GEORG-SIMON-OHM
HOCHSCHULE NÜRNBERG

Fakultät Elektrotechnik Feinwerktechnik Informationstechnik efi

Projekt: ReMaster Blaster

Prüfungsstudienarbeit von

Sergej Bjakow

Matr. - Nr.: 211 06 31

Bachelor Media Engineering 6. Semester

Thema

Reverse Engineering am Amiga Klassiker "Master Blaster"

Sommersemester 2012

Bestätigung gemäß § 35 (7) RaPO

Sergej Bjakow

Ich bestätige, dass Ich die Prüfungsstudienarbeit mit dem Titel:

Reverse Engineering am Amiga Klassiker "Master Blaster"

selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benützt, sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Datum: 03.08.2012

Unterschrift:_____

Abstract

Gemäß Paragraph 21, Absatz Eins der Rahmenprüfungsordnung, kurz RaPo, sind Prüfungsstudienarbeiten und Prüfungsleistungen überwiegend zeichnerischem, gestalterischem oder sonstigem komplexen Inhalt und offenem Lösungsweg zum Nachweis kreativer Fähigkeiten, die sich wegen der umfassenden Aufgabenstellung und der Art der Aufführung in der Regel über einen längeren Zeitraum erstrecken.
Die einzelnen Prüfungsstudienarbeiten des Projektteams, bestehend aus Sebastian Adam, Sergej Bjakow, Michael Kao, Pavlina Pavlova und Maximilian Seyfert, welches sich mit der Realisierung eines Remakes des AMIGA Klassikers MasterBlaster mit neuen Webtechnologien (HTML5, CSS, JavaScript) befasst hat, sollen einen individuellen Einblick auf die jeweiligen Anforderung, die es mit Lösungsstrategien zu bewerkstelligen galt, gewährleisten.
Diese, meine eigene Studienprüfungsarbeit, will Auskunft über die Bedeutung der Rolle meiner Person in der gemeinsamen Teamarbeit verdeutlichen.

Im folgenden Bericht werde ich auf meinen Teil der Projektstudienarbeit "ReMaster Blaster" eingehen, dem Reverse Engineering am Amiga Klassiker "Master Blaster". Hierbei erläutere ich die genauen Workflows, die nötig waren um sämtliche visuelle Spielinhalte aus dem Originalspiel filtern zu können, die Bearbeitung der Audio-Dateien sowie die Schwierigkeiten, die besonders bei der Automatisierung Selbiger in JavaScript aufgetreten sind. Darüberhinaus gehe ich auf JavaScript-spezifische Funktionalitäten innerhalb der Spielelogik, wie die Alarmfunktion ein. Dieser Bericht soll vor allem verdeutlichen, dass zur Realisierung eines Reverse Engineering Projekts eine intensive Auseinandersetzung mit dem Grundspiel essentiell ist.

Projekt-Roadmap

Tätigkeit	Dokument	Beteiligte
Erstellung des Lastenhefts	02_Dokumente/04_Lasten- und Pflichtenheft/Lastenheft_v1_1	Sebastian Adam
Erstellung des Pflichtenhefts	02_Dokumente/04_Lasten- und Pflichtenheft/Pflichtenheft_v1_1	Sebastian Adam
Erstellung des Zeitplans	02_Dokumente/03_Projektplan/ Remasterblaster.pdf	Sebastian Adam
Erstellung des Protokolls	02_Dokumente/01_Protokolle/ ...Protokoll_01.pdf	Sebastian Adam
Erstellung des Protokolls	02_Dokumente/01_Protokolle/ ...Protokoll_02.pdf	Sebastian Adam
Erstellung des Protokolls	02_Dokumente/01_Protokolle/ ...Protokoll_03.pdf	Sergej Bjakow
Erstellung des Protokolls	02_Dokumente/01_Protokolle/ ...Protokoll_04.pdf	Sebastian Adam
Erstellung des Protokolls	02_Dokumente/01_Protokolle/ ...Protokoll_05.pdf	Maximilian Seyfert
Erstellung des Protokolls	02_Dokumente/01_Protokolle/ ...Protokoll_06.pdf	Michael Kao
Erstellung des Protokolls	02_Dokumente/01_Protokolle/ ...Protokoll_07.pdf	Sebastian Adam
Erstellung des Protokolls	02_Dokumente/01_Protokolle/ ...Protokoll_08.pdf	Sebastian Adam
Erstellung des Protokolls	02_Dokumente/01_Protokolle/ ...Protokoll_09.pdf	Maximilian Seyfert
Erstellung des Protokolls	02_Dokumente/01_Protokolle/ ...Protokoll_10.pdf	Sebastian Adam
Erstellung des Protokolls	02_Dokumente/01_Protokolle/ ...Protokoll_11.pdf	Sergej Bjakow
Erstellung des Protokolls	02_Dokumente/01_Protokolle/ ...Protokoll_12.pdf	Sebastian Adam
Erstellung des Protokolls	02_Dokumente/01_Protokolle/ ...Protokoll_13.pdf	Sebastian Adam
Erstellung der Testbench mit der Sprite – Engine	04_Quellcode/TestBench/	Pavlina Pavlova
Erstellung der ersten Testumgebung	04_Quellcode/TestEnvironment/	Michael Kao
Programmierung der Spielelogik	04_Quellcode/finalGame/game.js	Michael Kao
Programmierung des Graphical User Interface	04_Quellcode/finalGame/gui.js	Pavlina Pavlova
Erstellung der Sprites – Quelldatei	04_Quellcode/finalGame/ sprite_players.psd	Michael Kao Sergej Bjakow
Erstellung eines Structure Trees	04_Quellcode/finalGame/ Structure_Tree.txt	Michael Kao

