Die Informationen dieser zwei Teile sind in einem eigenen Datenformat gespeichert. Raster und Objekte werden in je einer separaten Datei gespeichert, damit man einfach die Datei mit den Objekten löschen kann, falls man alle Objekte aus einem Level löschen möchte. Dies kann nützlich sein, wenn man das Dateiformat der Objekte verändert um neue Objekte speichern zu können.

Es gibt zudem ein zusätzliches Format in dem ein laufendes Level gespeichert werden kann. Es muss anders sein, denn gewisse Variablen kommen erst ins Spiel wenn es gestartet wird.

[Bild Level]



Die verschiedenen Komponenten, welche das Level ausmachen

1.1 Levellayers

Das Game wird in sieben schichten gezeichnet. In der hintersten sind Objekte, welche vom zweiten Layer verdeckt wird. Dieser ist der hintere Layer des Rasters. Dann kommen wieder Objekte, dann die Ebene des Spielers. Dann wieder Objekte worauf der vordere Layer des Rasters kommt und dann wieder Objekte.

Hinter der ganzen Umgebung kann man noch beliebig viele Hintergrundbilder einfügen, welche sich je nach deren Abmessung verschieben. Dies gibt einen dreidimensionalen Tiefeneindruck.

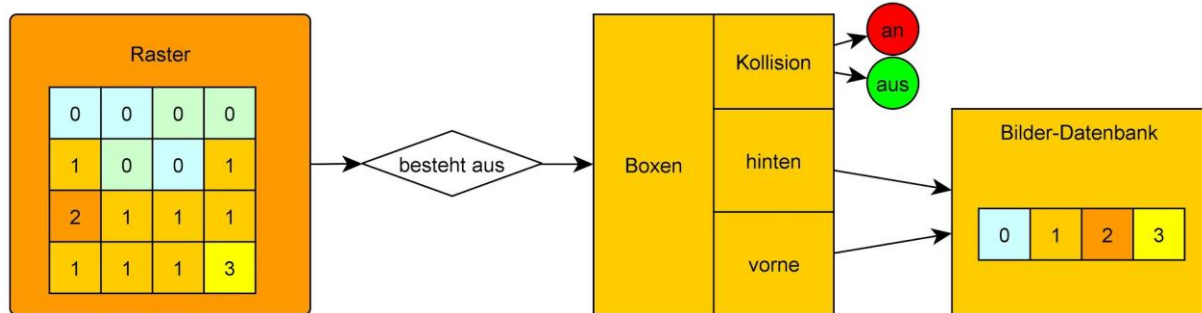
1.2 Raster

Das Game ist basiert auf dem rasterartigen, schachbrettartigen Feld. Jede Box des Feldes hat zwei Bilder aus einer Liste. Diese werden wie oben beschrieben, eins vor und eins hinter dem Spieler gezeichnet. Zudem bestimmen die Felder wo Wände, Boden und Decke ist, also haben

sie eine Kollisionsbox.

Natürlich könnte man das auch alles mit Objekten lösen welche frei von einem Raster sind, jedoch ist es Ressourcensparender mit weniger Objekten und auch einfacher zum designen der Levels.

[Bild Raster]

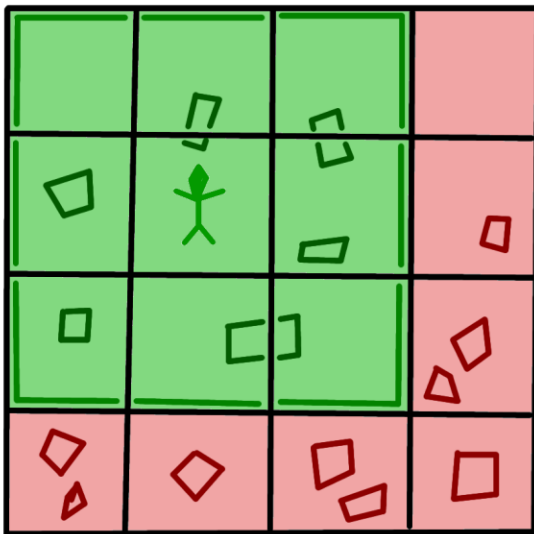


Aufbau des Rasters aus Boxen

1.3 Objekte

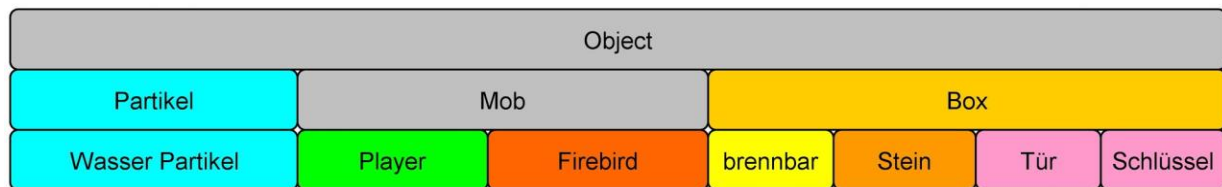
Alles was nicht im Raster des Levels ist ist von der Klasse der Objekte abgeleitet. Damit es ressourcensparender ist werden sie in sogenannte Chunks eingeteilt. Das heisst es gibt ein grossmaschiges Raster und jedes der Felder darin hat eine Liste mit den Objekten welche in diesem Abschnitt des Levels sind. Falls ein Objekt den Ort seines Chunks verlässt muss das Objekt natürlich auch den Chunk wechseln. Der Vorteil von Chunks ist, man muss nur Chunks berechnen welche nahe am Spieler sind. Natürlich gibt es Objekte die immer berechnet werden müssen. Dafür gibt es noch eine zusätzliche Liste für diese Objekte. Damit die Objekte auch in verschiedenen Layers gezeichnet werden können, gibt es pro Chunk fünf Listen.

[Bild chunks]:



Das Raster Stellt die Chunks dar. Grün ist aktiv, rot passiv. Der Spieler ist in der Mitte der aktiven Zone. Grüne Objekte werden berechnet, rote nicht.

[Bild Vererbung]:

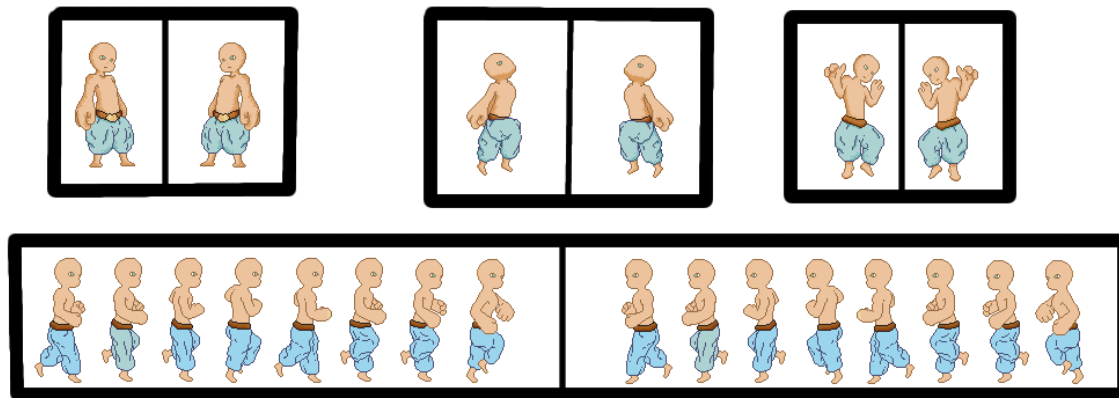


Vererbung der Klassen: von oben nach unten, Untere sind von Oberen abgeleitet.

1.3.1 Player

Der Player ist das einzige Objekt, welches vom Spieler direkt angesteuert wird. Der Player kann seinen Modus ändern, wenn er beispielsweise ins Feuer geht, wird er selber zu Feuer. Für jeden dieser Modi gibt es es ein eigenes Objekt, welches die Informationen für den Modus enthält (Gravitation, Geschwindigkeit, Sprungkraft und spezielle Fähigkeiten). Zudem hat jeder Modus seine eigenen Animationen für Stehen, Gehen Springen und Fallen.

