GEORG-SIMON-OHM
HOCHSCHULE NÜRNBERG

Fakultät Elektrotechnik Feinwerktechnik Informationstechnik

Gesamtdokumentation von

Sebastian Adam, Sergej Bjakow, Michael Kao,
Pavlina Pavlova und Maximilian Seyfert

Bachelor Media Engineering – 6. Semester

„Gesamtdokumentation für das Projekt „ReMasterBlaster“ im Rahmen des Studienprojektes im 6. Semester des Bachelorstudiengangs Media Engineering“

Sommersemester 2012

Bestätigung gemäß § 35 (7) RaPO

Adam, Sebastian
Bjakow, Sergej
Kao, Michael
Pavlova, Pavlina
Seyfert, Maximilian

Wir bestätigen, dass wir die Gesamtdokumentation mit dem Titel:

**„Gesamtdokumentation
für das Projekt „ReMasterBlaster"
im Rahmen des Studienprojektes im 6. Semester
des Bachelorstudiengangs Media Engineering"**

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

Datum: 03.08.12

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
Erstellung der Sprite Zuordnung	04_Quellcode/finalGame/sprites.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

Modifikation der Sounds zu mp3 Dateien	04_Quellcode/finalGame/sounds/*	Sergej Bjakow Maximilian Seyfert
Capturing der einzelnen Elemente (Sprites)	05_Materialien/Sprites/*	Sergej Bjakow
Zusammenfügen der Sprites	04_Quellcode/finalGame/ sprites.psd	Michael Kao
Vorbereiten der einzelnen Szenenelemente	04_Quellcode/finalGame/img/*	Pavlina Pavlova Sergej Bjakow Michael Kao
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
Erstellung des Wikis	http://schorsch.efi.fh-nuernberg.de/mewiki/index.php/Project-BME-2012-07-RemasterBlaster/Project-BME-2012-07-RemasterBlaster	Sebastian Adam
Erstellung der Gesamtdokumentation	03_Abschluss/Gesamtdokumentation	Sebastian Adam

INHALTSVERZEICHNIS

1. Ausgangssituation

Ausgangssituation für die Projektarbeit „ReMasterBlaster" war die für das 6. Semester im Bachelorstudiengang Media Engineering angesetzte Projektarbeit. Als Auftraggeber traten in unserem Fall Prof. Dr. Matthias Hopf sowie Prof. Dr. Stefan Röttger in Erscheinung. Der zeitliche Rahmen umfasste den Zeitraum vom 26.03.2012 bis letztendlich 03.08.2012. Das Projektteam für das Projekt „ReMasterBlaster" setzte sich aus folgenden Mitgliedern zusammen: Sebastian Adam, Sergej Bjakow, Michael Kao, Pavlina Pavlova und Maximilian Seyfert.

2. Projektidee

Die ursprüngliche Projektidee stammte von Prof. Dr. Matthias Hopf, welcher auf Nachfrage von Sebastian Adam das Projektthema zur Verfügung stellte. Hintergrund des Ganzen war, dass der Fokus auf Reverse Engineering[1] gelegt werden sollte. Hierbei gilt es ein bereits vorhandenes Produkt – in unserem Fall der AMIGA Klassiker MasterBlaster – in seine einzelnen Bestandteile zu zerlegen und nachzukonstruieren. Es sollte hierbei jedoch nicht wieder ein Spiel für die AMIGA entstehen, viel mehr wollte das Projektteam das Spiel für die breite Masse zugänglich machen. Also entschieden wir uns für die Umsetzung mittels neuer Webtechnologien im Internetbrowser der jedem Endanwender heutzutage zur Verfügung steht. Hierbei kamen die Markup Language HTML5[2], CSS[3] und JavaScript[4] sowie die Bibliothek jQuery[5] und die Crafty JavaScript Sprite Engine[6] zum Einsatz.

[1] Eine genauere Definition finden Sie hier:
http://de.wikipedia.org/wiki/Reverse_engineering
[2] Hyper Text Markup Language, eine Auszeichnungssprache im Internet
[3] Cascading Style Sheets, eine Möglichkeit HTML-Inhalte zu layouten
[4] eine Skriptsprache, die auf der Clientseite im Browser abläuft
[5] populäre JavaScript Bibliothek, www.jquery.com
[6] die von uns verwendete Sprite Engine, mehr Infos unter www.craftyjs.com

3. Rollenvergabe

Die Rollenvergabe soll erzielen komplexe Aufgaben, die sich über einen längeren Zeitraum erstrecken, zu strukturieren und somit die Arbeit innerhalb des Projektteams zu erleichtern sowie einen harmonischen Arbeitsfluss zu gewährleisten.

Das Team der Projektgruppe „ReMasterBlaster" entschied sich zu Beginn des Projekts ebenfalls eine Rollenvergabe vorzunehmen. Als Programmierer der Spielelogik traten Michal Kao sowie Maximilian Seyfert in Erscheinung. Pavlina Pavlova wurde dem GUI[7] zugeteilt und Sergej Bjakow kümmerte sich größtenteils um den Teil des Reverse Engineerings um unser neues Spiel mit möglichst allen Details nahe am Original erscheinen zu lassen. Sebastian Adam wurde nach einer kurzen Besprechung mit allen Teilnehmern zum Projektleiter ernannt. Diese Rolle ist in einem Projekt natürlich von essentieller Bedeutung, da der Projektleiter die Schnittstelle zu den betreuenden Professoren darstellt und für die gesamte Organisation innerhalb des Projektgeschehens zuständig ist.

4. Projektorganisation

Eine gelungene Projektdurchführung ist gekennzeichnet von einer akribischen Projektorganisation. Dies erforderte in unserem Fall regelmäßig veranstaltete Treffen, die Errichtung eines Entwicklungsblogs sowie zu Beginn des Projekts die Einrichtung eines öffentlich (für alle Teammitglieder) zugänglichen Ordners innerhalb der Dropbox[8]. Diese gemeinsamen Meetings und Dokumentations- bzw. Organisationstools werden wir in diesem Kapitel näher erläutern.

4.1 Meetings

Zur optimierten Kommunikation sowie zur Stärkung und Verbesserung der Teamarbeit wählten wir die Option, dass wir wöchentlich mindestens ein

[7] Graphical User Interface
[8] Eine kostenlos angebotene Cloud Speicherlösung im World Wide Web

gemeinsames Treffen abhielten. Hier wurden jeweils von den Teammitgliedern die neuesten Arbeitsergebnisse vorgestellt und wiederum neue Aufgaben verteilt. Die Aufgabe eines Jeden von uns war es zudem abwechselnd Sitzungsprotokolle dieser wöchentlich stattfindenden Teammeetings zu erstellen, um sich auch rückblickend über die einzelnen Treffen informieren zu können. Diese Protokolle wurden immer zeitnah nach dem eigentlichen Treffen den anderen Mitgliedern in digitaler Form zur Verfügung gestellt. Einen Großteil der Protokollaktivitäten übernahm im Laufe des Projekts Sebastian Adam, in der Rolle des Projektleiters.

Gegen Ende des Projekts – wo erfahrungsgemäß die Zeit immer knapper wird und noch dazu die reguläre Prüfungszeit ansteht – führten wir zudem sogenannte „Projekttage" ein bei denen gezielt eine Aufgabe bis zur Vervollständigung verfolgt wurde. Hier ist z.B. das „mergen"[9] der beiden Programmteile (Spielelogik und GUI) zu erwähnen.