Modifikation der Sounds zu mp3 Dateien	04_Quellcode/finalGame/sounds/	Sergej Bjakow
Capturing der einzelnen Elemente (Sprites)	04_Quellcode/finalGame/sprites.psd	Sergej Bjakow
Vorbereiten der einzelnen Szenenelemente	04_Quellcode/finalGame/img/	Pavlina Pavlova
Erstellung der Zwischenpräsentation	02_Dokumente/05_Präsentationen/ 2012_05_09_ReMasterBlaster.pdf	Sebastian Adam
Erstellung der Abschlusspräsentation	02_Dokumente/05_Präsentationen/ 2012_07_20_ReMasterBlaster.pdf	Sebastian Adam Michael Kao Pavlina Pavlova
Erstellung eines Plakats für die Abschluss-präsentation	05_Materialien/Extras_Plakat.pdf	Sebastian Adam

Inhaltsverzeichnis

1. Reverse Engineering

Dreh- und Angelpunkt der Projektarbeit war es die Entwicklungsprozedur „Reverse Engineering" kennen zu lernen. Reverse Engineering ist eine Methode der Softwareentwicklung bei der ein vorhandenes, fertig gestelltes Softwareprodukt her genommen und mittels Analyse und diverser Entwicklungstools in seine Grundbestandteile, sowohl visuelle, auditive als auch logische Inhalte zerlegt wird, um es nach zu konstruieren. Sinn dieses Vorganges ist es die separierten Konstruktionselemente so zusammen zu bringen, dass das Originalprodukt weitestgehend exakt nachgebildet werden kann, jedoch unter der Grundvoraussetzung auf einer neuen Plattform zu laufen unter Berücksichtigung einer Architektur, die auch potentielle Weiterentwicklungen ermöglicht. Gegenstand unseres reversen Entwicklungsprozesses war der Computerspiel Klassiker „Master Blaster" von Alexander Ivanov aus dem Jahre 1994 für die Amiga. Bei dem Spiel handelt es sich um ein Top-Down Geschicklichkeitsspiel bei dem es gilt als eine von maximal 5 Spielfiguren auf einem überschaubaren Spielfeld Bomben zu legen um damit Gegenspieler auszuschalten. Dabei stehen zahlreiche Blöcke im Weg, die durch Detonation gesprengt werden und hilfreiche Items beherbergen können, wie ein größerer Sprengradius oder temporäre Unbesiegbarkeit. Das Prinzip lässt sich am besten mit den „Bomberman"-Spielen von Hudson vergleichen.
Da das Spiel auf einer veralteten Plattform läuft, deren Nutzung und Anschaffung eher unkomfortabel ist und zu dem auch der Markt derzeit boomt was die Portierung alter Spieleklassiker auf aktuellen Plattformen angeht, seien es die internen Emulatoren aktueller Spielekonsolen (Virtual Console, Xbox Life) oder auch browserbasierte Varianten, bot sich die Rehabilitierung dieses Spieles besonders an. Meine Aufgabe in diesem Projekt war es die Kollegen mit Spielematerial aus dem Originalspiel, sprich Grafiken und Sounds, aber auch mit den nötigen Infos bezüglich der Spielelogik zu versorgen. Der Reverse Engineering Prozess umfasste die Extraktion sämtlicher Grafiken im Spiel, auch

„Sprites“ genannt, Schnitt und Bearbeitung aller Sound-Dateien sowie der Analyse von Animationstiming, Auftrittswahrscheinlichkeiten erscheinender Items und Gameplay-spezifischen Eventualitäten.

2. Spielanalyse

Wichtig beim Reverse Engineering Prozess ist es noch bevor irgend etwas aus dem Spiel herausgenommen wird erst einmal Selbiges zu analysieren, also genau festzustellen was eigentlich genau aus dem Spiel heraus genommen werden muss. Um eine bequeme und unkomplizierte Filterung des Spieles möglich zu machen, wurde es nicht auf der original Plattform abgespielt sondern auf einem Emulator, dem „WinUAE“, der über das Abspielen der jeweiligen Spiele hinaus auch über unzählige Zusatzfunktionen verfügt, wie Screen Capturing und Audio Recording, was sich für den weiteren Verlauf der Projektarbeit als praktisch erwies.
Zunächst habe ich mir einen Überblick verschafft über den Gameplay Fluss um ersehen zu können wann welche grafischen Inhalte auf dem Screen auftauchen und extrahiert werden können. So stellte sich zum Beispiel heraus, dass ein bestimmtes Item, nämlich das „Cash“-Goodie gar nicht erst in einer Spielerunde erscheint, wenn in den Einstellungen der Shop abgeschaltet wurde. Auch die Spielrundendauer hing von einem entscheidenden Faktoren ab, nämlich ob ein Timeout alle übrigen Spieler „entfernt“ oder sie durch das im Menü einstellbare „Shrinking“ von Blöcken zerquetscht werden, die gegen Ende einer Runde kontinuierlich das Spielfeld einengen. Außerdem musste das Spiel soweit studiert werden um optimale Grundvoraussetzungen für einfachere und bequemere Sprite Extraktion zu gewährleisten, gerade bei schnell ablaufenden Animationen.
Für Die Filterung von Grafiken und auch weiterer Untersuchungen wurde der emulatorinterne Screen Recorder verwendet, der den Spielebildschirm unkomprimiert in Originalgröße als abspielbares Video speichern konnte.
Als Dateiformat für sämtliche Sprites fiel die Wahl auf PNG, da diese typisch für

eine solche 8-Bit Grafik äußerst simpel sind, nur einen Farbraum von 256 Farben aufweisen und damit auch keine Farbverläufe vorkommen. Daher war es auch wichtig ein kompressionsloses Dateiformat zu wählen, dass die klaren Farbstrukturen nicht durch Anti-Aliasing Effekte verfremdet.

