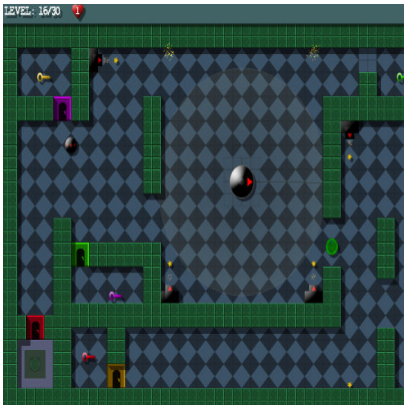


Wie man in 72h ein größeres Spiel macht!

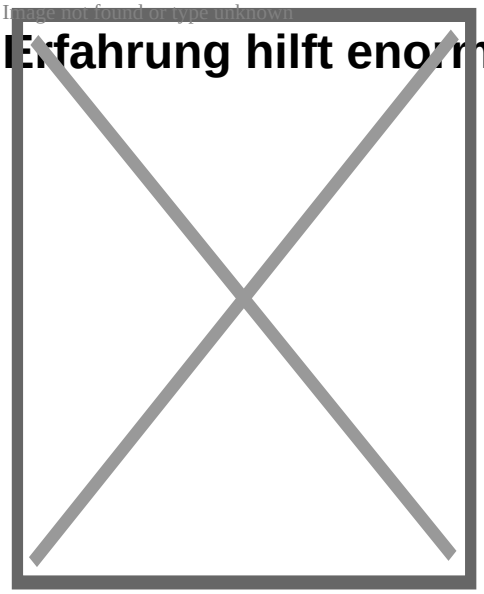
Description

JUST DO IT! ist ein kleines Geschicklichkeitsspiel. Es entstand innerhalb 72 Stunden für den Dr. Snails Game Jam und ist, zum Erstaunen mancher, sehr umfangreich. Wie funktioniert das? Wie kann man alleine binnen drei Tage ein Spiel mit 30 Level, zahlreichen Grafiken, Menüs, Optionen, Soundeffekten, Partikeln, Musik und vielen anderen Kleinigkeiten, die alle Zeit kosten, erstellen? Dieser Artikel soll die wichtigsten Tricks verraten.

Bevor es los geht: Wer sich einen Eindruck vom Spiel verschaffen möchte, kann hier die HTML5-Version spielen.

Image not found or type unknown

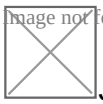
Erfahrung hilft enorm



Es ist vielleicht nicht unmöglich, aber extrem schwierig ohne Erfahrung überhaupt ein Spiel in 72h zu erstellen. Zum Glück ist es nicht mein erstes Spiel sondern ziemlich genau mein 15., wobei ich acht davon nahezu alleine erstellt habe. Die anderen Spiele entstanden in kleineren und zwei in etwas größeren Teams. Die über 25 Jahre gesammelten

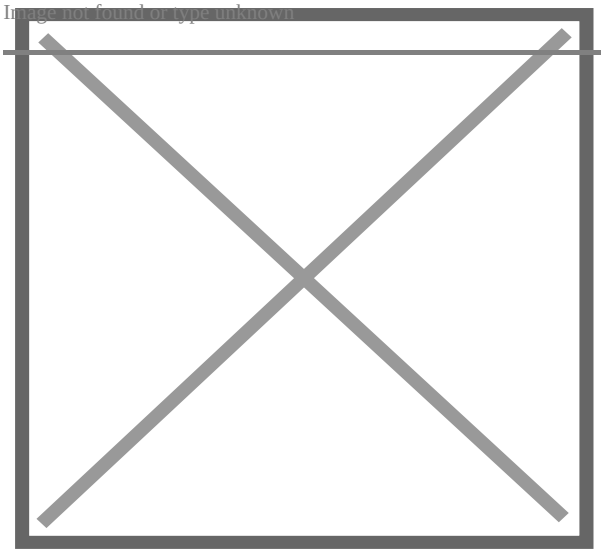
Erfahrungen helfen, den Aufwand einzuschätzen, aber das alleine reicht nicht. Auf meinen Festplatten befinden sich auch viele angefangene Projekte, Experimente und Übungen, bei denen ich bestimmte Spielelemente immer wieder neu entwickelte. So gab es in JDI! kaum etwas, was ich nicht schon irgendwann gemacht habe und so war es möglich, sehr viele Dinge nahezu blind runter zu tippen.

Ebenfalls hilfreich ist es, wenn man sich bereits gut mit seiner Entwicklungsumgebung auskennt. Wenn man erst lange suchen muss, wo und wie man bestimmte Dinge umsetzt, schafft man vieles einfach nicht. Neu bei dem Spiel war für mich, dass ich mich zum ersten Mal im GameMaker intensiver mit Tiles befasste. Beim Levelbau hätte es mir beinahe das Genick gebrochen, aber die Bedienung ist zum Glück derart intuitiv, dass ich die verlorene Zeit zum Ende hin aufholen konnte.