[Beispiel Bild]



links-oben: stehen, mitte-oben: springen, rechts-oben: fallen, unten: gehen (Animations-Strip)

1.3.2 Firebird

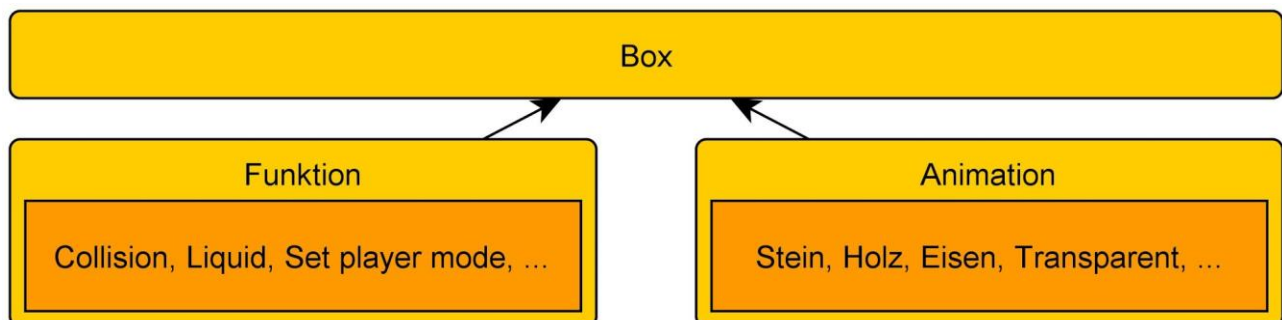
Er wartet bis der Player genügend nahe ist. Dann überprüft er ob dieser nicht im Pflanzenmodus oder im Feuermodus ist. Dann stürzt er sich hinab auf ihn. Dabei befolgt er einer Berechnung und für den Spieler ist es schwierig auszuweichen, besonders wenn es mehrere Firebirds hat.

Er hat zwei Bilder, eins fürs Sitzen und eines fürs Herunterstürzten welches je nach Richtung noch gespiegelt wird. In der Zukunft könnte man anstelle der Bilder noch Animationen einfügen.

1.3.3 Boxen

Die Boxen sind wohl das komplexeste Konstrukt des Games. Jede Box wird von einer Instanz verwaltet und von einer anderen wird sie gezeichnet. Dank dieser Trennung kann man mehr kombinieren und muss nicht bis für jede Funktion die Bilder separat laden.

[Bild Box]



Struktur der Box, bestehend aus Funktion und Animation

Nun werde ich jede dieser Funktionen kurz erklären:

1.3.3.1 Image

Hat keine Funktion, nur das Bild oder die Animation der Box wird angezeigt. Falls gewünscht kann man einstellen, das das Bild oder die Animation als Beleuchtung verwendet wird. Somit passiert mit dem Bild in etwa dasselbe wie bei den Lamps.

1.3.3.2 Check Point

Sobald der Player diese Box betritt wird das Level gespeichert. Sollte der Player sterben wird er zu diesem Punkt zurückgesetzt, das gespeicherte Level wird geladen. Hier wird mit dem zweiten Dateiformat gearbeitet, sodass alle Variablen gespeichert sind.

1.3.3.3 Level Completed

Wenn der Spieler diese Box betritt, ist das Level abgeschlossen. Wenn dieses Level zum ersten mal abgeschlossen wurde, wird das nächste Level freigeschaltet.

1.3.3.4 Fire Surprise

Normalerweise sieht man nicht von dieser Box. Erst wenn der Spieler sie betritt kommt Feuer von unten und zünden den Player an. Die Feueranimation ist von Partikeln gezeichnet was eine realistischere Wirkung gibt. Wie der Name „Fire Surprise“ sagt, soll hier auch ein Überraschungseffekt entstehen.

1.3.3.5 Collision

Diese Box ist eine einfache Kollisionsbox. Man kann aber gewisse Seiten deaktivieren und somit beispielsweise Plattformen bauen, welche von unten erreichbar sind, man aber nicht durch sie hindurchfallen kann.

1.3.3.6 Collision Mode Sensitive

Sie sind gleich wie die normalen Kollisionsboxen, sind aber je nach aktuellem Modus des Players aktiviert oder nicht. Hiermit kann zum Beispiel ein Gitter realisiert werden. Im Wasser-Modus fällt man durch, im Normalzustand nicht.

1.3.3.7 Set Player Mode

Wenn der Spieler sie betritt, erhält er einen anderen Modus.

1.3.3.8 Death Zone

Es gibt verschiedene Einstellungen für diese Funktion. Es lassen sich zum Beispiel Spikes erstellen, welche den Player je nach Modus umbringt oder auch nicht. Für das Pflanzentier wird die Einstellung verwendet, bei der dem Player das Wasser abgesogen wird und er als Konsequenz davon stirbt. Die Funktionen sind hier sehr einfach im Code erweiterbar.

1.3.3.9 Liquid

Sie kann auf Wasser oder Lava eingestellt werden. Momentan funktioniert aber nur die Wasser-Funktion. Sie verleiht dem Wasser die Eigenschaften, man kann in der Box schwimmen und wird zu Wasser verwandelt.

1.3.3.10 Teleporter

Mit dem Teleporter kann man den Player oder in Zukunft eventuell auch andere Objekte teleportieren, dh. deren Position verändern. Man kann mit ihr die Illusion einer Tür erzeugen mit der man ein Haus betritt oder eine Pipeline durch die man gesogen wird. Während dem Teleportieren ist man im Transparenz-Modus und am Ende der Teleportation kann der Modus beliebig eingestellt werden.

1.3.3.11 Lamp

Mit Hilfe der Lightengine kann Licht im Game zu Echtzeit berechnet werden. Mit der Funktion Lamp kann man eine Lichtquelle erstellen welche dann berechnet wird.

1.3.3.12 Music

Mit ihr kann die Hintergrundmusik eingestellt werden. Die Box enthält den Namen der Musik mit welchem dann die Musik gesucht und abgespielt wird.

1.3.3.13 Music Changer

Wird diese Box betreten wird die Hintergrundmusik ersetzt mit einer anderen.

1.3.3.14 Sound

Beim Betreten wird ein Soundeffekt abgespielt.

1.3.3.15 Sound Repeater

In definierten Zeitintervallen wird ein Soundeffekt mit Hilfe der Soundengine je nach position rechts- oder linkslastig und lauter oder leiser abgespielt wird.

1.3.3.16 Light Handler

Man kann die Standardbeleuchtung des Levels einstellen. Zudem kann man wählen ob diese Einstellung von Beginn des Levels oder erst nach der Betretung aktiviert wird. Diese Funktion kann zum Beispiel für den Kontrast von Höhle und draussen benutzt werden.

1.3.3.17 Dialogue

Mit dieser Funktion kann zum Beispiel eine Person ins Level gestellt werden die mit einem Spricht und einem Tips gibt. Die Dialogtexte sind in einer Liste gespeichert und mit dem Namen welcher in der Dialog-Box gespeichert ist kann der entsprechende Text gefunden und angezeigt werden.

1.3.3.18 Dialogue Changer

Nachdem man den Tip befolgt hat und irgendwo hingegangen ist kann der Dialogtext respektive der Name für den Text der Dialog-Box ausgetauscht werden. Dazu müssen die beiden Boxen denselben Namen haben.

1.3.3.19 Sign

Man kann am Wegrand Schilder mit Text aufstellen. Diese werden auch in einer Liste gespeichert und Mithilfe des Namens aufgerufen.

1.3.3.20 Verbrennbare Boxen

Sie sind von den Normalen Boxen abgeleitet, haben aber eine zusätzliche Animation für das Abbrennen und müssen deshalb von denen Getrennt werden. Berührt der Player im Feuer-Modus die Box beginnt diese zu brennen und verschwindet danach. Zudem zünden sich brennende Boxen gegenständig ab. Somit kann zum Beispiel eine ganze Brücke abbrennen.

1.3.3.21 Stein Boxen

Sie können vom Spieler im Stein-Modus zerschlagen werden. Dafür braucht es aber eine Animation für das Zerfallen der Box. Somit braucht es gleich wie bei den Verbrennbahen Boxen auch eine separate Programmierung.

1.3.3.22 Tür und Schlüssel Boxen

Tür und Schlüssel sind durch einen gemeinsamen Namen verbunden. Die Türen haben je eine Animation für das öffnen, offen stehen, schliessen und geschlossen sein. Der Schlüssel hat je nachdem ob die Türe offen oder geschlossen ist eine andere Animation.

Mit diesem System kann man einfache Rätsel erstellen in der man den Schlüssel versteckt und man ihn suchen muss. Oder man muss herausfinden auf welchen Knopf man stehen muss um die Tür zu öffnen.

1.3.4 Partikel

Partikel verbessern die Stimmung und geben den Eindruck, dass die Gamewelt physikalisch korrekt funktioniert. Schwierig ist es jedoch die richtige Anzahl zu finden. Je mehr Partikel desto besser der Effekt aber je mehr Partikel desto mehr Leistung wird verbraucht. Details auf welche ich stolz bin ist der Partikel-Effekt wenn man im Pflanzen-Modus die Blätter abschüttelt oder wie die Wassertropfen auf dem Boden aufprallen und verspritzen.