Da das Spiel grafisch wenig aufwendig war, konnte bei einigen Darstellungen der Aufwand der Extraktion umgangen werden, wie zum Beispiel die grüne Bodenfläche, auf der sich die Spielfiguren bewegen. Diese konnte durch einen simplen Cascading Style Sheet reproduziert werden, was auch die wesentlich performantere Lösung ist, als ein großes PNG-File zu verwenden. Um den exakten Farb-Ton zu treffen wurde lediglich der Bildschirm extrahiert, und mit der Pipette in Photoshop der exakte RGB-Wert ermittelt.

Neben der groben Spielanalyse für die Spritefilterung war auch eine genauere Untersuchung des Spiels nötig wie die Auftrittswahrscheinlichkeiten von erscheinenden Items, Steuerungsmechanismen oder gewisser Sonderfälle innerhalb der Spielelogik um die Spielbalance authentisch replizieren zu können. Eine der ersten Analysen für die Minimalanforderungen war das feststellen der Geschwindigkeit der Spielfiguren. Um diese zu messen, wurde in einer Spielpartie vom Screen Recorder ein kurzer Ausschnitt aufgenommen, wie eine der Figuren von links nach rechts läuft. Dieses Video wurde anschließend in der Videobearbeitungssoftware „Virtualdub" abgespielt, um aus der Differenz von verschiedenen Framepositionen die Zeit herauszurechnen, die die Spielfigur brauchte an einer relativen Bezugsgröße, in diesem Falle ein Spielblock vorbei zu laufen. Entsprechend verlief die Analyse der Spielfigur, nachdem sie das „Speed Up"-Powerup eingesammelt hatte, was die Laufgeschwindigkeit erhöht. Die Anfangsgeschwindigkeit einer jeden Spielfigur betrug 0,36 Sekunden pro Block und einer Kürzung um 0,12 Sekunden bei jedem Einsammeln des „Speed Up"-Powerups. Zusätzlich musste das Intervall nachgemessen werden, wie oft sich ein Sprite innerhalb eines Animationszyklus in Folge wiederholt, ehe ein neuer Sprite geladen wird. Im

Falle der Spielfiguren ist es beispielsweise so, dass die Laufanimation aus der mehrfachen Wiederholung einer einzelnen Pose besteht, ehe die nächste geladen wird. Die Analyse ergab, dass jeder Sprite exakt 6 Frames lang bestehen bleibt, was einer Zeit von 0,12 Sekunden entspricht, ehe der nächste geladen wird. Proportional dazu reduzierte sich pro „Speed Up"-Powerup die Anzahl der Frames um 1, was auch hektischere Bewegungen simuliert.

Eine weitere wichtige Messung war die Auswertung der Auftrittswahrscheinlichkeiten der Goodies, die sich in sprengbaren Blöcken befinden. Es gibt 12 sammelbare Items sowie ein Totenkopf-Feld und der gesonderte Bröckel-Status eines Blockes. Die Wahrscheinlichkeit der Auftrittshäufigkeiten dieser Variablen wurde anhand von mehreren Spieldurchgängen ausgerechnet. In 10 Spielen wurden so viele Blöcke, wie es in der limitierten Zeit möglich war gesprengt und das Erscheinen jeder Entität notiert. Da ich die Auswertung alleine vorgenommen habe, war es nicht möglich alle 122 Blöcke innerhalb der limitierten Zeit zu sprengen, somit bezogen sich meine notierten Häufigkeitswerte auf die Anzahl der tatsächlich gesprengten Blöcke, welche im Durchschnitt etwa 91 betrug. Es stellte sich heraus, dass die häufigste Konsequenz aus einem gesprengten Block der „Bröckel"-Status ist, der darauf hinweist, dass der Block noch zwei weitere male gesprengt werden muss, ehe er vollständig beseitigt ist. Aus der Anzahl der erschienenen Items wurde der Durchschnitt berechnet und dieser in Relation zu den gesprengten Blöcken gebracht, wobei sich dabei herausstellte, dass die Summe aller Wahrscheinlichkeiten knapp über 50% lag, was bedeutet, dass die andere Hälfte leere Blöcke waren.

3. Sprite Extraktion

Da das Originalspiel nur als Emulator Version vorlag, konnte auf die einzelnen Konstruktionselemente nicht direkt zugegriffen werden und so mussten sämtliche grafischen Inhalte über den indirekten Weg des Screen Capturings