4.2 Blog

Da das Projekt „ReMasterBlaster" ein Open Source Projekt darstellt sollten auch die Fortschritte, die im Projekt getätigt wurden, öffentlich für Jedermann einsehbar sein. Hier entschieden wir uns dazu einen Blog einzurichten. Den Webspace der dafür nötig war stellte Sebastian von seinem privaten Account bei einem Hoster für das Projekt zur Verfügung. Wir entschieden uns jedoch zusätzlich eine „internationale" Domain für den Zeitraum des Projekts anzumieten. Hier entschieden wir uns für den Namen: http.//www.remasterblaster.com – wie innovativ! Als Blogsystem kam Wordpress zum Einsatz. Sebastian passte das Layout und das Design des Blogs ein wenig an unsere Bedürfnisse an und von nun an war es seine Aufgabe einmal wöchentlich über das Projektgeschehen in englischer Sprache zu informieren. Diese Aufgabe war auch deswegen sehr hilfreich für ihn, da er dadurch öfter als nur einmal (beim wöchentlichen

[9] zusammenfügen von einzelnen, vorher unabhängig voneinander entwickelten, Programmteilen

Teammeeting) mit den einzelnen Teammitgliedern in Kontakt trat um den aktuellen Stand der Entwicklungsarbeiten zu erfragen. Diese Tatsache hat ihm sehr geholfen den Überblick über das gesamte Projekt zu bewahren. Somit konnten nicht nur die Leser des Blogs, sondern auch er in seiner Tätigkeit als Projektleiter, von unserem Entwicklungsblog profitieren.

4.3 Zeitplanung

Die Zeitplanung ist wohl eine der Wichtigsten Aufgaben innerhalb eines Projektes. Hierbei versuchten wir uns an mehreren „Projektplanungstools" von denen viele jedoch eher enttäuschten bzw. nach einiger Zeit kostenpflichtig waren. Um die Zeitplanung für unser Projekt letztendlich durchzuführen nutzte Sebastian das Open Source Tool „Gantt Project". Im weiteren Verlauf werden wir kurz die einzelnen Phasen des Projektes vorstellen und anschließend den Gantt – Chart[10] der aus diesen Phasen und Teilaufgaben resultierte erläutern.

1. **Planungsphase (26.03. – 10.04.2012)**
   Einarbeitung in den Bereich der Sprite Engines
   Erstellung einer Testumgebung mit diversen Sprite Engines
   Definition des Lastenheftes
   Erstellung eines Projektablaufplans
2. **Entwicklungsphase (10.04. – 03.07.2012)**
   Entwicklung einer Spielumgebung anhand des Originals
   Implementierung der Bewegungsfunktion eines Spielers
   Einbindung der Goodies nach einer zufälligen Verteilungswahrscheinlichkeit
   Abstraktion des Player-Objekts um mehrere Spieler einbinden zu können
   Implementierung des Konfigurationsmenüs
   Erstellung einer Hall of Fame die die Sieger nach einer Spielrunde anzeigt

[10] Zeitlinie über das gesamte Projekt

Entwicklung einer Shrinking-Funktion, welche das Spiel nach einer vorgegebenen Zeit beendet

3. Dokumentationsphase (03.07. – 17.07.2012)

Erstellung der Präsentationen (Zwischen- und Abschlusspräsentation)
Erstellung der Prüfungsstudienarbeiten der einzelnen Teammitglieder
Erstellung der Gesamtdokumentation (inkl. Wiki-Eintrag)

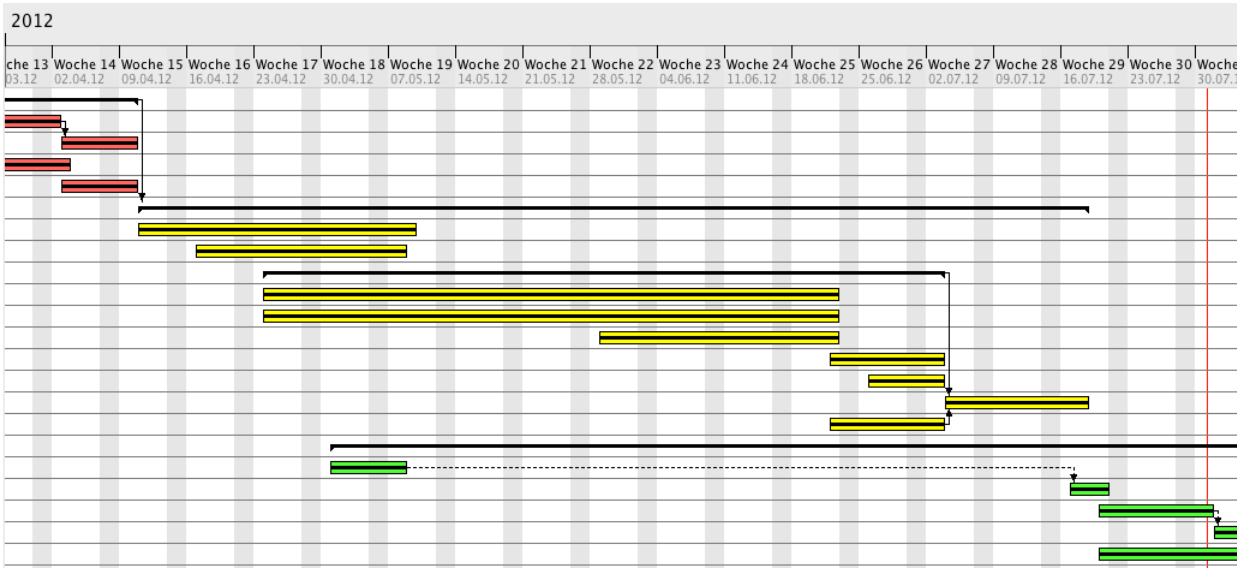

Abbildung 1: Auszug aus dem Gantt-Chart erstellt mit GANTTproject

Abbildung 1 zeigt einen Auszug aus dem Gantt-Chart des Zeitplans des Projektes „ReMasterBlaster". Was man sehen kann ist, dass sich der Zeitplan um 17 Tage nach hinten verschoben hat. Diese Tatsache kam dadurch zustande, da wir die Dokumentationsphase eigentlich komplett in den Prüfungszeitraum eingeplant haben und die anstehenden Prüfungen etwas unterschätzt haben. Zudem gab es Mitglieder im Team die noch Wiederholungsprüfungen aus vorangegangenen Semestern nachholen mussten. Was auch zu sehen ist, dass sich die Phasen teilweise überlappen. Beispielsweise fällt eine Aufgabe, nämlich die Vorbereitung und Erstellung der Zwischenpräsentation am 09. Mai 2012, mitten in die Entwicklungsphase. Dies ist aber nur der Tatsache geschuldet, dass wir die Zwischenpräsentation ebenfalls der Dokumentationsphase zugeordnet

haben. Alles in allem sind wir vom zeitlichen Ablauf her sehr zufrieden mit dem Projekt.

4.4 Facebook Gruppe

Es mag im Zusammenhang mit einer Projektorganisation vielleicht etwas profan klingen wenn man sagt, dass man eine Facebook Gruppe erstellt, jedoch erwies sich dies in unserem Fall als sehr nützlich. Jedoch war dies eine gute Möglichkeit um kurzfristige Änderungen an alle Teammitglieder zu kommunizieren. Auch wurden hier Umfragen gestartet welcher Termin am Geeignetsten für ein nächstes Projekttreffen bzw. zum Schluss den Projekttag erscheint. Da man heutzutage Änderungen und Mitteilungen auch per Push-Nachricht auf sein Smartphone erhält konnte man so gezielt und schnell die Projektgruppe erreichen.