JDI! beinhaltet, je nachdem, wie man zählt, rund 10 Gegner-Typen. Dabei kommen zwei simple Tricks zur Anwendung: Für die Programmierung ist es einfach, wenn man nur zwei oder drei verschiedenartige Gegner hat und den Rest kopiert bzw. variiert. So gibt es einen Gegner, der horizontal hin und her fliegt. Dann den, im Prinzip gleichen, der das vertikal tut. Ein dritter fliegt bis zur Wand und biegt dann immer nach links ab, so dass er im Kreis fliegen kann (genau genommen ein Rechteck, aber egal). Ein Gegner, der von Links nach Rechts schießt, kann das auch umgekehrt und natürlich auch vertikal. So schafft man mit einfachen Mitteln Abwechslung. Der zweite Trick sind die Grafiken: Einfache Geometrie + Photoshop-Ebeneneffekte. Mit wenigen Mausklicks hat man Grafiken, die für ein 72h Spiel ansprechend genug aussehen und ihren Zweck mehr als erfüllen.

Image not found or type unknown



Das Geheimnis der Selbstschussanlagen, die dem Spieler folgen, ist auch schnell erzählt. Zunächst braucht es zwei Grafiken. Die eigentliche Objekt-Grafik und dazu einen zweiten Sprite mit dem Dreieck, welches dem Spieler folgt. Beide wurden in Photoshop erstellt und mit gleicher Größe exportiert, so dass sie perfekt passen. Im Code legt man ein paar Parameter fest wie den Radius, die Geschwindigkeit und dergleichen. Ich habe das später noch so verfeinert, dass der Startwert für die Drehung zufällig ist, damit nicht alle im Level synchron laufen (sah komisch aus). Die Abfragen, wo der Spieler ist bzw. wann er den entsprechenden Bereich überquert sind nur ein paar Zeilen Code. GameMaker hilft da mit Befehlen wie *collision_circle*, *point_direction*, *collision_line* und *lengthdir_x* bzw. *lengthdir_y*. Wenn sich der Spieler innerhalb des Kreises befindet, dreht sich das Objekt hin und wenn die Linie den Spieler berührt, schießt das Objekt. Das sind rund 60 Zeilen Code.

Die Darstellung ist dann denkbar einfach. Man zeichnet zunächst den Kreis, dann die Linie, welche abhängig von *image_angle* ist. Anschließend kommt das eigentliche Sprite und ganz oben das Dreieck, ebenfalls abhängig von *image_angle*. Die ganze Optik mit Farben, Outline etc. sind nur 15 Zeilen. Mit etwas Erfahrung ist das in Nullkommanichts umgesetzt.

Vorbereitung und Planung

Das Problem beim Jam war, dass das Thema erst kurz vor Beginn feststand. Die zur Wahl stehenden Themen waren teils so unterschiedlich, dass eine Vorarbeit nahezu unmöglich war. Das war die Absicht dahinter. Meine Vorbereitung bestand vor allem darin, mir Gedanken zu machen und mich auszuruhen. Das begann schon mit dem Schlafrhythmus. Ich stand erst wenige Stunden vor Beginn auf, war komplett ausgeruht und erstaunlich gut „im Saft“.

Zu ein paar Themen hatte ich bereits einige, teils verrückte Ideen. Als das Thema „Einer gegen alle“ feststand, musste ich mich für eine Idee entscheiden. Grob gesagt fiel die Entscheidung zwischen „mach was verrücktes, aber wenig davon“ und „mach was solides, aber dafür mehr Inhalt“. Es wurde Letzteres, da mir die Idee kam, gleich mehrere Kriterien bzw. zur Auswahl stehende Themen zu erschlagen. Als spezielle Herausforderung, sozusagen. Es sollte ein wenig wie alte DOS oder Amiga-Spiele wirken (Retro) und ich wollte einen Umfang von circa 30 Minuten Spielzeit erreichen.

Das Spielprinzip von JDO! war mir auch nicht ganz neu. Vor acht Jahren machte ich ein vergleichbares Spiel, damals aber viel umfangreicher mit gerenderten 3D Grafiken. Ich konnte also zumindest abschätzen, wie viel Arbeit der Levelbau in Anspruch nimmt.

Alte Batman-Veteranen kennen den Spruch: „Erfahrung lehrt langsam und auf Kosten vieler Fehler.“ Deshalb weiß ich mittlerweile, dass man mit den Dingen beginnen sollte, die am meisten Konzentration abverlangen. Klar, man kann auch mit Grafiken und Sounds beginnen und dann Level bauen, aber ich ziehe es vor, mit dem Code zu starten. Hier kann man mit Platzhalter-Grafiken arbeiten und wenn man das sauber löst, hat man für die kommenden Stunden ein relativ ruhiges Leben. Übermüdet komplexere Dinge zu programmieren führt nur selten zu einem beglückenden Ergebnis.