und Cropping aus dem Spiel exkludiert werden. Hierfür wurde wie bereits bei der Geschwindigkeitsanalyse der emulatorinterne Screen Recorder verwendet, wobei hier nun ein unkomprimiertes Datenformat nötig war, sprich eine Wiedergabe in vollen Einzelbildern, um die graphischen Inhalte des Spieles verlustfrei und in Originalgröße herausfiltern zu können. Zwar hätten statische Sprites wie etwa die Items oder die Blöcke auch durch reine Screen Snapshots gefiltert werden können, doch da der Großteil der zu filternden Sprites nur in Animationen vorkamen, war die ökonomischste Lösung in einer Spielrunde möglichst viele sichtbare grafische Elemente offenzulegen, um diese in einem Video abrufbar zu machen.
Das Video wird nun mit einem herkömmlichen Media Player (z.b. VLC) abgespielt und die jeweiligen Standbilder durch die playerinterne Snapshop Funktion als eigenständige PNG-Files abgespeichert. Grafiken, wie eben die Item-Miniaturen lassen sich dann in Photoshop durch simples Zuschneiden zu einzelnen Bilddateien reduzieren. Grafikelemente, wie etwa der Totenkopf die nicht die komplette Fläche von 32x32 Pixel bedecken, bedürfen etwas Freistellarbeit und einem daraus resultierenden transparenten Hintergrund. Nach ein paar Spielrunden schon ließen sich sämtliche Items herausfiltern und übrig blieben die wesentlich aufwendiger zu filternden Sprites, die in Animationen verbaut sind, wie Spielfiguren, deren „Sterbe"-Animationen und Explosionen, die nur mit stark verringerter Abspielgeschwindigkeit der Videowiedergabe vernünftig zu identifizieren und separieren waren.
Hier habe ich nun von jedem der 5 Spielfiguren Videos aufgenommen, wie sie auf einer freien Fläche in alle Richtungen laufen um sämtliche Spriteposen zu erhalten. Die freie Fläche diente dazu das Freistellen der Figuren etwas unkomplizierter zu gestalten, da die Spielfiguren sich sonst mit Blöcken oder Items überlappen und eines zusätzlichen Radieraufwands bedürfen.
Bei der Extraktion der ersten Spielfigur fiel auf, dass einige Posen, wie die Seitenansichten identisch, nur lediglich gespiegelt sind. Daher bin ich bei den späteren Spielfiguren so vorgegangen bei den Videoscreenshots lediglich nur eine Seite auf zu nehmen und sie in Photoshop einfach zu duplizieren und

horizontal zu spiegeln. Was jedoch für zusätzlichen Aufwand sorgte, war die Tatsache, dass die Spielfiguren alle exakt 2 Pixel breiter waren als die statischen Bildinhalte. Dies hatte zur Folge, dass eine Spielfigur nicht zwischen zwei Spielblöcke gepasst hätte. Im Originalspiel konnten die überstehenden Konturen die Spielblöcke überlappen, doch im Rahmen der Projektarbeit lies sich dieses kleine Detail nicht realisieren und so wurde entschieden, jede Sprite Pose horizontal auf 32 Pixel runter zu stauchen, was zwar bei genauerem Blick in kleinen Anti-Aliasing Effekten resultierte, diese jedoch während des Spiels zu keiner Zeit auffallen sollten.

Bei der Extraktion der visuellen Spieleinhalte ist vor allem die konsequente Einhaltung eines Größenformates wichtig, was später beim Abruf der Spritedateien mittels der Sprite-Engine „Crafty" zur Geltung kommt. Da bis auf die Spielfiguren alle graphischen Elemente des Originalspieles 32x32 Pixel groß sind, war es wichtig Selbige auch beim herausfiltern aus dem Emulator in dieser Größe zu erhalten. Aus Flexibilitätsgründen wurden die Item-Miniaturen in separaten kleinen PNG-Files gespeichert, doch für die Endverwendung wurden diese in einem umfassenden PNG-File wieder zusammengetragen. Statt im Quellcode unzählige PNG-Dateien Aufzurufen wurde stattdessen bei der Definition von Figuren und Items ein Teilbereich der Bildfläche definiert und dem Objekt zur visuellen Darstellung zugewiesen, wobei Item- und Spielfiguren Sprites wegen unterschiedlichen Größenverhältnissen in jeweils separaten Files gelagert wurden.
Für eine problemlose und einfache Definition des gewünschten Spritebereiches wurden die Bilddateien konsequent und gleichmäßig nahtlos nebeneinander gesetzt. Hierfür bietet Photoshop ein individuell verstellbares Raster, das sich sehr gut zur Überprüfung eignet. Um einen Sprite auszuwählen, wird bei der Definition der Entität ein Bereich, relativ zum Ursprung des Bildes, welcher sich im oberen linken Eck einer jeden Bilddatei befindet, in Pixelgröße festgelegt. Ausgehend von diesem Ursprung können vier Pixelkoordinaten festgelegt werden, sprich zwei x/y-Paare, die den darzustellenden Bereich eingrenzen.

Vom ersten Koordinatenpaar ausgehend wird quasi ein unsichtbares Rechteck gezogen, das bis zum zweiten Koordinatenpaar reicht. In verwendeten Sprite-Engine Crafty sieht solch eine Definition wie folgt aus:

```
Crafty.sprite("sprite_players.png", {
            POLICEMAN: [0, 0, 32, 44]});
```