[3 Bilder (Wasser-)Partikel]



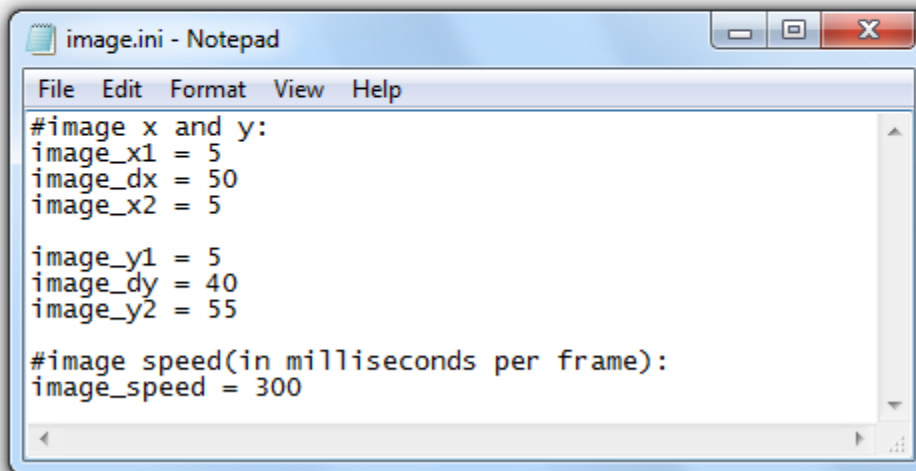
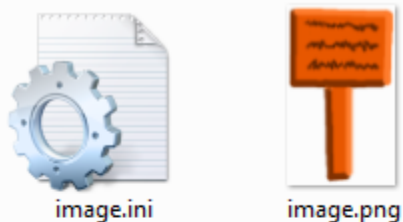
von links nach rechts: 1&2: Wasser-Jump hinterlässt Wassertropfen welche auf dem Boden aufprallen und zerspritzen, 3: Feuerpartikel welche aus dem Boden kommen.

2. Datafilehandler

Damit gewisse Einstellungen für diverse Ladeprozesse einfacher zu handhaben sind habe ich ein einfaches Variablen-System erstellt in dem man in einer Datei Variablennamen Werten gegenüberstellen kann. Es ist so konzipiert, dass man auch ohne Problem Änderungen von Hand durchführen kann. Der Vorteil ist, dass man sehr dynamisch Einstellungen anpassen kann. Hier ist auch ein grosser Teil der Zusammenarbeit mit Noah. Er sollte seine Grafiken so einfach wie möglich einfügen können und die wichtigen Informationen dem Game selber geben können.

2.1 Beispiel: Boxen

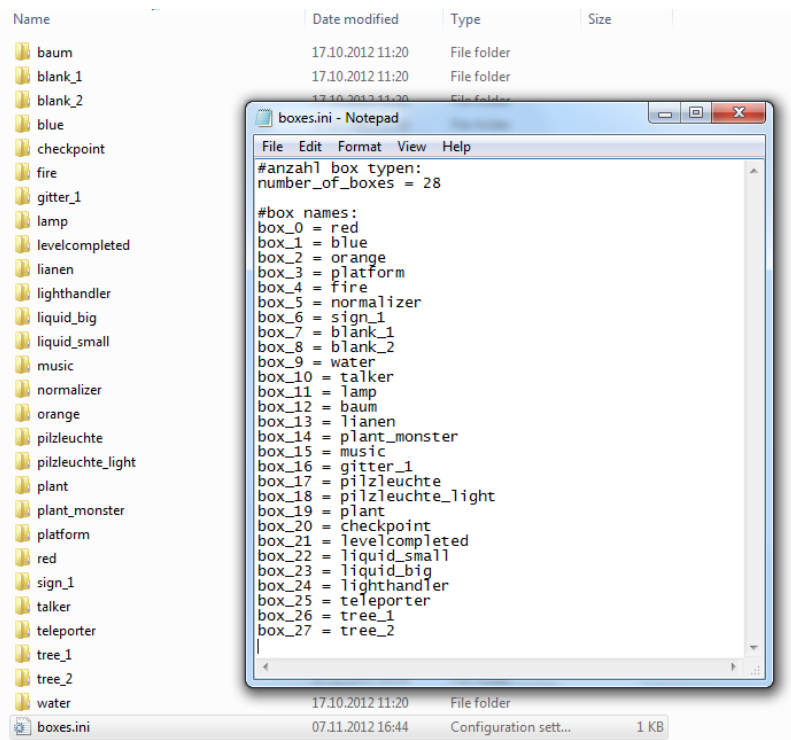
In demselben Ordner, in welchem das Bild der Box (hier ein Schild) sich befindet ist auch das Dokument welches die Eigenschaften dieses Bildes Definiert. Das Wird durch setzen von Werten in Variablen realisiert.



Screenshot aus dem Ordner mit dem Bild für eine Box

2.2 Beispiel: Liste aller Boxen

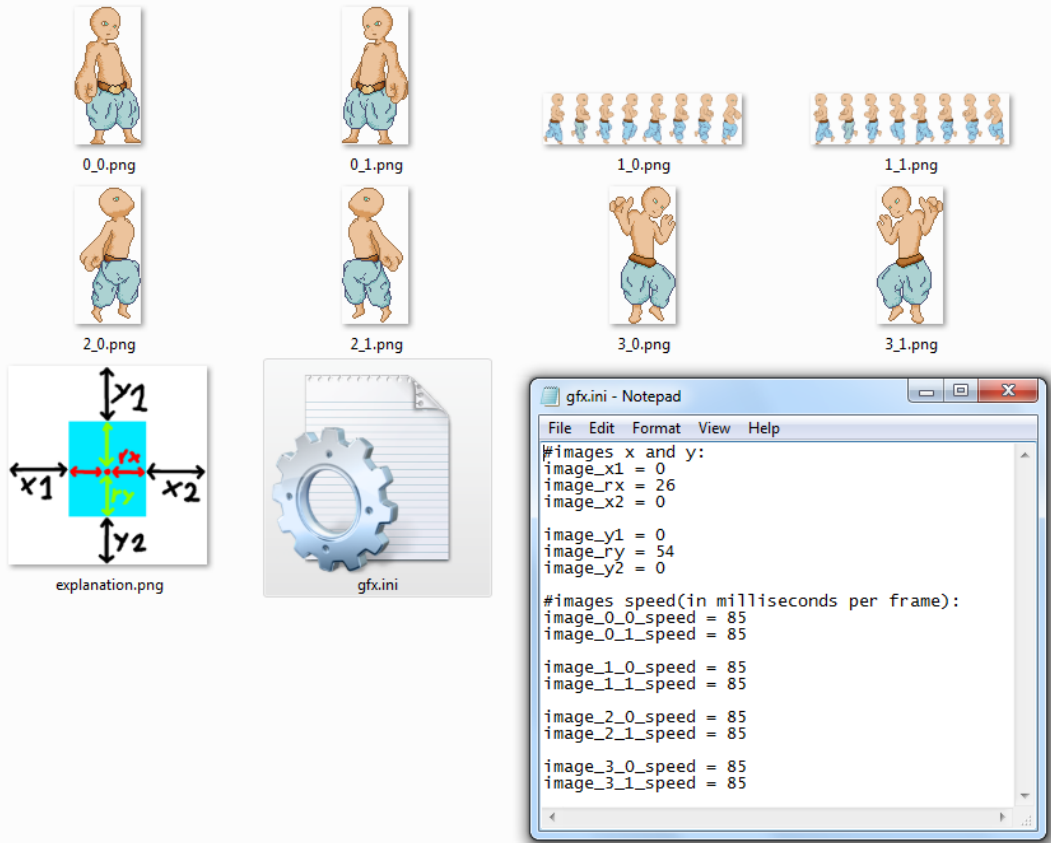
Irgendwie muss das Game wissen woher es die Bilder für die Boxen bekommt. Auch dies ist in einem File definiert. Das File gibt an wieviele Boxen es gibt und danach wie sie heissen.