4.5 Google Kalender

Der Google Kalender erschien uns ebenfalls als probates Mittel um die zeitliche Organisation der Teammitglieder zu optimieren. Hier verhält es sich ähnlich zu Punkt 4.4, dass jeder Teilnehmer der Projektgruppe einen Google Account hat. Diese Tatsache nahmen wir zum Anlass einen gemeinsamen Kalender für das Projekt zu erstellen welcher dann für alle in der Gruppe freigegeben wurde. So konnte jeder diesen „Projektkalender" in seinen eigenen Kalender mit einfügen und hatte stets die anstehenden Termine im Blick. Alles in allem war dies eine sehr einfache Möglichkeit die Termine zu koordinieren und im Blick zu halten.

5. Definition des Lasten- und Pflichtenhefts

Da zu einem Projekt normalerweise natürlich immer Auftraggeber und Auftragnehmer gehören musste Sebastian als Projektleiter das Lasten- sowie das Pflichtenheft erstellen. Im Normalfall wird ein Lastenheft vom Auftraggeber spezifiziert und z.B. für eine Ausschreibung des gewünschten

Projekts verwendet. Es beschreibt die Anforderungen die der Auftraggeber an das fertige Produkt stellt. Die Formulierungen hierbei sollten natürlich so allgemein wie möglich und so einschränkend wie nötig gewählt werden damit man während der Entwicklung immer noch Spielraum für etwaige Änderungen hat.

Im Pflichtenheft hingegen wird in konkreter Form beschrieben wie der Auftragnehmer gedenkt die an ihn gestellten Anforderungen zu lösen. Hier wurden von uns die von der Projektgruppe gesteckten Ziele niedergeschrieben. In den folgenden drei Abschnitten beschreiben wir die selbst gesetzten Muss-, Soll- und Kann-Kriterien, da diese die wichtigsten Punkte aus dem Pflichtenheft darstellen.

5.1 Muss-Kriterien

Am Ende des Projekts muss ein spielbares Produkt von „ReMasterBlaster" vorliegen wobei zwei Spielfiguren auf einem Spielfeld mittels einer Tastatur gesteuert werden können. Es müssen Bomben gelegt und Wände gesprengt werden können. Des Weiteren wird die Spielfigur eines Gegenspielers vom Spielfeld entfernt wenn diese innerhalb eines Bombenradius getroffen wird.

5.2 Soll-Kriterien

Es sollen weitere Spieldetails umgesetzt werden. Beim Wegsprengen einer Wand sollen verschiedene Goodies, welche aus dem Original bekannt sind, erscheinen und der Spielfigur, die das jeweilige Goodie einsammelt, zusätzliche Eigenschaften übertragen. Hierbei sollen folgende Goodies berücksichtigt werden:

- Additional Bomb (man erhält eine extra Bombe)
- Higher Range (der Bombenradius erhöht sich um ein Feld)
- Superman (man kann Mauernblöcke verschieben)
- Ghost (man ist nur 1x1px groß und kann durch Wände gehen)
- Controller (man kann die Bombe fernsteuern)

- Invincible (man ist für eine Aktion, die die Spielfigur des Spielers normalerweise vom Feld entfernen würde, geschützt → Spielfigur wird weiß)
- Remote Detonated Bomb (Bombenexplosion ist zeitgesteuert)
- Speed Up (man wird schneller → max. 4x möglich)
- Stopschild (alle anderen Spieler drehen sich auf der Stelle)
- Fragezeichen (hier verbirgt sich eines der folgenden Goodies: Protection, Ghost, Stopschild oder Krankheit → Timer für Bombenexplosion verkürzt sich)
- Death (die Spielfigur des Spielers wird entfernt)

Weiterhin sollen verschiedene Szenen definiert werden, die in einer vordefinierten Reihenfolge ablaufen. Am Anfang soll ein Startscreen mit dem Logo erscheinen, auf den das Konfigurationsmenü folgt. Die Konfigurationsmöglichkeiten sollen mittels eines JSON-Objekts eingebunden werden. Nachdem man die Konfiguration abgeschlossen hat soll man in die eigentliche Spielszene gelangen wo man dann das Spiel spielen kann.

5.3 Kann-Kriterien

Im Optimalfall, vor Allem aber falls am Ende der Muss- und Sollkriterien noch Zeit vorhanden sein sollte, kann überprüft werden ob mehr als 2 Spieler möglich sind. Des Weiteren kann eine Analyse stattfinden ob andere Eingabegeräte eingebunden werden können. Ebenfalls kann überprüft werden, ob das Spiel auch in Firefox, IE (v9 oder grösser) und Safari läuft bzw. lauffähig gemacht werden kann.

6. Engine Auswahl

Nachdem jeder von uns sich das Spiel auf einem Emulator angeschaut hatte, war es an der Zeit, dass wir mit der Planung anfangen. Schon vor

der ersten Vorbesprechung haben wir von Prof. Hopf eine Tabelle[11] mit einer Übersicht sämtlicher JavaScript Game-Engines bekommen.
Ein "Remake von MasterBlaster mit aktuellen Technologien" bedeutete für uns, dass wie eine passende Spiele Engine brauchen. Herr Hopf hat uns drei Kriterien genannt, auf die wir achten sollten:

- eine Sprite Engine, um Spielgrafik anzuzeigen.
- schnelle Animation, damit viele Animationen gleichzeitig ruckellos ablaufen können (dass es möglich ist einen "Teppich aus Bomben zu legen, die gleichzeitig explodieren")
- z-Tiefenwert unterstützt.

Uns, als Gruppe, war es wichtig, dass die Engine aktuell, gut dokumentiert und vollständig Implementiert ist.
Die über 50 Game-Engines haben wir zu einer Rangliste mit 7 Favoriten gekürzt. Es ist ziemlich schwierig 50 Engines gleich zu überprüfen, wenn man sich wenig auskennt, aber wir haben da eine sehr einfache Regel befolgt - "dress to impress". Wenn eine Engine eine nicht "seriös" genug wirkende Webseite hatte, wurde sie durchgestrichen. Dies hat uns eine Menge Ärger und Zeit gespart, weil so sind solche Engines weggefallen, die nicht mehr unterstützt und gepflegt werden, auch solche die nicht vollständig implementiert sind.
Als diese Rangliste stand war unsere Aufgabe eine Testbench für unsere Favoriten zu schreiben, damit wir die Schnelligkeit der Animation testen könnten. Als Favorit aller Teammitglieder hat sich die Engine Crafty rausgestellt.
Die Testbench bestand aus ein Feld der Größe 25x21 Blöcke. Jeder Block enthielt eine 16 Pixel Figur, die sich in Endlosschleife gedreht hat.

6.1 Erste Tests

Als die Testbench das bestätigt hat, was wir uns erhofft haben - dass Crafty die Engine für uns ist, war es unsere Aufgabe die erste Spielumgebung

[11] https://github.com/bebraw/jswiki/wiki/Game-Engines

damit aufzubauen. Da hat uns die ausführliche Dokumentation auf der Crafty Webseite sehr geholfen. Die gut dokumentierte Seite und ihre Tutorials waren einige der Hauptgründe, wieso uns diese Engine so gut gefallen hat. Die einfachen Tutorials haben uns bei der Erstellung der Umgebung geholfen, so dass nach kurzer Zeit die erste bewegbare Figur in einem 15x19 Feld stand. Alles wurde mit Crafty-eigenen Funktionen realisiert, die eine Steuerung und Animation erleichtern.

7. 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 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. 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.

7.1 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, nämlich 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 wir uns 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 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 Grün-Ton zu treffen wurde lediglich der Bildschirm extrahiert, um mit der Pipette in Photoshop den exakten RGB-Wert zu ermitteln.

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 auf 5.