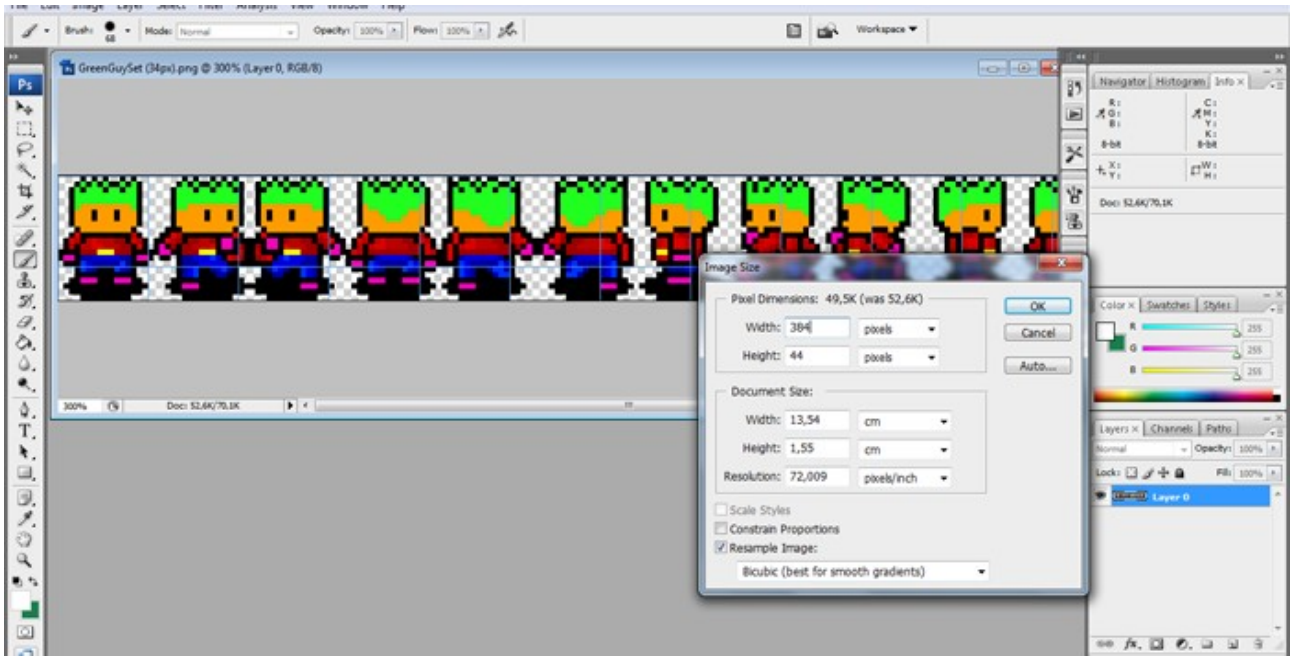
Hier wird aus der Bilddatei „sprite_players" für die Spielfigur „Policeman" ein 32x44 Pixel großer Bereich festgelegt, der im Ursprung (obere linke Ecke) beginnt. Im Falle der Items, die ausschließlich in quadratischen Sprites gefasst sind, bietet Crafty auch eine etwas einfachere Form der Definition:

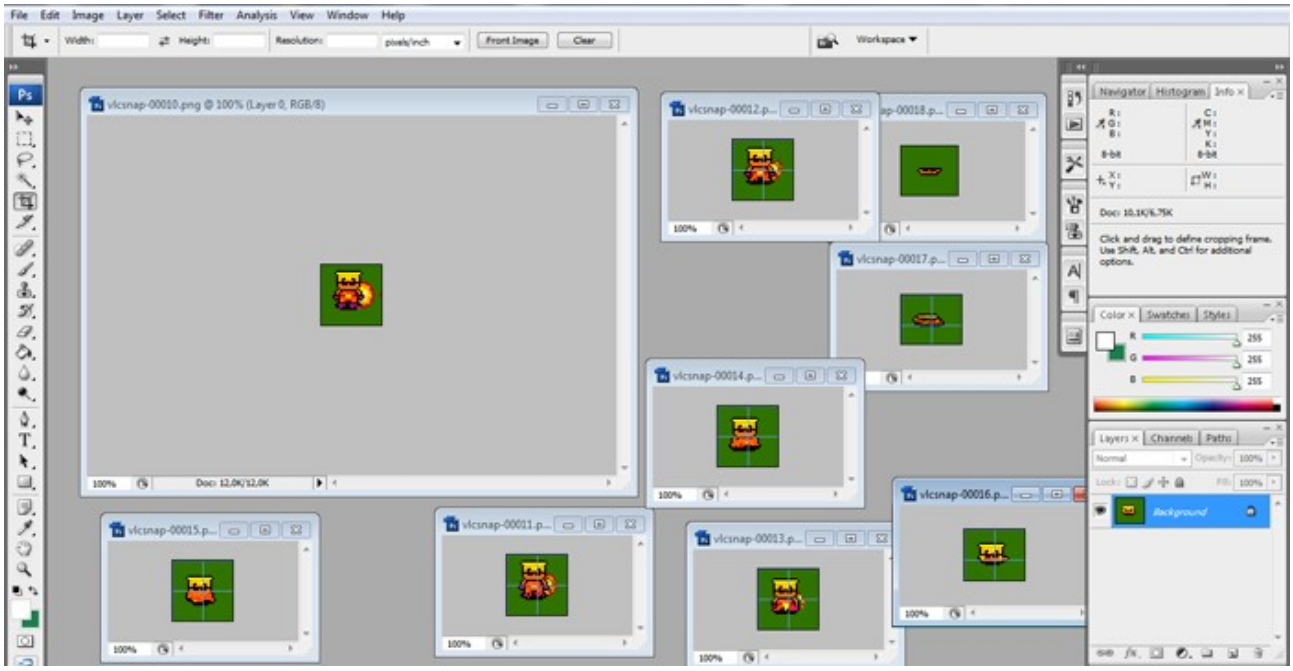
```
Crafty.sprite(32, "sprites.png", {
            money: [1, 7]});
```

Hier wird von vornherein die Seitenlänge des quadratischen Definitionsbereiches in Höhe von 32 Pixeln festgelegt. Die Zahlen in den eckigen Klammern geben die Startposition an, die intern mit der Seitenlänge multipliziert werden. Das heißt, dass in der Spritedatei die Startposition der Auswahl einmal um 32 Pixel nach rechts und 224 Pixel (7x32) verschoben wird, ehe der Auswahlbereich von 32x32 Pixeln gezogen wird. Diese Form der Auswahl ist natürlich nur möglich, wenn alle Bildelemente die exakt gleiche Größe aufweisen und nahtlos aneinander liegen, da jede Verschiebung zur Folge hätte, dass ausgewählte Dateien abgeschnitten werden und unvollständig erscheinen.