Screenshot aus dem Ordner mit den Boxen

2.3 Beispiel: Animationen für Spieler

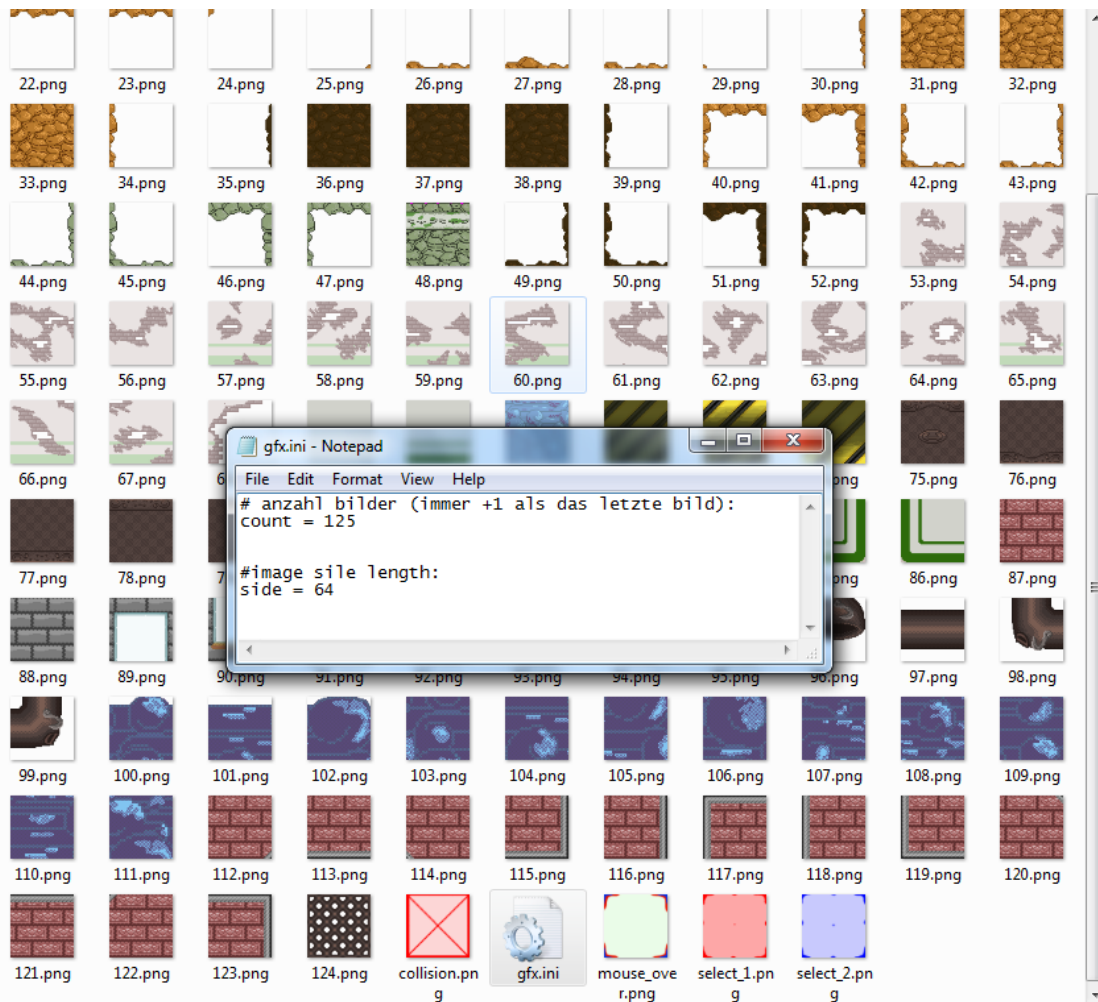
Im Ordner für die Animationen des Spielers im Normal-Modus befindet sich neben den Bildern auch noch ein File welches die Abmessungen der Bilder befinden. Dies ist notwendig um die Animations-Strips richtig zu verschneiden. Zudem wird die Abspielgeschwindigkeit in Millisekunden pro Bild angegeben. Zusätzlich ist im Ordner noch ein kleines Bild damit Noah weiss wie er die Angaben machen muss, sollte er es vergessen haben.



Screenshot aus dem Ordner mit den Bildern für die Animation des Spielers im Normal-Modus

2.4 Beispiel: Bilder für das Raster














Damit das System für die Bilder des Rasters möglichst dynamisch ist kann man die Anzahl der Bilder verändern. Man könnte auch die Abmessung verändern, jedoch ist sie standardmässig auf 64x64 Pixel gesetzt. Die Bilder sind nummeriert und werden geladen, startend bei 0 und endend bei "count"-1.

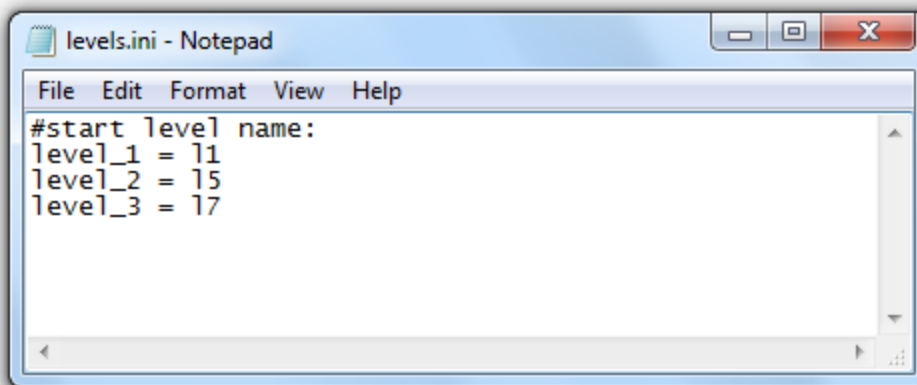


Screenshot aus dem Ordner mit den Bildern für das Raster

2.5 Beispiel: Levels

Es muss auch definiert werden welche Levels gespielt werden können und in welcher Reihenfolge. Das File verweist auf die Ordner, welche die Levels enthalten.

Name	Date modified	Type	Size
 I1	17.10.2012 11:20	File folder	
 I2	17.10.2012 11:20	File folder	
 I3	17.10.2012 11:20	File folder	
 I4	17.10.2012 11:20	File folder	
 I5	18.10.2012 16:00	File folder	
 I6	25.10.2012 15:42	File folder	
 I7	07.11.2012 16:05	File folder	
 I8	07.11.2012 16:38	File folder	
 SLOT_0	17.10.2012 11:20	File folder	
 SLOT_5	17.10.2012 11:20	File folder	
 SLOT_19	17.10.2012 11:20	File folder	
 vapour	17.10.2012 11:20	File folder	
 levels.ini	07.11.2012 16:05	Configuration sett...	1 KB



Screenshot aus dem Ordner welche die Levels enthält

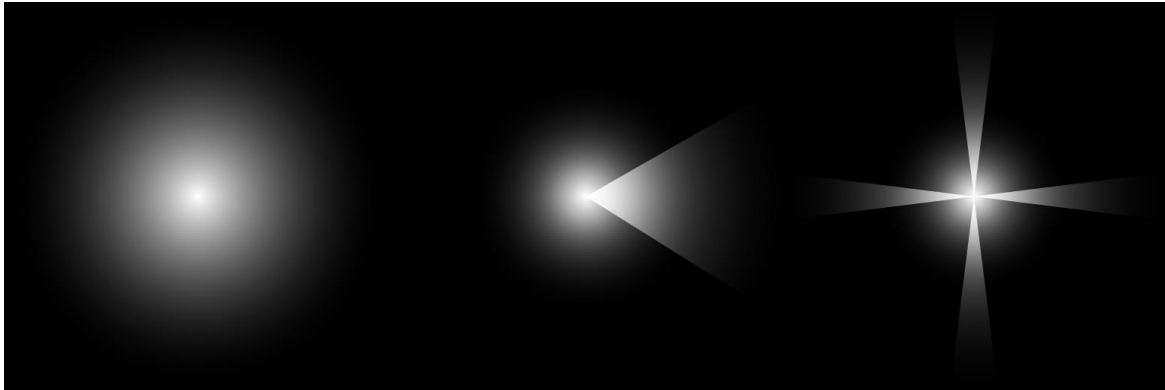
3. Lightengine

Mit Hilfe von OpenGL kann man viel machen, das zeigt auch die Lightengine im Game. Licht und Schatten werden zu Echtzeit berechnet. Man kann Lichter im Level platzieren und diese verschieben. Zudem werfen die Felder des Rasters welche die Kollision an haben Schatten. Nun werde ich den Berechnungsvorgang schrittweise erklären:

3.1 Bild mit Helligkeitsstufen:

Für jedes Licht braucht es ein Bild mit Helligkeitsstufen. Der Lichtwurf ist durch die Grösse, Form und Farbe dieses Bildes definiert. Das Bild kann auch einfach in Graustufen sein und dann im Game eingefärbt werden. Das Licht muss nicht zwingend einen kreisförmigen Lichtwurf haben, im Game jedoch werden nur kreisförmige Lichter verwendet. Was jedoch wichtig ist, man muss darauf achten, dass der Helligkeitsverlauf nicht linear sondern exponentiell oder quadratisch ist, damit wenn sich mehrere Lichter überlappen keine seltsamen unerwünschte Effekte entstehen.

[Bild 1, 3 Lichter]



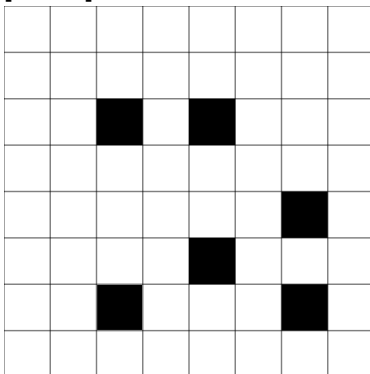
von links nach rechts: kreisförmiger Lichtwurf, Taschenlampenlichtwurf, Warnlicht

Dieses Bild wird nun auf einen separaten Buffer gezeichnet, damit das Originale Bild bestehen bleibt wenn man nachher die Schatten auf den Buffer und nicht auf das Bild.

3.2 Jedes Objekt, welches Schatten wirft finden

Die Objekte welche Schatten werfen sind bis jetzt nur die Boxen des Rasters, welche die Kollision an haben und somit etwas Solides sind.