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, das Erscheinen jeder Entität notiert. Es stellte sich heraus, dass die häufigste Konsequenz aus einem gesprengen Block der „Bröckel"-Status ist, der darauf hinweist, dass der Block noch zwei weitere male gesprengt werden muss, ehe er vollständig beseitigt worden ist. Die Aus der Anzahl der erschienenen Items wurde der Durchschnitt berechnet und dieser in Relation zu den gesprengten Blöcken gebracht und sich dabei herausstellte, dass die Summe aller Wahrscheinlichkeiten knapp über 50% lag, was bedeutet, dass die andere Hälfte leere Blöcke waren.

7.2 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 Emulator interne 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 Player interne Snapshot 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 Hintergrundes. 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 haben wir 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 sind wir bei den späteren Spielfiguren so fortgefahren 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 wurde 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.

Während statische Sprites nur einmalig aufgerufen werden, müssen Sprites, die Teil einer Animation sind besonders verbaut werden. Bei 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.

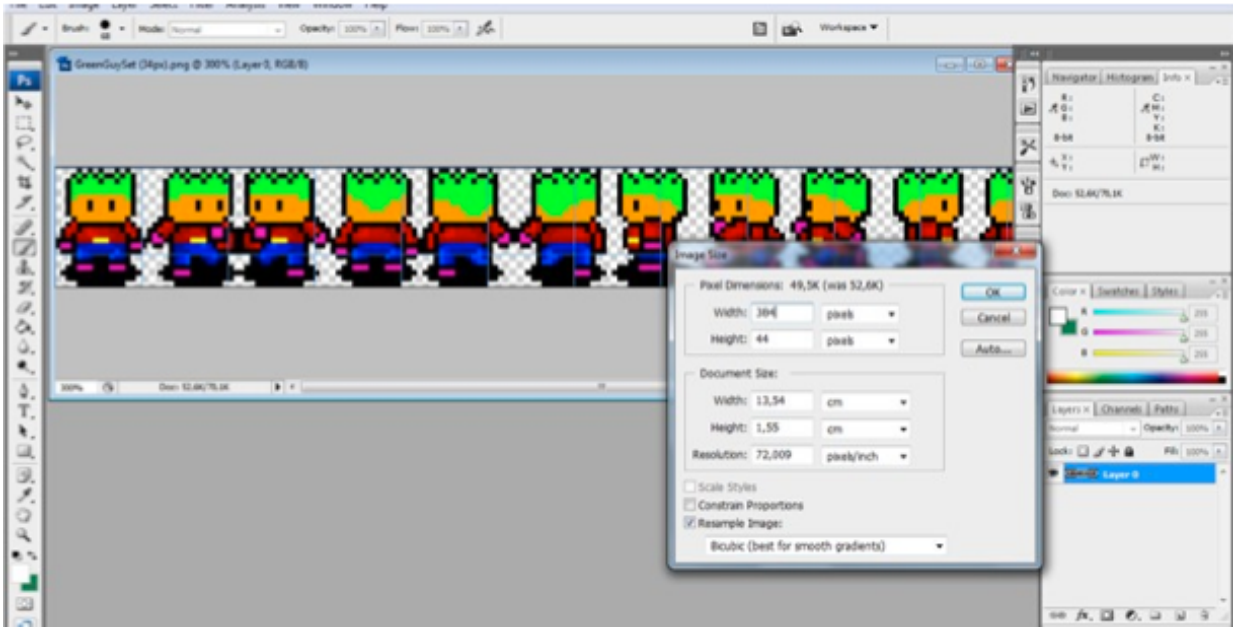

Abbildung 2: Zusammenstellung eines Bewegungsablaufes

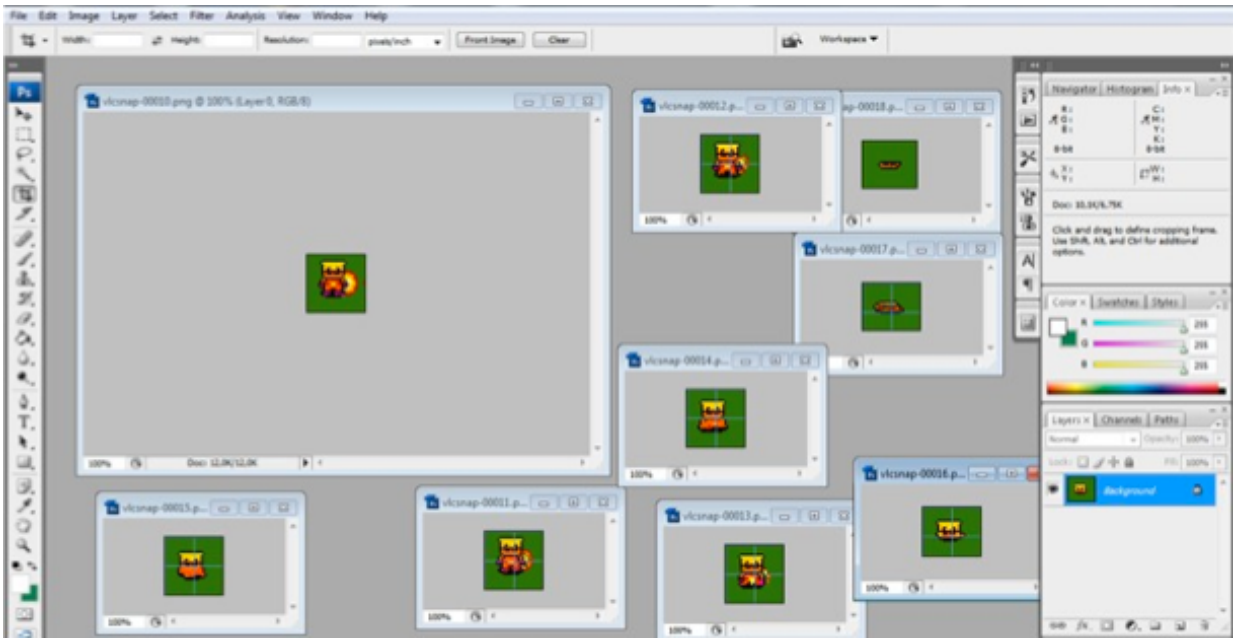

Abbildung 3: Einzelne Phasen einer Animation

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 „Unbesiegbarkeit"-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 uns jedoch so unbedeutend, dass es nicht weiter berücksichtigt wurde, da es sich vermutlich sogar um einen unbeabsichtigten Fehler des Ursprungs-Entwicklers gehandelt haben könnte.

Abbildung 4: "Pixelfehler" im Kopf-bereich des Spielersprite

8. Programmierung der Spielelogik

Im Folgenden werden die Tätigkeiten während der Umsetzung der Spielelogik genauer erläutert. Da die Programmierarbeit nicht linear stattfand und sich einzelne Programmteile immer wieder änderten, gehen wir nun im Folgenden auf die Implementierung der ersten Spieleumgebung ein, sprich die Erstellung der Masterblaster Welt, sowie Spielern mit

rudimentären Funktionalitäten, um dann im zweiten Teil dieses Abschnitts auf das Endresultat der Spielelogik detaillierter einzugehen.