Während statische Sprites nur einmalig aufgerufen werden, müssen Sprites, die Teil einer Animation sind besonders verbaut werden. Bei einer zerstörten Wand beispielsweise handelt es sich um eine Abfolge von 4 verschiedenen Sprites, die nacheinander abgespielt werden müssen. Hierfür werden die einzelnen Phasen der Zerfall-Animation nebeneinander gesetzt und nach Initiierung der Reihe nach aufgerufen. Die Geschwindigkeit, wie schnell solch eine Animation abgespielt wird, wird dabei von der Anzahl der sich wiederholenden Sprites definiert. Je öfter derselbe Frame hintereinander abgespielt wird, ehe der nächste Sprite geladen wird, desto langsamer wirkt die Animation.

Für die Spielfiguren musste ähnlich vorgegangen werden, nur dass diese aus wesentlich mehr Animationsphasen bestehen und diese auch nur durch unterschiedliche Events getriggert werden können.
Für die Spielfiguren wurde, wie bereits erwähnt wegen der Abweichenden Größenverhältnissen zu den Umgebungsgrafiken eine separate Sprite-Datei verwendet in der die Anordnung aller Animationsphasen einheitlich nach einem Muster gerichtet war, damit sich die Figuren durch Knopfdruck exakt gleich bewegen und die entsprechenden Zustände wie „Tod“ oder „Unbesiegbarkeit“ nach demselben Algorithmus aufgerufen werden können.





Die Sprites für den „Unbesigbarkeit"-Status konnten relativ simpel reproduziert werden, da es sich bei diesem Figuren-Zustand um nichts anderes, als eine komplett weiße Silhouette der normalen Spieler-Sprites handelt. In diesem Falle mussten nicht umständlich 12 neue Screenshots gemacht und zugeschnitten werden, sondern es konnten die vorhandenen Sprites mit der RGB-Farbe #ffffff eingepinselt werden.

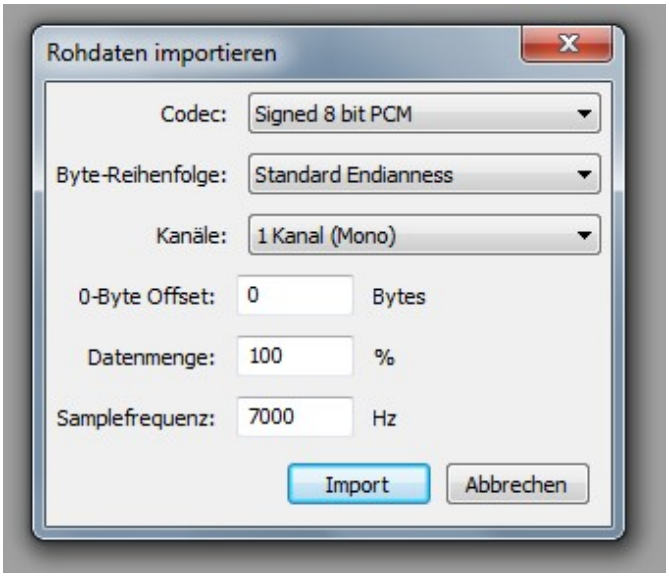
Natürlich konnte dieser Prozess erst auf die letzten 4 Charaktere angewandt werden, da beim ersten Durchgang diese weißen Sprites auf die tatsächliche Identität überprüft werden mussten. Kurioserweise stellte sich tatsächlich ein minimaler Unterschied heraus, nämlich dass die Figur über einen zusätzlichen abstehenden Pixel im Kopfbereich verfügte. Dieses Detail war für mich so unbedeutend, dass es nicht weiter berücksichtigt wurde, da es sich vermutlich sogar um einen unbeabsichtigten Fehler des Ursprungs-Entwicklers gehandelt haben könnte.

4. Sound Implementierung

Die Implementierung der Geräuschkulisse entpuppte sich als eine der schwierigsten Disziplinen, da einige Anforderungen wegen eines zu hohen Aufwandes und nicht ausgereifter Technik nicht erfüllt werden konnten.
Anders als die visuellen Elemente des Spiels mussten Audio Dateien (größtenteils) nicht aus dem Spiel herausgefiltert werden, sondern lagen bereits in Rohformat vor und wurden vom betreuenden Professoren zu Beginn des Projekts zur Verfügung gestellt.
Diese lagen in dem Datentyp SPL vor, ein abstraktes Format das zu der Zeit in der das Spiel erschienen ist noch gar nicht existierte. In welchem Dateiformat die Sounddateien ursprünglich vorlagen ist nicht gewiss, eben sowenig weshalb sie als SPL Dateien vorhanden waren. Es verstrich zunächst einige Zeit in die Recherche nach dem Wesen und der Nutzung dieser „Spool" Files, bis sich herausstellte, dass es völlig ausreichte diese von einem Audio-Converter wie „Audacity" übersetzen zu lassen.

Die Spool Dateien mussten in Audacity als RAW-Files importiert
und und in der Signalverarbeitung angepasst werden. Für einen gemächlichen Klang in Normalgeschwindigkeit musste die Samplefrequenz auf etwa 12 kHz runter gesetzt werden, wobei einige Dateien weniger oder auch mehr Kilo Hertz benötigten, um der Abspielgeschwindigkeit des Original Spieles so nahe

wie möglich zu kommen.
Für den Ausgang musste der Mono-Kanal für sämtliche Dateien gewählt werden, da es bei Stereo bereits zu eklatanten Qualitätseinbußen kam. Ebenfalls von Belang war auch der Codec, der auf 8 bit runter gesetzt werden musste, jedoch nicht wie zu Beginn vermutet unvorbehaftet sondern signiert sein musste, was einen erheblich angenehmeren, vollkommen rauschfreien Klang zum Resultat hatte.



Somit wurden für alle Dateien der Codec 8 bit signed und der Mono-Kanal gewählt, während nur die Samplefrequenz eine individuelle Betreuung brauchte.