[Bild 2]

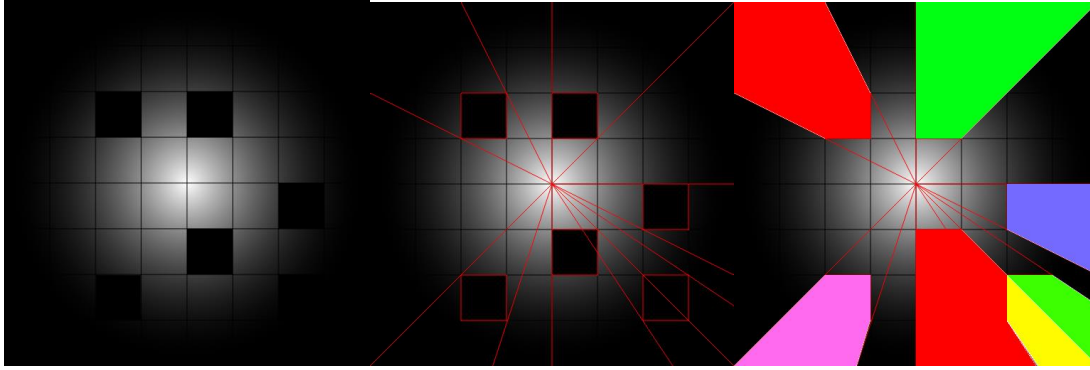


Schwarze Kästchen schellen Schattenwerfende Objekte dar.

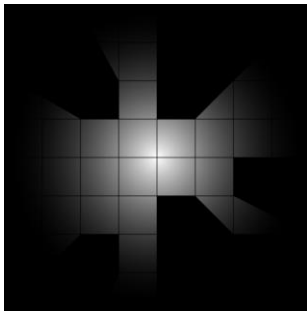
3.3 Schatten Zeichnen

Nun müssen die Schatten berechnet werden, die Winkel und Formen gefunden werden. Um besser zu verstehen wie das funktioniert sind die Objekte in den Bildern eingezeichnet, in wirklichkeit sind die aber nicht da.

[4 Bilder]



von links nach rechts: 1: Licht über Objekte gelegt, 2: Winkel eingezeichnet und Objekte eingegrenzt, 3: Schattenflächen eingegrenzt



Ergebnis: Licht mit Gezeichneten Schatten und darunter das Level

3.4 Lichter kombinieren

Dieser Resultierende Buffer wird danach auf einen weiteren Buffer gezeichnet auf dem schlussendlich alle Lichter gezeichnet werden. Dabei werden die Helligkeitswerte addiert, das heisst mit additiver Farbmischung gezeichnet. Somit wird es heller wenn zwei Lichter da sind und das eine Licht überschreibt nicht das Alte.

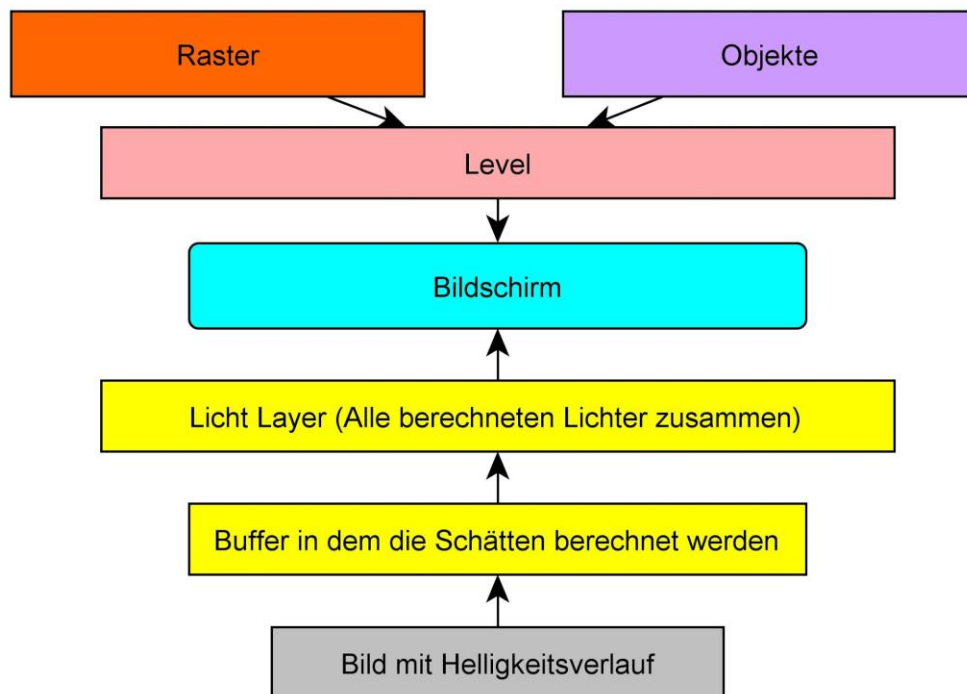
3.5 Lichter auf den Bildschirm malen

Schlussendlich wird dieser finale Buffer auf den Bildschirm gezeichnet, nachdem das Level auf dem Bildschirm ist. Das Zeichnen des finalen Buffers muss mit multiplikativer Farbmischung geschehen. Ist eine stelle im finalen Buffer hell, so sieht man alles auf dem Level. Ist sie hingegen dunkel sieht man weniger bis gar nichts.

Man kann das Level aber auch im Halbdunkeln ohne Lichter spielen. Dafür muss der finale Buffer nicht mit schwarz sondern mit irgend einer Graustufe gefüllt werden, bevor die Lichter darauf gezeichnet werden.

Um die Lightengine zu verwirklichen hatte ich nicht genügend Wissen über und Erfahrung mit OpenGL. Deshalb ging ich im Internet auf Programmierforen nach Lösungen suchen. Schlussendlich habe ich etwas gefunden und den Code auch integriert und angepasst [<http://www.blitzforum.de/forum/viewtopic.php?t=27274&highlight=fbo>].

[Bild lightengine]



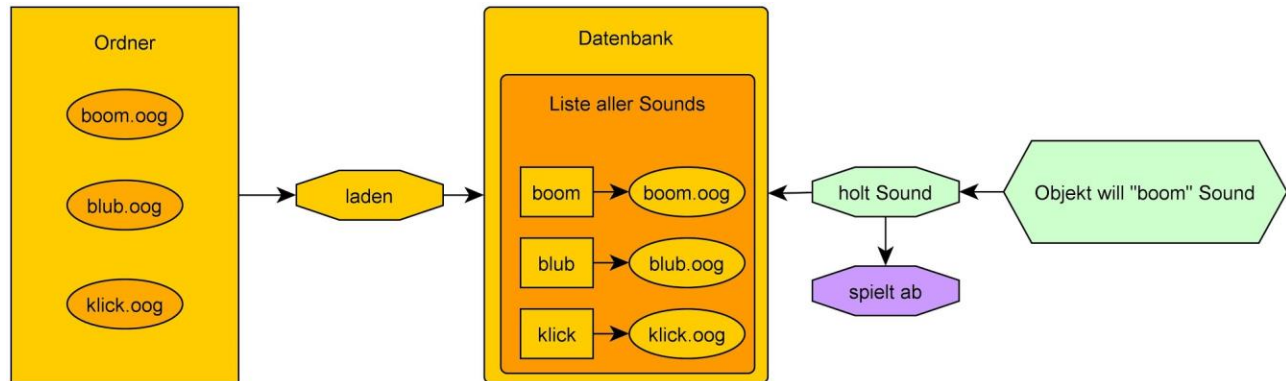
Grafik zur Berechnung was auf den Bildschirm gezeichnet wird, dh. Level und Lichter.

4. Soundengine

Die Soundengine ladt alle Lieder und Sounds warend dem Ladeprozess. Da gewisse Lieder sehr gross sind obwohl sie im OGG-Format sind, braucht dieser Ladevorgang relativ lange. Die Sounds und Lieder werden in einer Liste gespeichert in der sie mit ihrem Namen verknupft sind und so sehr schnell gesucht werden konnen. Ein Music-Block muss nur den Namen der Musik der Soundengine ubergeben und schon startet diese das Lied.

Zusatzlich kann die Soundengine Zweidimensionalen Sound abspielen. Das heisst wenn man links von der Quelle steht hort man den Sound von rechts und umgekehrt. Und je naher man ist, desto lauter tont es.

[Bild Soundengine]



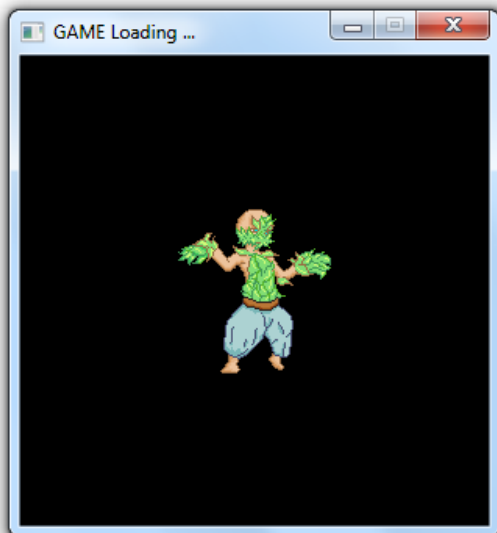
Struktur der Soundengine: Laden der Soundfiles, in einer Datenbank ablegen und mit Namen verknüpfen. Sobald der Sound gebraucht wird suchen und abspielen.