Auf der anderen Seite ist es für mich besser, Level zu bauen, wenn ich etwas müde bin. Da bin ich zwar nicht kreativer, aber da ich immer sofort teste, werden die Level um einiges leichter als im *fitten* Zustand. Den Trick habe ich bei CYPEST gelernt, weshalb ich jedes Level noch ausgiebig teste, wenn ich sehr müde bin.

Die Schlafeinteilung ist zwar eine wichtige Komponente, lässt sich bei mir aber nicht perfekt planen. In den 72h habe ich etwa sechs Stunden geschlafen. Nach dem Aufwachen habe ich mich mit Kaffee versorgt, ein paar Sachen getestet, dann einfache Dinge eingebaut und als ich fit war, ging es an die komplexen Arbeiten. Am Ende, vor allem am letzten Tag, war der Levelbau angesagt. Grundsätzlich war es auch hilfreich, die Arbeiten so einzuteilen, dass ich nicht zu lang die selben Aufgaben durchführte. Zwischen dem Code ging es an Soundeffekte, Musik, ein paar Grafiken und dann wieder, geistig ausgeruht, wieder an den Code. Auch hier hilft die Erfahrung, um sich die Kraft optimal einzuteilen.

Entwicklungsumgebung

Image not found or type unknown 

Man kann zu GameMaker Studio 2 stehen, wie man will, aber wenn man sich damit auskennt, ist es extrem effizient. Das liegt an der generellen Logik des GameMakers, aber auch an der eigenen Sprache GML. Schatten von Objekten sind jeweils zwei Zeilen. Wenn man mit Vererbungen arbeitet, spart man zusätzlich enorm viel Zeit. Die ganze Spielsteuerung sind relativ wenig Zeilen Code. Da ich vieles in Scripten und globale Variablen auslagerte, ist das extrem schnell eingebaut.

Außerdem gibt es Verfahren, die sich bei mir seit Jahren bewährt haben. Wie baue ich generell ein Spiel auf? Wo platziere ich welche Objekte und welche Werte frage ich wann ab? Wenn man das noch nie gemacht hat, hängt man sich schon an so simplen Dingen wie einer globalen Steuerung für Fullscreen/Fenstermodus auf. Hier spielen Erfahrungen und Entwicklungsumgebung perfekt zusammen. Seit Jahren verwende ich die gleichen Variablen und Abläufen, die sich im laufe der Zeit

nur etwas verbessert haben. Dazu kommen ein paar nützliche Skripte, die mich ebenfalls seit Jahren begleiten. Das ganze INI-System, Mehrsprachigkeit und andere Dinge begleiten mich bereits seit Jahren. Neu im Spiel war allerdings der Anspruch, das Menü mit Tastatur, Gamepad UND Maus zu bedienen. Das hatte ich vorher noch nie, weshalb ich auch auf kein bestehendes Menüsystem zurückgreifen konnte und alles neu schrieb. Aber auch das ist mit GameMaker relativ einfach umsetzbar, wenn man weiß, wie es geht.

Der Vorteil an der effizienten Vorgehensweise ist zusätzlich, dass man Bugs sehr schnell beheben kann. Erst kurz vor Release tauchten ein paar Fehler auf und beispielsweise der Wunsch eines Testers, Checkpoints einzubauen. Letzteres war in 20min umgesetzt, für die Grafik gingen weitere fünf Minuten drauf. Auch hier ist es gut, wenn man sich mit einem Programm wie Photoshop ausreichend auskennt.

Für die Sounds verwende ich ebenfalls Programme, die ich mittlerweile blind bedienen kann. Für Soundeffekte und Schnitt kommt natürlich Audacity zum Einsatz, Lieder werden ausschließlich in Renoise komponiert.

Recycling lebt vom Mittmachen

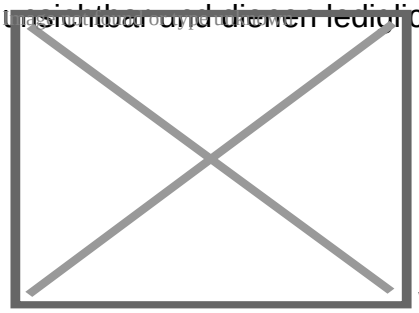
Das Recycling bei den Gegnern habe ich bereits beschrieben. Die Grafiken wurden zwar extra für das Spiel erstellt, dennoch gab es viele Stellen, an denen ich tricksen konnte. Wer Spiele von mir kennt, wird beispielsweise auch die Soundeffekte aus CYPEST kennen. Zumindest ein paar konnte ich übernehmen, da sie zum Retro Thema passen. Ein paar andere musste ich aber noch extra erstellen.