8.1 Erstellung der ersten Spieleumgebung

Wie bereits erwähnt, lieferte das Crafty Tutorial eine gute Basis, um mit der Entwicklung mit Crafty beginnen zu können. Ähnlich wie im Original Masterblaster waren die Inhalte des Tutorials Spielfiguren, eine rechteckige Welt und Gegenstände, die auf der freien Fläche liegen. Wir entschieden uns dafür, als erstes einen lauffähigen Nachbau der Masterblaster Welt zu implementieren. Dies beinhaltete unter anderem die undurchdringbaren Wände „*wall*", den grünen Untergrund, sowie die zufällig verteilten Blöcke „*brick*". Um mit ersten Sprites arbeiten zu können, nahmen wir einen Screenshot aus dem Emulator, und bauten die „wall" und den „brick" in Photoshop nach. Da der Emulator die Spielefläche in einer Größe von 608 * 480 Pixeln darstelle, ergab sich eine Sprite Größe von 32 *32 Pixeln, welche fortan als das Maß für die quadratischen Sprites verwendet wurde. Dies erleichterte das Extrahieren der Sprites und das Weiterverarbeiten ohne skalieren. Um die Bilder einer Ressource zu laden, wurden alle Sprites, außer die Spielersprites, fortan in der Datei sprites.png aufgeführt. Beim Laden der Sprites durch Crafty ist es möglich die Größe anzugeben, in unserem Fall „32". Um nun die jeweiligen Sprites benutzten zu können, wurde ein Objekt übergeben, mit den Namen des jeweiligen Sprites, sowie relativer Position in der Spritesmap. Abbildung 1 zeigt wie die Sprites aus der Spritesmap mittels Crafty.sprite definiert wurden.

```
Crafty.sprite(32, "img/sprites.png", {
      wall: [0, 0],
      brick: [0, 1],
});
```

Abbildung 5: Laden der Sprites

Um die Sprites auf das Spielfeld zu bringen, musste Crafty als erstes mit der „*Crafty.init*" Funktion gestartet werden. Diese erhielt als Parameter die x- sowie y-Länge des Spielfeldes. Mit Crafty.scene wurden zwei Szenen definiert, eine Loading Szene, welche solange „Loading" anzeigt bis alle Spritemaps in den Cache geladen wurden, sowie die Hauptszene, in welcher die Spielewelt definiert wurde. Die Hauptszene startete die „*generateWorld*" Funktion, welche die Blöcke und Wände als Entities generierte. Die Entities sind die zentralen Elemente, welche wichtige Interaktionen ermöglichen. Diese bekommen jeweils als Attribut den Namen des jeweiligen Sprites, sowie x-, y- und z-Position. Die z-Position gibt hierbei den z-Bufferwert an. Die „*generateWorld*" Funktion hat zwei Schleifen, welche über die x- und y-Richtung des Spielfeldes laufen und an den jeweiligen Stellen die richtigen Entities setzt. Das Resultat ist in Abbildung 2 zu sehen.

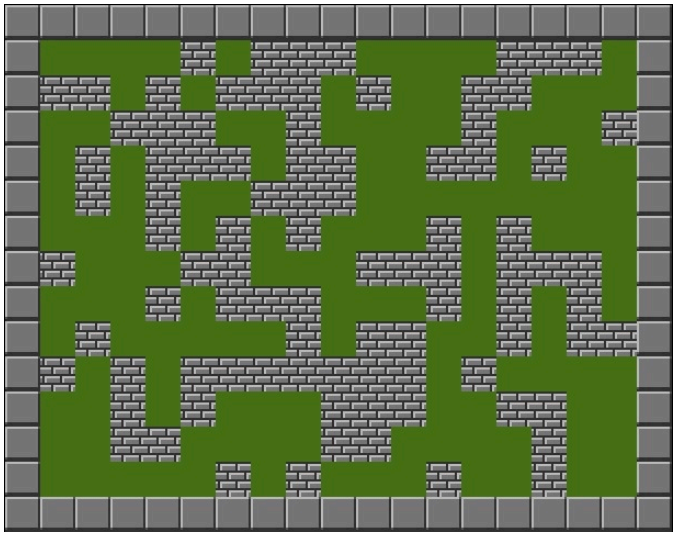

Abbildung 6: Erstellung der Welt

Die Erstellung einer spielbaren Figur erfolgt ähnlich der von Blöcken. Jedoch muss eine Spielerfigur um erheblich mehr Funktionalität erweitert werden. Zu einem muss die Figur steuerbar sein, sprich in unserem Fall auf Tastaturevents reagieren, als auch in Folge dieser Events animieren. Um zwei steuerbare Spieler zu implementieren, griffen wir auf Teile des Bespielcodes aus dem Tutorial zurück und passten diesen auf unsere Gegebenheiten an. Zur Animation der Bewegung musste der Name der Animation definiert und die dazugehörigen Koordinaten der Animationsschritte aus der Spritesmap „*player_sprites.png*" angegeben

werden. Damit die Spieler Entity auf die definierten Animationen zugreifen kann, sind diese in einer Komponente verpackt. Weiter gibt es eine Komponente, welche die Tastenevents definiert. Diese Komponenten erledigten im Prinzip immer die gleichen Aufgaben, waren jedoch durch kleine Unterschiede nicht für alle Spieler Entities anwendbar. Dadurch zeichnete sich bereits zu Beginn ein wachsender redundanter Code ab, was unter anderem zeigte, dass die anfängliche Implementierung noch nicht gut gelungen war.

8.2 Entwicklung der Spielelogik

Es kristallisierte sich heraus, dass die implementierte Steuerung der Spielfiguren wenig Flexibilität besaß. Ein anderer wichtiger Punkt war die Kollisionserkennung. Spieler dürfen beispielsweise nicht durch Blöcke laufen, aber durch andere Spieler. Die durch Crafty gelieferte Kollisionserkennung bot hier nur wenig Möglichkeit, auf spezielle Anforderungen wie wir sie hatten, genügend einzugehen.
Aus den genannten Gründen entschieden wir uns dazu, weniger vorgefertigte Komponenten wie die Kollisionserkennung, oder die Spielersteuerung von Crafty zu verwenden, um stattdessen diese Komponenten selbst zu schreiben. Dadurch musste natürlich erheblich mehr Code geschrieben werden, was aber ermöglichte dem Verhalten des Originalspiels erheblich näher zu kommen. Fortan arbeiteten wir an einer Lösung, welche es ermöglichte das Spielverhalten genauer zu definieren. Im Folgenden gehen wir auf die letztendlichen Ergebnisse dieser Arbeit ein.

8.2.1 Übergabe des Spielstandes

Das eigentliche Spiel wird aus der von Pavlina geschriebenen GUI geladen. Um dies zu ermöglichen packten wir den gesamten Code in eine Wrapperfunktion, welche durch die GUI aufgerufen werden kann. Diese neue Funktion mit dem Namen „*startGame*“ bekommt zwei Parameter:

Als ersten Parameter erhält diese das Objekt „*gameState*", also den aktuellen Spielestand. Dies war von großer Bedeutung, da durch die Realisierung des Shops die Anforderung für eine flexible Aktualisierung der Spielereigenschaften gestiegen war. Der zweite Parameter ist eine Referenz zur aufrufenden Umgebung, womit es möglich ist, aus dem Spiel eine Funktion der GUI aufzurufen. In unserem Fall, um zu signalisieren, dass die Runde zu Ende ist.

8.2.2 Übergabe der Parameter an die Spieler

Damit der Spieler mit Leben befüllt werden kann, entwickelten wir nach und nach die Konstruktorfunktion „*Gamelogic*" aus. Mit der Erzeugung eines neuen Objektes als Komponente in Crafty werden fortan alle wichtigen Events empfangen bzw. Events ausgelöst, welche den jeweiligen Spieler betreffen. Das Gute an dieser Herangehensweise ist die Möglichkeit, die „Gamelogic" als Konstruktorfunktion in JavaScript zu verpacken. Dadurch kann für jeden Spieler ein neues Objekt dieser „*Gamelogic*" angelegt werden, worauf nur dieser Zugriff hat. Der Weg zu dieser eleganten Lösung wurde erst in den letzten Wochen der Entwicklungsphase intensiv beschritten, was vermehrt mit dem Vorhaben zu tun hatte, mehr als zwei Spieler im Spiel zu unterstützen.
Die „*Gamelogic*" Konstruktorfunktion besitzt eine Funktion mit dem Namen „*gamelogic*", welche an ihr aufgerufen werden kann, was bei der Erstellung der Spieler Entities geschieht. Mit dem Aufruf werden die initialen Spielerstände an den betreffenden Spieler weitergegeben. Wie oben bereits erwähnt, wird das Objekt, welches alle wichtigen Eigenschaften für den Spieler enthält, wie z.B. Spielername, Geschwindigkeit, Anzahl der legbaren Bomben, Reichweite des Feuer, usw. übergeben. In diesem Objekt stehen des Weiteren die Tastaturbefehle, welche intern für die Steuerung des Spielers benötigt werden. Diese fünf Befehle werden für alle vier Laufrichtungen, sowie für das Legen der Bombe als ASCII Wert übergeben.