Beispiele aus dem Game

Das Game mit den Leveln ist noch nicht ganz fertig, teilweise fehlen Beschriftungen oder es existieren nur Platzhalter da uns die Zeit oder Ideen fehlten. Nichts desto trotz haben wir bereits einige Test-Levels.

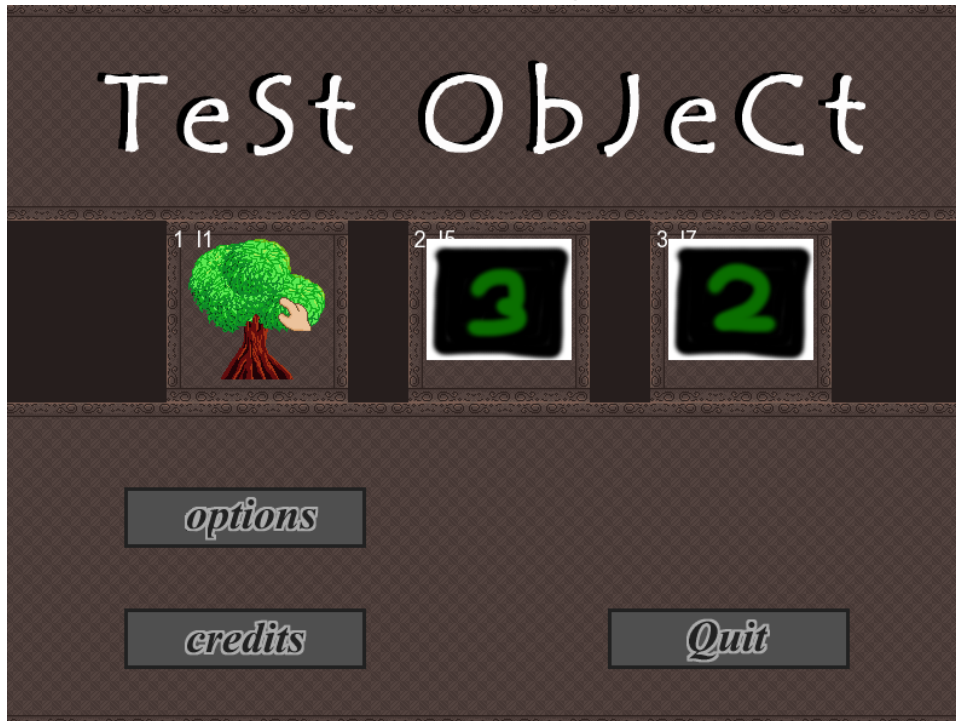
1. Loading-Screen

Da es doch einige Sekunden braucht bis alles Geladen ist, vor allem die Game-Musik braucht viel zeit, haben wir einen Loading-Screen eingefügt auf welchem man eine Animation sieht. Um das zu realisieren, also parallel eine Animation anzuzeigen und zu laden, musste ich das Programm in zwei Threads aufteilen, dh. es wird gemultitaskt.

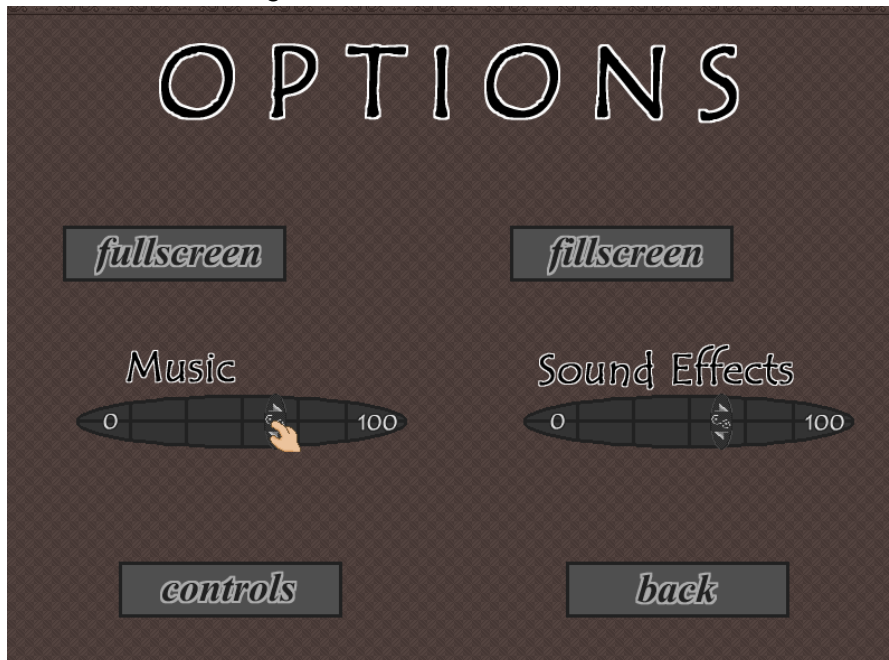


2. Menu

Das hier ist das Menu in welchem man beispielsweise ein Level auswählen kann.

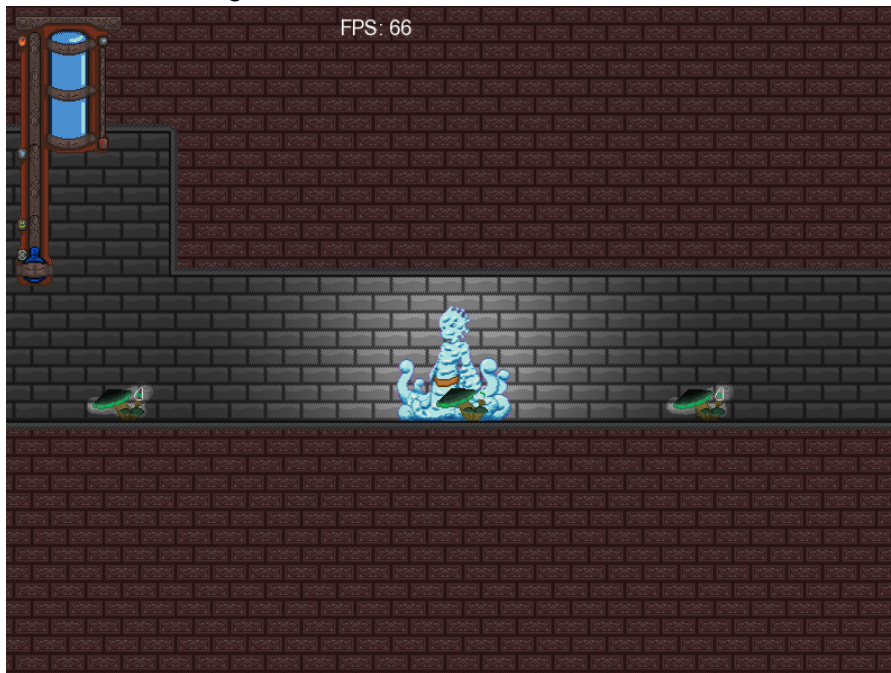


In den Optionen hat man diverse Möglichkeiten, man kann Änderungen an den Bildschirmeinstellungen, der Gamemusik- und Soundeffektlautstärke anpassen.



3. Lichteffekte

Dieses Thema wird genauer erklärt im Teil über die Lightengine. Diese Bilder sind Beispiele für deren Anwendung.



Hier sieht man den Spieler neben leuchtenden Pilzen. Es ist nicht perfekt, jedoch eine Demonstration was möglich ist.



Diese Bilder zeigen den Schattenwurf und auch die Möglichkeit farbige Lichter zu haben.

4. Schilder

Wenn der Spieler nahe dem Schild steht wird dessen Text angezeigt.