Das Tileset musste ich komplett neu erstellen, für die Buttons hatte ich bereits eine Vorlage aus anderen Projekten, beispielsweise Kopfnuss.

Wie bereits geschildert, lässt sich beim Code durch die konsequente Anwendung von Vererbungen viel einsparen. Ein zweiter Trick ist, alles, was man mehrfach braucht, in Skripte auszulagern.

Das offensichtlichste Element, bei dem Recycling zum Einsatz kam, waren die Level selbst. Zunächst baute ich mir eine Vorlage. Jedes Level hat eine Mindestgröße, ein paar Objekte die immer vorhanden sind und eine bestimmte Anzahl und Reihenfolge von Ebenen. JDI! hat acht Ebenen und eine Levelgröße von mindestens 1920×1080 Pixel. Immer vorhanden sind Wände, das Startfeld und das Ende-Feld. Der Spieler ist ein persistentes Objekt und wird vom Startfeld in Level 1 erzeugt, ab dann in jedem Level neu positioniert. Mit so einer Vorlage kann man schon ganz gut arbeiten. In vielen Fällen habe ich ein fertiges Level kopiert und angepasst, ggf. alle Inhalte gelöscht und die Größe erweitert. Als ersten Schritt setze ich Start und Ende auf ihre gewünschte Position und ziehe dann die Wände

ein. Zwischendurch habe ich an einer Stelle eine konkrete Idee und platziere da schon Gegner, Schlüssel und Türen. Anschließend wird getestet. Wenn der grobe Aufbau gut genug ist, wird er verfeinert, verbessert und am Ende werden die Tiles gemalt. Die Wand-Objekte sind schließlich unsichtbar und dienen lediglich der Kollisionsabfrage.



Was mich mit Abstand am meisten Zeit gekostet hat, waren die Schatten der Wände. Objekte wie Schlüssel, Gegner und Spieler erzeugen ihre Schatten in Echtzeit, aber die Schatten der Wände werden durch Tiles *erzeugt*. Das sieht im Spiel zwar ordentlich aus und kostet fast keine Ressourcen, doch im Levelbau ist es unglaublich aufwändig. Es gibt sieben verschiedene Schattengrafiken und pro Level kommen hunderte zum Einsatz. Im GameMaker kann man zwar über Lines größere Bereiche schnell zeichnen, aber wenn es zu kleinteilig wird, hilft das nicht. Es kostet nicht nur Zeit sondern lädt auch zu Fehlern ein. Immer wieder fand ich beim testen Stellen, an denen Schatten fehlten. Das nervt und gibt selbst nach Release kein gutes Gefühl.

Zur Musik: Am Anfang waren nur zwei Lieder geplant. Ein kurzes Lied im Menü und eines im Spiel. Vor einigen Jahren komponierte ich für einen Bekannten ein Dutzend Lieder für ein Spiel, welches nie erschien. Zwar davon nahm ich, komponierte sie um und verwendete andere Instrumente. Das ging relativ flott und klang meiner Meinung nach auch richtig gut. Mit dem wachsenden Inhalt musste ich aber feststellen, dass selbst der beste 2min-Loop nervt, wenn man ihn eine halbe Stunde hört. Also schaute ich in meinem Archiv der selbst komponierten Lieder nach und fand ein paar, die ich dazu mischen konnte. Es ist zwar selbst gemacht, aber, abgesehen von ein paar Anpassungen und dem eigentlichen Mix, nicht in den 72h. Übrigens ist das auch der Grund, warum das Spiel rund 17MB hat. Es stecken mehr als 10min Musik darin.

Wille schafft alles

Mein persönliches Ziel mit dem Spiel war es, ein Projekt abzuliefern, mit dem ich mich nicht blamiere. Außerdem wollte ich sehen, was ich in 72h noch auf die Beine stellen kann, schließlich wird man ja nicht jünger.

Die erste Hälfte ging relativ flott, aber der Levelbau in der zweiten Hälfte raubte mir ziemlich viele Nerven. Hier half eine immer lautere, schnellere Musik, Koffein und der unbedingte Wille, es schaffen zu wollen. Perfekt lief das natürlich nicht ab, beispielsweise musste ich zwischendurch für ein paar Stunden raus um einzukaufen, aber man muss das positiv sehen. Bewegung kurbelt den Kreislauf an und man kann sich Gedanken über Leveldesign, Schwierigkeitsgrad, Lernkurven etc. machen. Ich denke, dass das am Ende auch das Entscheidende war: positive Gedanken. Statt vorab Ausreden für das eigene Scheitern (in Bezug auf verfehlte Ziele) zu suchen, muss man von sich und der Arbeit überzeugt sein.

Date Created
8. November 2018

Author
sven