8.2.3 Tastatur Eventhandling

Durch Crafty kann über Events auf Tastatureingaben reagiert werden. Da jedoch bei der Ansteuerung von zwei Spielern in der alten Implementierung, die Tastaturbefehle den anderen Spieler beeinflussten und dieser somit immer zum Stillstand gelang, entschlossen wir uns kurzer Hand dazu eine andere Ressource zu nutzten. Die Bibliothek jQuery, welche uns aus dem Multimediaapplikationen Unterricht geläufig war, bot eine simple Möglichkeit, die Tastaturbefehle abzufangen. Das Handling ist so aufgebaut, das Tastenab und Tastenauf Events separat an alle Spieler gesendet werden. Durch die Übergabe der initialen ASCII Werte zur Steuerung der Spielfiguren ist es nun möglich, dass nur diejenigen Spieler auf die Events reagieren, welche auch auf den ASCII Wert initialisiert wurde. So muss auch keine Vorauswahl der Tastatureingaben getroffen werden. An der Gamelogic Komponente sind drei Events gebunden, was bedeute, wird ein Event an einem Spieler *„getriggert*", reagiert dieser darauf. Im Fall der Tasten Events können Spieler somit auf *„keyup*" und *„keydown*" Events reagieren.

8.2.4 Spielerbewegung

Durch die *„keydown*" Events wird in der *„Gamelogic*" ein Objekt mit den booleschen Variablen *„left*", *„right*", *„up*" und *„down*" auf wahr gesetzt. Gegenteiliges passiert mit dem Event *„keyup*", welches die jeweilige Richtung auf falsch setzt. Wie oben erwähnt sind drei Events an die Komponente gebunden. Das dritte Event ist das *„enterframe*" Event. Dieses Event wird durch Crafty für jeden neuen Frame aufgerufen. In der Callback Funktion des Events wird das Objekt, welches die aktuelle gedrückte Richtung enthält, jede Bildwiederholung neu abgefragt. Ist eine der Richtungen auf wahr gesetzt, werden folgende Punkte abgehandelt:

1. Die Animation zur Bewegung gestartet
2. Geprüft, ob sich vor dem Spieler etwas undurchdringbares befindet

3. Wenn nein, wird die Position des Spielers entsprechend verändert

Das setzten der neuen Position geschieht dadurch, dass auf die aktuelle Position die Geschwindigkeitsvariable „*speed*" in x- oder y-Richtung addiert bzw. subtrahiert wird.

8.2.5 Umpositionierung der Spieler

Im Original Masterblaster ist das Laufverhalten von Spielern an eine Art Gitter gebunden. Dadurch läuft der Spieler immer auf einer vordefinierten Bahn, welche er bei Richtungswechsel verlassen kann, jedoch geschmeidig wieder auf die naheliegendste Bahn zurück geschoben wird. Wichtig für die Implementierung dieses Verhaltens war somit die Ermittlung, wie entschieden wird, dass der Spieler auf der ihm naheliegendsten Bahn läuft. Die Lösung hierfür waren Modulo Operationen auf die Position des Spielers. Durch den Rest konnte entschieden werden, ob und in welche Richtung der Spieler geschoben werden musste. Um den Effekt des geschmeidigen Laufens zu erzielen, wurde je nach Restergebnis der Modulo Operationen „1" auf x bzw. y addiert oder subtrahiert. Dieses Laufverhalten macht sich besonders bemerkbar, wenn sich die Spielerfigur vor einem Hindernis befindet. Bei der Berechnung wird die Mitte des Spielersprites betrachtet, womit entschieden wird ob sich der Spieler bei Tastendruck gegen das Hindernis vorbei bewegen soll. Abbildung 3 soll diesen Vorgang verdeutlichen.

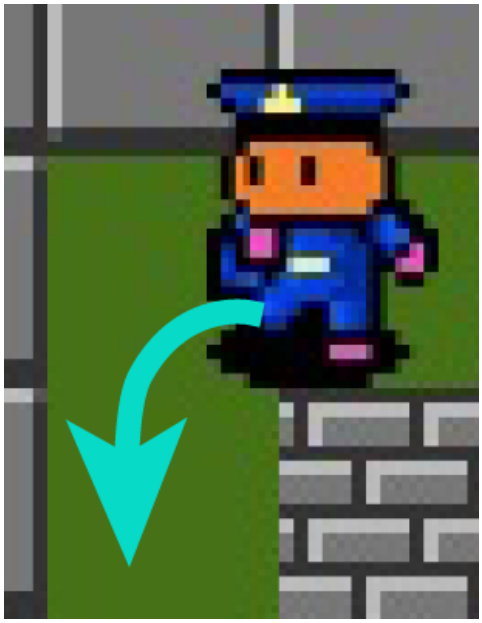

Abbildung 7: Bewegung

8.2.6 Animation der Spieler

Die Animation der Spielerfiguren geschieht in Folge des „*enterframe*" Event Aufrufs. Wie erwähnt wird in deren Callback Funktion die Bewegungsrichtung des Spielers abgefragt. Trifft eine Richtung zu, wird die jeweilige Animation gestartet. Da der Spieler durch das Goody „Invincible" seine Animation auch grundlegend ändern kann, ist es wichtig andere Sprites zur Animation angeben zu können. Wir entschieden uns dazu, die Definition der Animationen somit in weiteren Komponenten auszulagern. Dies hat den Vorteil, dass die Komponenten je nach Gebrauch an der Spieler Entity hinzugefügt und auch wieder entfernt werden können. Um diese Komponenten jedoch für alle Spieler nutzbar zu machen, wird an die Komponente der Spielername übergeben, womit diese dann mit einer weiteren Funktion „*getPlayerCord*" die Koordinaten aus der Spritemap für den jeweiligen Spieler geliefert bekommt. Probleme bereitete das Beenden der Animation. Erfolgt das Keyup Event unterbrach die Animation mit dem letzten Sprite der Animationskette. Dies hatte zur Folge das der Spieler manchmal einen bewegten Sprite angezeigt hat, obwohl die Spielfigur eigentlich still stand. Die Lösung war es, innerhalb des Keyup Events die

laufende Animation zu stoppen und darauf hin eine Animation für eine stehende Spielerfigur zu starten. Zur Veranschaulichung ist in Abbildung 4 die benötigte Anordnung der Spielersprites gezeigt.

Abbildung 8: Spielersprite

8.2.7 Solid Test

Für die Spieler sind zwei Gegenstände undurchdringbar. Diese Beiden sind Wände und Blöcke. Zur Definition wo sich eine solide Einheit befindet, wird zur Initiierung der Entities „brick" und „wall" gleichzeitig ein Array namens „*brick_array*" mit Number Werten befüllt, welche die jeweiligen Einheiten repräsentieren. Im Falle der Blöcke gibt es mehrere Zustände, welche hier berücksichtigt werden mussten. Um auf diese Entities später wieder zugreifen zu können, wurden die Entities ebenfalls in einem Array angelegt. Um nun zu überprüfen, ob sich vor der Spielerfigur eine solide Einheit befindet, wird die Position des Spielers vor dem Verändern durch vier Funktionen überprüft, jede für die entsprechende Laufrichtung. Leider mussten wir vier verschiedene Funktionen schreiben, da jede Richtung ihre besonderen Feinheiten mit sich brachte. Diese Funktionen überprüfen, ob die Position des Spielers relativ mit einer Einheit im oben erwähntem Array übereinstimmt. Befindet sich also eine „2" im Array, teilt die Funktion der Gamelogic mit, dass der Spieler hier nicht weiterlaufen kann.