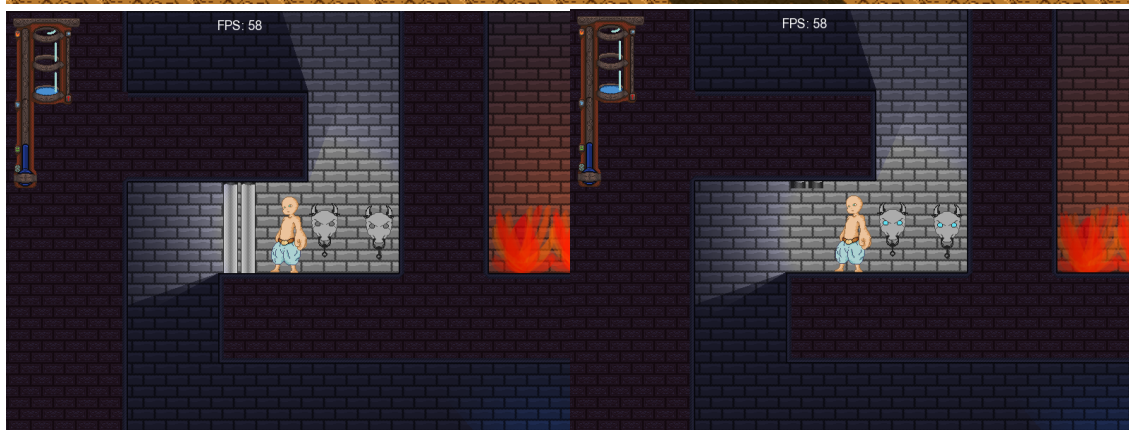


Screenshot mit Schild, es ist ausserdem ein gutes Beispiel für die Lichtengine, denn man bekommt den Eindruck, dass der Hintergrund leicht dreidimensional ist, was wahrscheinlich durch die unterschiedliche Beleuchtung und die Eigenschaften des Hintergrundbildes entsteht.

5. Türen und Schalter



Zwei Schalter, einer öffnet die Metalltüren, der andere schliesst sie



Dieser steinerne Kopf ist ein Türöffner, der bläuliche Augen bekommt wenn er aktiviert ist.

6. Spieler-Modi

Hier sind vorallem die Unterschiede zu Beachten. Der Feuer-Modus hat unten einen Feuer-Balken und sein Thermometer ist auf dem maximalen Wert. In diesem Modus schwebt man ein wenig. Der Wasser-Modus hat Wasser im Wassertank und seine Temperatur ist tief. Zudem kann man im Wasser-Modus nicht normal springen sondern muss sich aufladen und verliert beim springen Wasser.

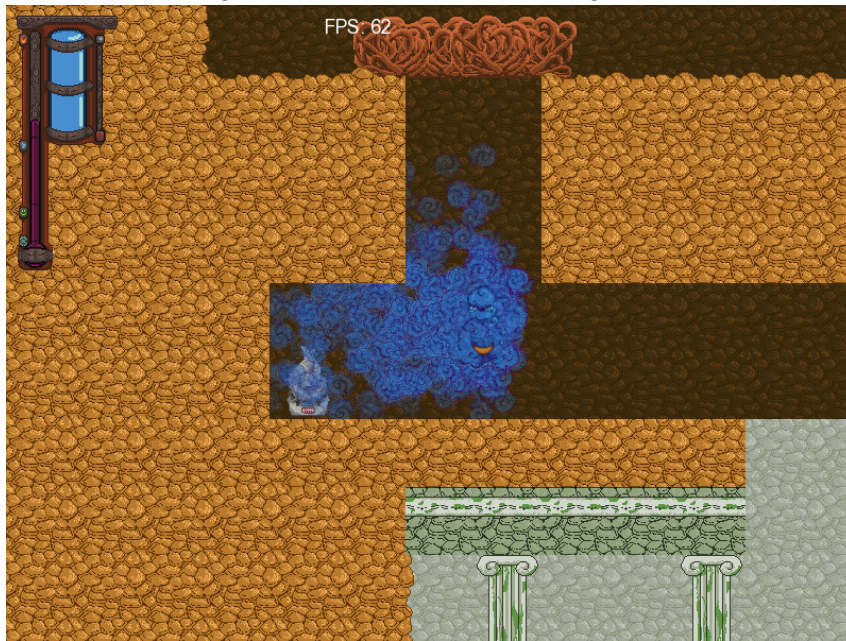


Im Pflanzen-Modus kann man zwar nicht springen, jedoch neben dem Pflanzenmonster durchgehen. Wenn man wieder in den Normal-Modus will kann man die Blätter einfach abwerfen.



Wasser und Feuer gibt Dampf. Im Dampf-Modus kann man nur nach links und rechts steuern

und schwebt langsam nach oben bis man abgekühlt ist.



7. Pflanzenmonster

Das Pflanzenmonster ist für den Spieler im Wasser-Modus eine Bedrohung. Wie man im Bild sieht saugt er einem das Wasser weg bis man verschwindet.



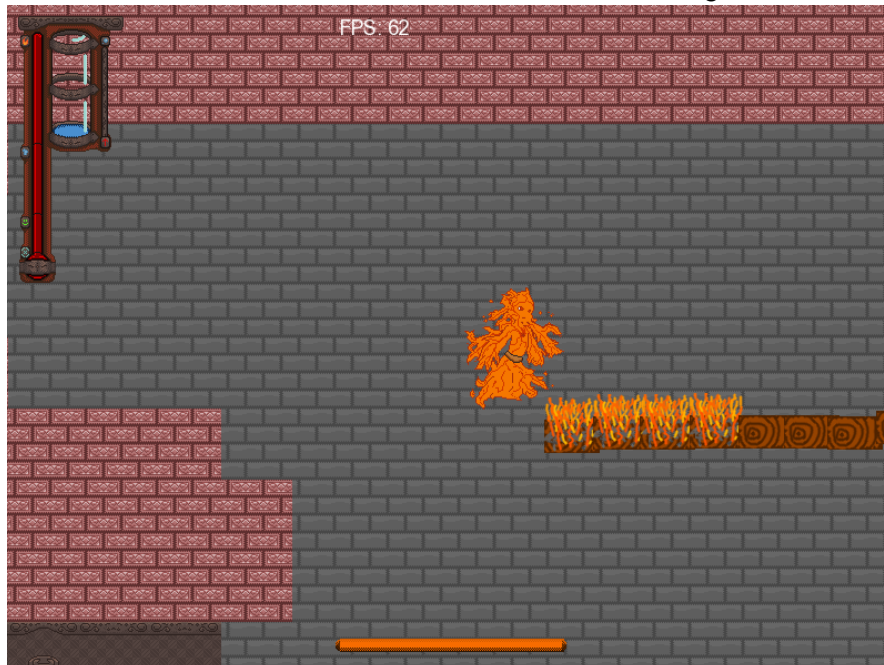
8. Der, der mit einem redet

Während dem Spiel findet man zb Personen und kann mit ihnen reden.



9. Holzblöcke

Wenn einem eine Holzbrücke aus Holzblöcken im Weg ist kann man sie einfach anzünden.



Einblick in den Editor

In diesem Teil möchte ich gerne einen kleinen Einblick in den Editor geben und wie man ihn

brauchen kann. Ich werde aber nicht alles im Detail erklären. Das Ziel ist es zu vermitteln wie ich arbeitete und nicht ein Tutorial zu schreiben.

1. Levelauswahl

Zuerst muss man ein leeres Level erstellen oder ein bereits bestehendes laden.

The image shows a software interface for level selection, divided into two main sections: 'CREATE' (top) and 'LOAD' (bottom). Both sections have a grey background and contain the following elements:

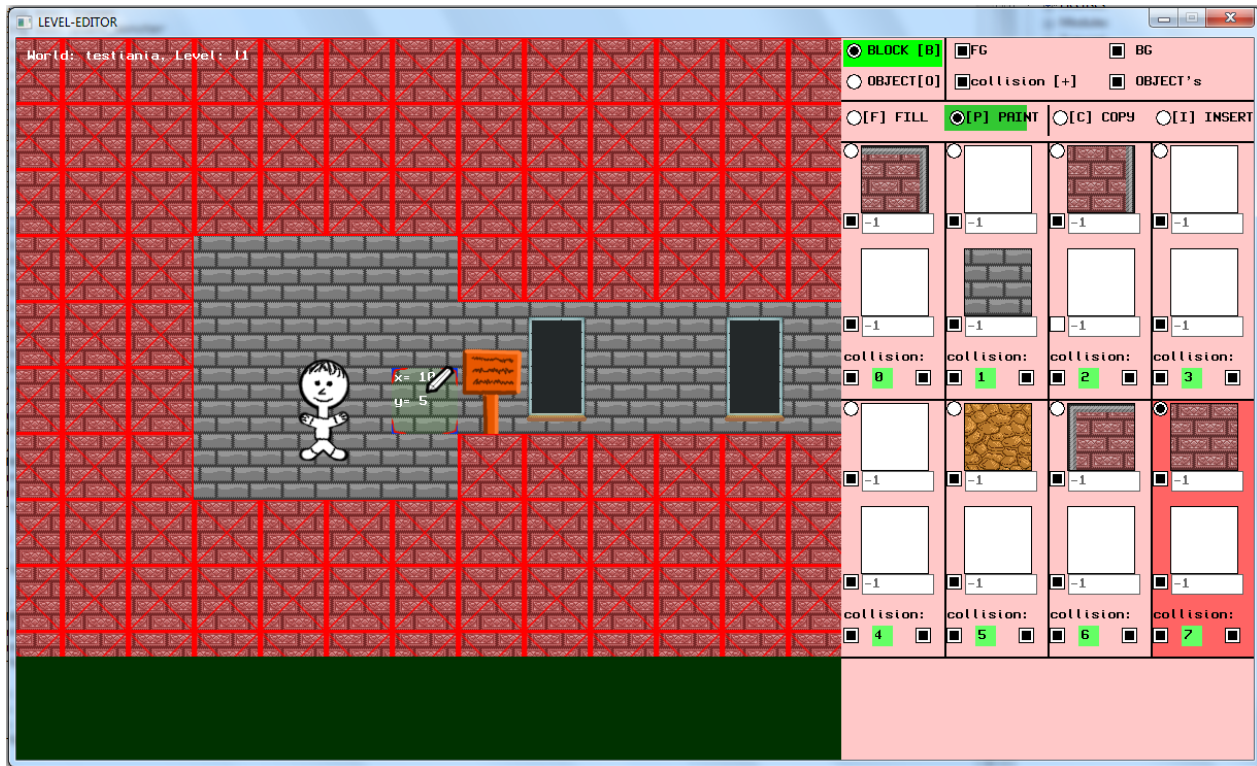
- WORLD:** A text input field containing 'testiania'.
- LEVEL:** A text input field containing 'l1'.
- Dimensions:** Two input fields labeled 'x' and 'y' (or 'x' and 'y' in the top section), both containing '100'.
- Buttons:** A green 'OK' button and a white button labeled 'CREATE' (top) or 'LOAD' (bottom).
- Status:** A green box on the right side of each section. The top section's box is labeled 'OVERWRITE?' in yellow text, and the bottom section's box is labeled 'EXISTS' in green text.