Bei sämtlichen Sound-Dateien handelt es sich bis auf eine Ausnahme um Sound-Effekte, die pro Initiierung einmalig aufgerufen werden. Die Ausnahme bildet die Hintergrundmusik, die durchgehend eine komplette Spielrunde begleitet. Diese liegt jedoch nur in einem drei sekündigen Sample vor, da Selbiges bereits die komplette Komposition der Hintergrundmusik ausmacht. Im Originalspiel wird das Sample in einer Wiederholschleife abgespielt und dabei mit jeder Wiederholung dezent schneller, sodass die Musik nach knappen 90 Sekunden sehr hektisch wird.
Ursprünglich war die Idee das Sample mittels Java Script zu loopen und die Frequenz zu parametrisieren, was letztlich jedoch an der Tatsache scheiterte, dass nahtlose Wiederholschleifen von Sound-Dateien mit HTML und JavaScript einfach nicht möglich sind. Zunächst fiel der Verdacht für diese unsaubere Wiederholschleife auf das verwendete mp3-Format, das durch die Kompression immerhin den Soundkanal mit einer verschwindend kurzen Zeitspanne von Geräuschlosigkeit einleitet. Doch die Verwendung von sauber geschnittenen

WAV-Dateien hatte denselben Effekt, was zeigte, dass es an diesem Faktor nicht liegen konnte, zumal ein generierter Loop in Audacity tatsächlich nahtlos abläuft ungeachtet des Dateiformates.

Um das Problem in den Griff zu bekommen, bot sich die JavaScript Sound Engine „Soundmanager2" an, das zahlreiche Soundmodifikationsfeatures zur Verfügung stellte, jedoch über keine Funktion verfügte, die in die Samplefrequenz eingreifen und diese parametrisieren könnte. Zudem konnte auch hier der an sich simple Anspruch eines nahtlosen Soundloops nicht erfüllt werden, weswegen die Verwendung dieser Engine und die damit investierte Zeit für die Einarbeitung wieder verworfen wurde.

Die unschönere Alternativlösung war daher die Musik nicht zur Laufzeit zu loopen, sondern sie vorher in einem Stück komplett zu generieren und dann abspielen zu lassen. Die einfachste Möglichkeit war es mit Hilfe des Emulator internen Audio Recorders den kompletten Sound-Output einer Spielrunde aufzunehmen und diesen dann auch als tatsächliches Endprodukt zu verwenden. Selbstverständlich wurden dabei in dieser Runde keine der Spielfiguren angefasst, um die Hintergrundmusik nicht durch die anderen Soundeffekte zu verfremden. Das Problem dabei jedoch war, dass unweigerlich auch der Alarm-Sound, der immer gegen Ende einer Runde gestartet wurde mit aufgenommen wurde, was ein unreines Klangergebnis mit sich zog, gerade weil der Sound für den Alarm separat vorliegt und bezüglich einer möglichen Skalierbarkeit einer Runde separat initiiert werden könnte. Um das Handicap zu kompensieren beschränkten wir uns auf die Originalspieldauer, die immer 2 Minuten und 15 Sekunden dauert wo der Alarm-Sound immer an derselben Stelle ansetzt.

Für die Implementierung der einzelnen Sound-Dateien in das neu geschriebene Programm bot sich natürlich die hauseigene Audio-API von Crafty an. Diese jedoch funktionierte nicht zuverlässig und nutzte sich etwas unbequem, daher

wurde auf reines JavaScript zurück gegriffen, das die Sounds tadellos abspielen konnte. Zwar bot die Crafty Engine auch einige Zusatzfunktionen wie eine Wiederholrate (die jedoch auch nicht nahtlos funktionierte) und einen Lautstärke-Regler, doch für unsere Zwecke hat JavaScript vollkommen ausgereicht.

Um in JavaScript eine Sound-Datei zu definieren genügt folgende Code-Zeile:

```
var exp = new Audio("Sounds/explode.mp3");
```

Die Sound-Datei „explode.mp3“, die sich im Ordner „Sounds“ relativ zum Skript befindet, wird der Variablen „exp“ zugewiesen, welche fortan wie folgt verwendet werden kann: `exp.play();`
Entsprechend wurden am Kopf der Quellcodedatei sämtliche zu verwendenden Dateien definiert und an der geforderten Stelle im Quellcode aufgerufen.