8.2.8 Bomben

Die zentrale Fähigkeit, welche der Spieler von Anfang an besitzt, ist das Legen von Bomben. Der Spieler kann zu Beginn lediglich eine Bombe platzieren, wenn diese dann explodiert kann er eine weitere legen. Diese Anzahl der legbaren Bomben kann durch das *„bombup*" Goody erhöht werden. Um nun einen Bombe zu legen, muss zu Beginn nach Erhalten des Tastendruck überprüft werden, ob die Anzahl der gelegten Bomben kleiner als die Anzahl der maximal legbaren Bomben des Spielers ist. Ist das gegeben, wird überprüft, ob sich an der Stelle, wo eine Bombe platziert werden soll, sich bereits eine Bombe befindet. Die Abfrage geschieht hier auch wieder über ein Array, in dem eine Bombe durch eine Zahl repräsentiert wird. Wenn nun diese Abfragen erfolgreich durchlaufen werden, wird ein neues „bomb" Entity erstellt, welches zugleich in ein Bombenarray geschrieben wird. Dies hat den Grund, dass das Feuer einer Bombe eine andere noch tickende Bombe zur Detonation bringen kann. Um also dies zu realisieren ist ein Bombenarray mit allen „bomb" Entities von Nöten, um diese referenzieren zu können. Ist die Entity gesetzt, wird diese animiert. An dem bomb Entity ist des Weiteren das Event „explode" gebunden. Dadurch ist es möglich die Bombe gezielt zu zerstören (Detonation durch Explosion benachbarter Bomben). Die Zerstörung der Bombe wird im Normalfall nach einer Zeit von drei Sekunden ausgeführt. Diese Zeit kann zusätzlich durch ein entsprechendes Goody verringert werden. Wird das Event „explode" der Bombe aufgerufen, sorgt eine weitere Funktion für das Setzten der Feuer Entities in die entsprechenden vier Richtungen.

8.2.9 Feuer

Das Feuer wird in der Funktion *„bombExplosion*" erstellt. Diese erhält die x- sowie y-Position der Bombe, sowie eine Referenz an den Aufrufenden Spieler. Die Explosion einer Bombe verursacht im Normalfall einen Feuerradius von 2 Einheiten. Das Besondere ist die Ausbreitung des Feuers, denn diese geschieht zeitversetzt. Anfänglich versuchten wir die

Ausbreitung über eine for-Schleife, welche bei jedem Durchlauf kurz pausieren werden sollte, zu realisieren. Dadurch kam es jedoch zu merkwürdigen Fehlern. Im Gespräch konnte uns Prof. Dr. Hopf weiterhelfen indem er uns nahe legte die Feuer Entities neue Feuer Entities aufrufen zu lassen. Für diese mussten dann lediglich einen Richtungsvektor angegeben werden, um zu bestimmen in welche Richtung diese weitere Feuer Entities setzten sollen und wie viele Male dieser Vorgang noch wiederholt werden sollte.
Wurde ein Feuer gesetzt, handelt deren Komponente „SetFire" noch einige wichtige Abfragen ab. Zu aller erst wird überprüft, ob sich an der Stelle etwas befindet, was durch das Feuer zerstört wird. Ist dies der Fall wird die Ausbreitung des Feuers nicht weiter geführt. Um herauszufinden, ob sich ein Spieler an der vom Feuer getroffenen Stelle befindet, wird über alle Spieler Entities gelooped und deren Position mit der des Feuers verglichen. Somit ist auch sicher gestellt, dass mehrere Spieler auf dem gleichen Feld durch das Feuer getroffen werden, was in früheren Versionen des Remakes nicht funktioniert hat. Wurde also ein Spieler getroffen, wird die Sterbeposition des Spielers auf die des Feuers gesetzt. Dies hat den Sinn, dass die Sterbeanimation auch da angezeigt wird, wo der Spieler getroffen wurde.

8.2.10 Goodys

Goodys sind Gegenstände, welche beim Sprengen von Blöcken erscheinen können. Diese können von den Spielern aufgenommen werden, und deren Attribute, Fähigkeiten bzw. Erscheinung ändern. Im Spiel gibt es 10 Goodys, die der Spieler aufnehmen kann. Neben diesen Goodys gibt es noch einen weiteren Gegenstand, den Totenkopf. Dieser führt beim übertreten zum Tod der Spielerfigur. Von den zehn Goodys gelang es uns sechs zu implementieren. Aufgrund der begrenzten Zeit waren leider nicht mehr Goody Implementierungen möglich.
Die Entwicklung der Goodys begann mit den für uns am wichtigsten erscheinenden Goodys. Das waren unter anderem das „bomb up", „speed

up" sowie „fire up" Goody. Nachdem diese Goodys implementiert waren erstellten wir eine priorisierte Liste mit den noch offenen Goodys. Wichtig war vor allem heraus zu finden, welche Goodys schneller zu implementieren sind.
Das Setzten der Goodys geschieht nach der Zerstörung eines Blocks. Wenn dieser vom Feuerradius einer Bombe getroffen wurde wird eine neue Entities erstellt, welche die Animation eines verbrennenden Blocks darstellt. Bevor diese Entity zerstört wird, ruft diese noch die Funktion „*rollTheDiceForGoody*" auf. Diese Funktion entscheidet durch das Ziehen einer zufälligen Zahl, ob ein Goody generiert wird, oder nicht. Die Goodys im Originalspiel haben Auftrittswahrscheinlichkeiten welche wir durch den Verzicht der 4 Goodys so genau wie möglich eingebunden haben. Wird nun durch die Zufallszahl entschieden, dass ein Goody erstellt werden soll, wird die Funktion „generateGoody" aufgerufen. Um die Goodys im Verlauf des Spiels erkennen und ansprechen zu können, werden diese in das „brick_array", sowie in das „goody_array" für die Speicherung der Entities geschrieben.
Der zweite wichtige Teil der Implementierung der Goodys ist die Erkennung, ob ein Spieler über einen Gegenstand läuft. Hierfür bedienten wir uns der Solid-Prüffunktionen. Läuft eine Spielfigur in eine Richtung, wird überprüft, ob sich ein Goody auf dessen Position befindet. Dazu wird in den Solid-Prüffunktionen die Funktion „checkForGoodys" aufgerufen. Erkennt diese über das „brick_array" einen Gegenstand, entscheidet diese welche Einflüsse auf den Spieler genommen werden müssen und entfernt das Goody danach.