A mouse cursor is visible over the 'OK' button in the top section.

Oben kann man ein Level erstellen und die Abmessung angeben. Unten kann man ein bestehendes Laden. Rechts wird angezeigt ob das Level bereits besteht, damit man nicht ein bestehendes unwillentlich überschreibt oder nicht versucht ein nicht existierendes zu laden.

2. Raster

Jedes Feld des Raster kann bearbeitet werden. Dafür ist rechts die Palette, welche auch verändert werden kann. Man kann auch ganze Flächen füllen oder kopieren. Oben rechts kann man einstellen welche Layers man genau sehen will.



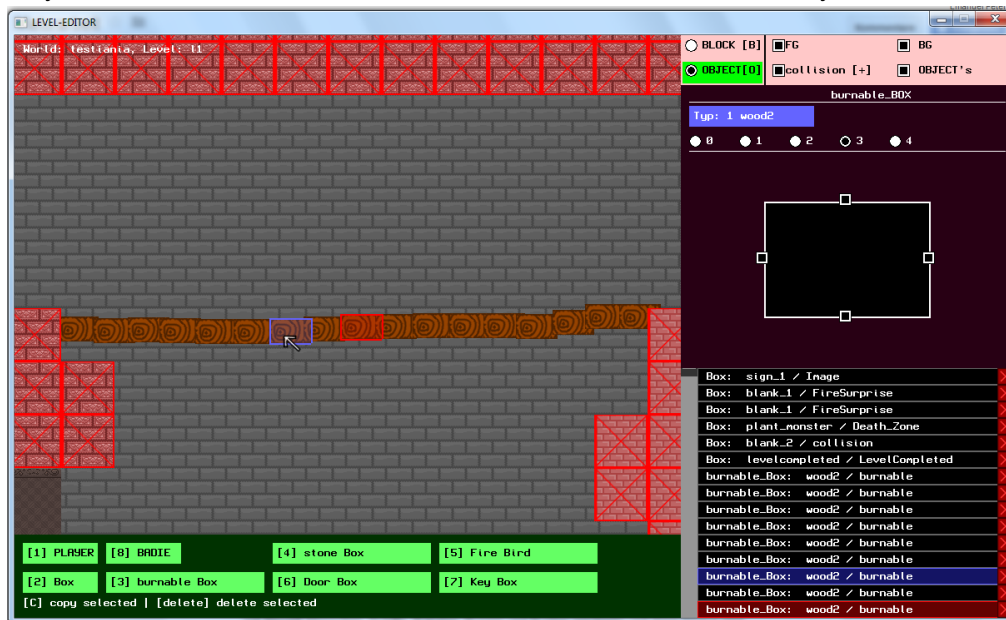
Wenn man auf [M] drückt kommt eine Übersicht über das Ganze level. Es wird jedoch nur das Raster dargestellt und keine Objekte.



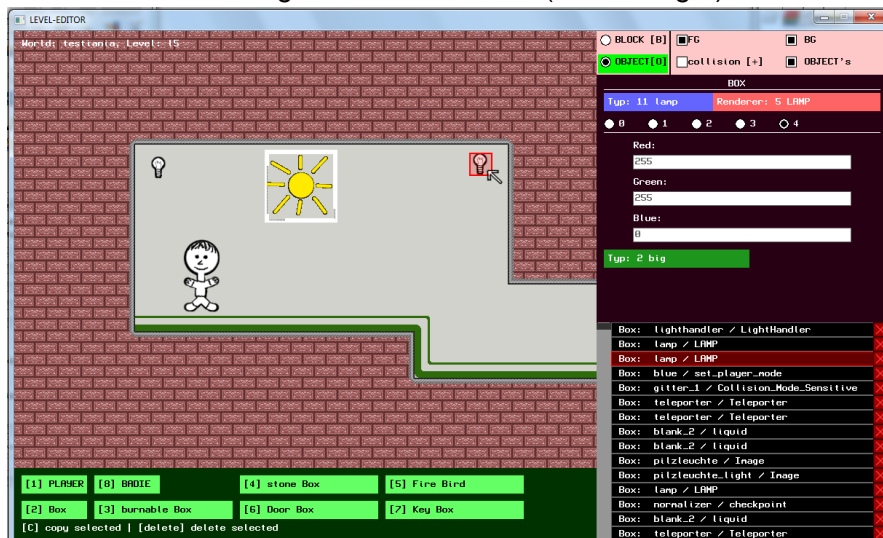
3. Objekte

Hier wird eine Brücke aus Holz-Boxen bearbeitet. Wenn man eine Box selektiert wird sie rechts zur Bearbeitung angezeigt. Man kann sie in einen Anderen Layer verlegen oder für gewisse

Seiten die Kollision ein oder aus stellen. Zudem hat es rechts unten eine Liste mit allen Objekten welche im Level sind. Unten links kann man neue Objekte erstellen.



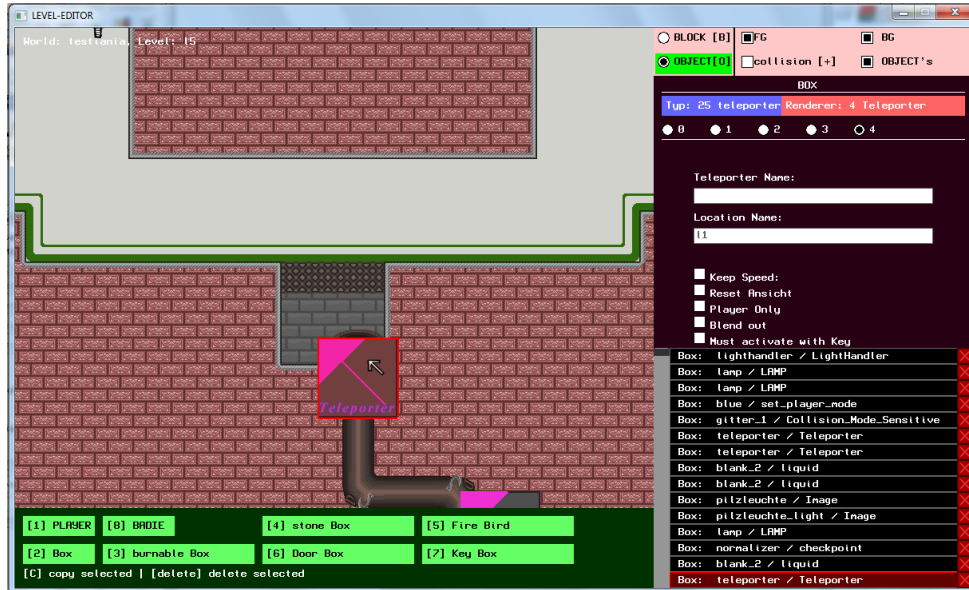
Man kann auch Lichter setzen und deren Farbe, Helligkeit und Bild festlegen. Mit der "Sonne" kann man das Hintergrundlicht einstellen (Ambientlight).



Teilweise Spielen das Raster und Objekte zusammen. Es kann sein, dass es einfacher ist das Bild vom Raster darzustellen lassen und die Funktion von einem Objekt nehmen. Diese durchsichtige Box hier lässt den Spieler je nach Modus durchfallen oder nicht. So kann man Gitter realisieren, durch welche man im Wasser-Modus nicht aber im Normal-Modus durchfällt.



Teleporter sieht man nur im Editor. Sind verlinkt durch Namen.



Rückblick

Bibliographie

<http://www.blitzforum.de/forum/viewtopic.php?t=27274&highlight=fbo>