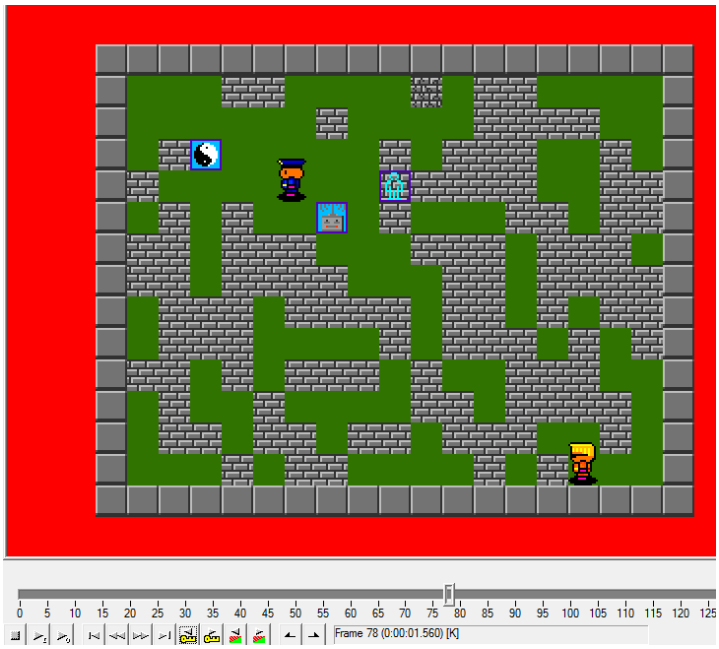
5. Die Alarm Funktion

Wie bereits im letzten Kapitel vorweggenommen endet jede Runde etwa 30 Sekunden vor Ende mit einem Alarm, der in einer Wiederholschleife das Spielgeschehen dramatisiert. Doch nicht nur audiotechnisch wird diese Endphase einer Spielrunde angesprochen, sondern auch visuell in Form eines rot pulsierenden Hintergrundes. Auch dieser Effekt lies sich unabhängig von der Sprite Engine durch reines JavaScript bzw. später durch Ergänzung und Optimierung mit jQuery realisieren. Der Algorithmus für die Visualisierung dieser Funktion sah wie folgt aus:

```
var startcolor = 0;
var neg = 1;
function backgroundAlarm(red) {
	var red;
	var frameskip = 40;
	document.body.style.backgroundColor = 'rgb(' + red + ', 0, 0)';

	setTimeout(function() {

		if(red>=255 || red<0){
			neg=neg*(-1);
		}
		backgroundAlarm(red=red+frameskip*neg);

	}, 100);
}
```

Die Funktion „backgroundAlarm“ erhält einen Eingabeparameter („red“), welcher den Rot-Kanal des RGB-Farbraumes markiert. Auf diesen wird JavaScript typisch über „document.body.style.BackgroundColor“ bzw. mit „$(body).css("background-color", "rgb("+ red + ",0,0)");“ in jQuery zugegriffen, wobei der Grün- und Blaukanal unberührt und konstant auf „0“ bleiben. Der Ausdruck wird dem Style-Selektor als String übergeben, jedoch mit der red-Variablen als veränderlichen Wert. Dieser Wird anschließend in der setTimeout-Funktion innerhalb eines Gültigkeitsbereiches zur Laufzeit verändert was durch eine Rekursion der Hauptfunktion erfolgt. Innerhalb des Timeouts wird die Funktion in sich neu aufgerufen, wobei der Eingabeparameter mit den vorher initialisierten „frameskip“ addiert wird, welcher wiederum mit der neg-Variablen multipliziert wird, was jedoch keine Auswirkungen hat, da diese auf 1 initialisiert ist. Sollte jedoch die red Variable einen Extremwert erreichen, also entweder 0 oder 255, wird die Variable „neg“ negiert, was beim rekursiven Aufruf zur Folge hat, dass der Rotwert ab 255 wieder um den frameskip subtrahiert wird und erst wieder positiv aufsteigt, sobald die 0 erreicht ist.

Der Frameskip dient hierbei also quasi als Mindestwert, der angibt in welchem Maße sich der Rot-Wert verändert. Je kleiner er ist, desto feiner verläuft der Übergang von Schwarz zu Rot, dauert aber auch umso länger. Der Timeoutwert bestimmt das Zeitintervall in dem die Funktion stattfindet. Durch geschickte Kombination lassen sich beide Werte für denselben Zweck nutzen und einen angenehmen pulsierenden Farbverlauf in Echtzeit visualisieren.



Um jedoch der Framerate im Originalspiel nahe zu kommen wieder mal über ein Video das Frameinterwall bestimmt, das die nötigen Zahlen für die Funktion lieferte. Hier habe ich gemessen innerhalb wie vieler Frames ein kompletter Rot/Schwarz Zyklus vollzogen wird und den Wert in die Funktion implementiert.

Damit diese Funktion auch zur gewünschten Zeit aufgerufen wird, muss sie in einer weiteren globalen Timeout Funktion aufgerufen werden:

```
setTimeout(function() {
            backgroundAlarm(startcolor);
}, 10000);
```

Hier erhält die bereits erläuterte Funktion backgroundAlarm die globale auf 0 initialisierte Variable „startcolor" als Eingabeparameter, damit der Effekt auch bei beim tiefsten Schwarz Wert beginnt und wird 10 Sekunden nach Spielbeginn gestartet.

6. Reflexion über die eigene Tätigkeit

Die Arbeit an der Wiederbelebung eines alten Computerspiels war für mich eine sehr interessante Erfahrung da systematische Zerlegen der Konstruktionselemente Einblicke in das fertige Spiel und die Logik gewährte, die beim gewöhnlichen Spielen verwehrt geblieben wären. Die Arbeit war nicht immer angenehm, da doch gerade die meisten Workflows im Reverse Engineering schnell in stumpfsinnige Repetition mündeten.
Dennoch hatte sich auch ein gewisser Reiz bei der Sprite Extraktion bemerkbar gemacht, der mich motiviert eigenständig weitere Klassiker aus der 8-Bit und 16-Bit Ära für solch ein Projekt herzunehmen und diese in einen neuen Spielekontext, z.B. in Form eines Crossovers zu bringen.
Das größte Manko war nach wie vor, die eingeschränkten Verarbeitungsmöglichkeiten von Sound-Dateien in JavaScript, was an sich fast schon als eigene Projektarbeit angesehen werden könnte.
Rückblickend bin ich etwas enttäuscht wegen der Verarbeitung des Gamematerials nur wenig mit der eigentlichen Programmierung zu tun gehabt zu haben. Andererseits war mein Tätigkeitsfeld sehr vielfältig, da ich in grundverschiedenen Disziplinen tätig war, die mich zu einem Allrounder konditioniert haben. Daher gehe ich trotz einiger nicht gelöster Probleme zufrieden aus dieser Projektarbeit heraus und freue mich darauf eigenständig ein neues Projekt dieser Art zu starten.