8.2.11 Shrinking

Das Shrinking ist ein Mechanismus des Spiels, lang andauernde Runden zu beenden. Dieser tritt nach 90 Sekunden auf und bewirkt, dass das Spielfeld nach und nach verkleinert wird. Dazu werden neue Wände in einer Spirale nach innen gesetzt.
Um diesen Mechanismus zu realisieren, entwickelten wir eine Komponente, welche zur Funktionsweise der Feuerkomponente gewisse Ähnlichkeit

aufweist. Zu Beginn wird eine Entity erstellt, welche zeitverzögert eine weitere erstellt usw. Ist eine Einheit an der Wand angelegt, muss diese entsprechend die Richtung ändern. Wie viele Wände noch gesetzt werden müssen, legt die „*wallsLeft*" Variable fest. Befindet sich ein Spieler an der Position an der eine neue Wand entsteht, wird dieser Spieler entsprechend zerstört.
Der Aufruf des Shrinking Mechanismus geschieht über die Standard JavaScript Funktion „*setTimeOut*". Dadurch dass wir keinen Weg gefunden haben eine Szene in Crafty zu zerstört, mussten einige Mechanismen eingebaut werden, welche es erlauben das Menü nach einer Runde in den Vordergrund zu befördern und bei erneutem Beginn einer Runde, Crafty ohne Seiteneffekte von neuem zu initialisieren. Besonders das Setzten einer Timeout Funktion, oder das Abspielen von Sounds machte diverse Schwierigkeiten, welche aber rechtzeitig behoben werden konnten. Neben dem Shrinking wird auch noch der Hintergrund um das Spielfeld animiert, welcher alternierend zwischen schwarz und rot wechselt.

8.2.12 Ende der Runde

Ist nur noch eine Spielfigur am Leben hat dieser Spieler gewonnen. Um dies festzustellen muss nach jedem „Tod" eines Spieler überprüft werden, wie viele Spieler noch leben. Falls nur noch einer am Leben ist, muss die Runde beendet werden. Hier kommt die Referenz auf die GUI Umgebung zum Einsatz, denn an dieser wird die Funktion „*gameFinished*" aufgerufen. Diese übergibt zusätzlich den modifizierten „*gameState*" zurück, worin am Gewinnerspieler ein Tag gesetzt wird, sowie das gesammelte Geld der jeweiligen Spieler vermerkt wird, um im Nachhinein damit im Shop Goodys für zukünftige Runden kaufen zu können.

9. Konfigurationsmenü und Szenen

Eine wichtige Aufgabe, die wir hatten, war die Konfigurationsmöglichkeiten des Spiels zu untersuchen und eine passende Architektur zu entwickeln. Nutzer aller Art von Videospielen sind es gewöhnt, dass man seine

persönlichen Präferenzen speichern kann, damit man sie nicht jedes Mal erneut eingeben muss. Auch ist es bei vielen Spielen möglich, eigenen Content zu dem Spiel hinzuzufügen, wie zum Beispiel eigene Spielfiguren. Da unser Spiel in einem Browser laufen sollte, wäre das Speichern von Konfiguration in Form von Cookies und das Laden von externen Sprite-Dateien möglich.
Unsere Aufgabe bestand darin uns darüber zu informieren, wie wir unser Programm aufbauen müssen, damit später eine Konfiguration via Cookies und externe Konfigurationsdateien möglich ist. Es hat sich schnell rausgestellt, dass sich JSON Objekte gut dafür eignen.
So entstand die Basis konfigurationsvariable config, die als JSON Objekt aufgebaut ist. Als dieses Konzept stand, war ziemlich deutlich wie der Rest der Architektur aufzubauen ist. Es bestand eine index.html, aus der alle anderen Funktionen aufgerufen werden, GUI und Spiellogik als externe Dateien.

9.1 GUI erstellen

Eine weitere Hauptaufgabe bestand darin die GUI mit allen Szenen und Abläufen so zu definieren, dass die Spielelogik mit allen nötigen Parametern aufgerufen werden kann.
Die GUI besteht aus Szenen, die per Tastendruck der Reihe nach gewechselt werden. Man kann Spieler- und Gewinnanzahl konfigurieren.
Wir haben versucht, uns so gut wie möglich an den Originalszenen zu halten, jedoch auch ein paar Änderungen und Anpassungen vorgenommen. Wir haben eine Schriftart gefunden, die sich an der Original-Amigaschriftart nähert, die wir in allen Szenen eingebunden haben um das Feeling beizubehalten. Anbei einen Überblick aller Szenen mit kurzer Beschreibung:

Startscreen - es wird das Logo des Spiels angezeigt. Bei uns ist das das Logo, das Michael für uns entwickelt hat.

Credits - ähnlich wie im Originalspiel haben wir die Credits so konstruiert, dass sie unsere Namen, das Originalspiel und ein kleines "Dankeschön" an dem Schöpfer des Spiels enthalten.

Config Menu - Das ist das Hauptkonfigurationsmenu und von diesem Menu hängt es ab wie viele Spieler spielen werden und wie viele Gewinne man braucht um den Sieger zu sein.

Um dem Originalspiel treu zu bleiben haben wir alle Konfigurationsmöglichkeiten beibehalten, es sind aber momentan nur einige veränderbar.

Mit den Tasten nach oben und unten wechselt man zwischen den unterschiedlichen Konfigurationsmöglichkeiten und mit den Tasten für links und rechts - wird der Wert erniedrigt oder erhört.

Wir haben eine Verbesserung des Konfigurationsmenus vorgenommen. Wir fanden, dass es eine gute Idee wäre zwischen Maximal- und Minimalwert nur mit einem Tastendruck zu wechseln, so dass wenn man die maximale Anzahl an Spieler (z.B.) erreicht hat, wieder mit nur einem Tastendruck nach rechts die minimale Spieleranzahl erreicht. Dies haben wir eingebaut, da man aus anderen Spielen gewöhnt ist ein solches Verhalten zu erwarten und das wiederholte Drücken einer Taste ohne Rückmeldung als störend empfunden wird.

Für die Werte YES/NO, ON/OFF ist ein Toggle eingebaut, so dass auch dies einfacher zu ändern ist.

Player Choice - Gleich nach der Konfiguration kommt die Möglichkeit für jeden Spieler eine Figur auszuwählen, ohne dass eine Figur 2x gleichzeitig ausgewählt werden darf. Da es ursprünglich geplant war nur 2 Spieler zu implementieren, dachten wir uns dass es etwas langweilig wäre, wenn man immer mit den gleichen 2 Spielerfiguren spielt. Unsere Idee war eine Art Auswahlmenu einzubauen. Dieses kleine Menu war nicht geplant und stand nicht als Ziel bei den Kann Kriterien, sondern ist komplett als "Goodie" gedacht. Mit ein paar Anpassungen der Variable zum Speichern von Spielerinformationen, war dies nun auch erledigt.

Ab jetzt ist jeder Spieler nicht nur per Spielernummer identifizierbar, sondern hat einen Namen - der Name seiner Spielfigur.
Alle diese Szenen werden nur einmal am Anfang, wenn das Spiel gestartet wird, ausgeführt. Alle folgenden Szenen werden wiederholt aufgerufen, so oft bis einer von den Spielern die benötigte Anzahl an Gewinne erreicht hat.
Shop - Wenn während des Spiels ein Spieler Münzen aufgesammelt hat, erscheint bei dem nächsten Loop der Shop, in dem der Spieler einkaufen darf. Ein Spieler darf solange er Geld hat einkaufen oder bis er auf "Exit" gedrückt hat.
Wir haben den Shop bei der Implementierung etwas erweitert und angepasst. Der Button EXIT wird farblich markiert, was eine bessere Übersicht darstellt. Und es ist zu jedem Zeitpunkt sichtbar welcher Spieler grade den Shop betreten hat - im Originalspiel stand nur eine Nummer, bei uns ist jeder Spieler durch seinen Namen identifizierbar.
Countdown - nachdem alle Spieler ihre Shop Auswahl bestätigt haben startet einen kleiner Countdown - 3,2,1 - und das Spiel kann beginnen!
Spiel - hier wird die Funktion startgame() von game.js aufgerufen, in der das Spiel stattfindet.
Hall of Fame - nachdem das Spiel zu Ende ist, wird Objekt zurückgegeben, das alle Spielerinformationen enthält - ihre Gewinne und das gesammelte Geld. Die "Halle der Gewinner" ist eine rein informative Szene, die nur als Anzeige der vorhandenen Pokale dient.

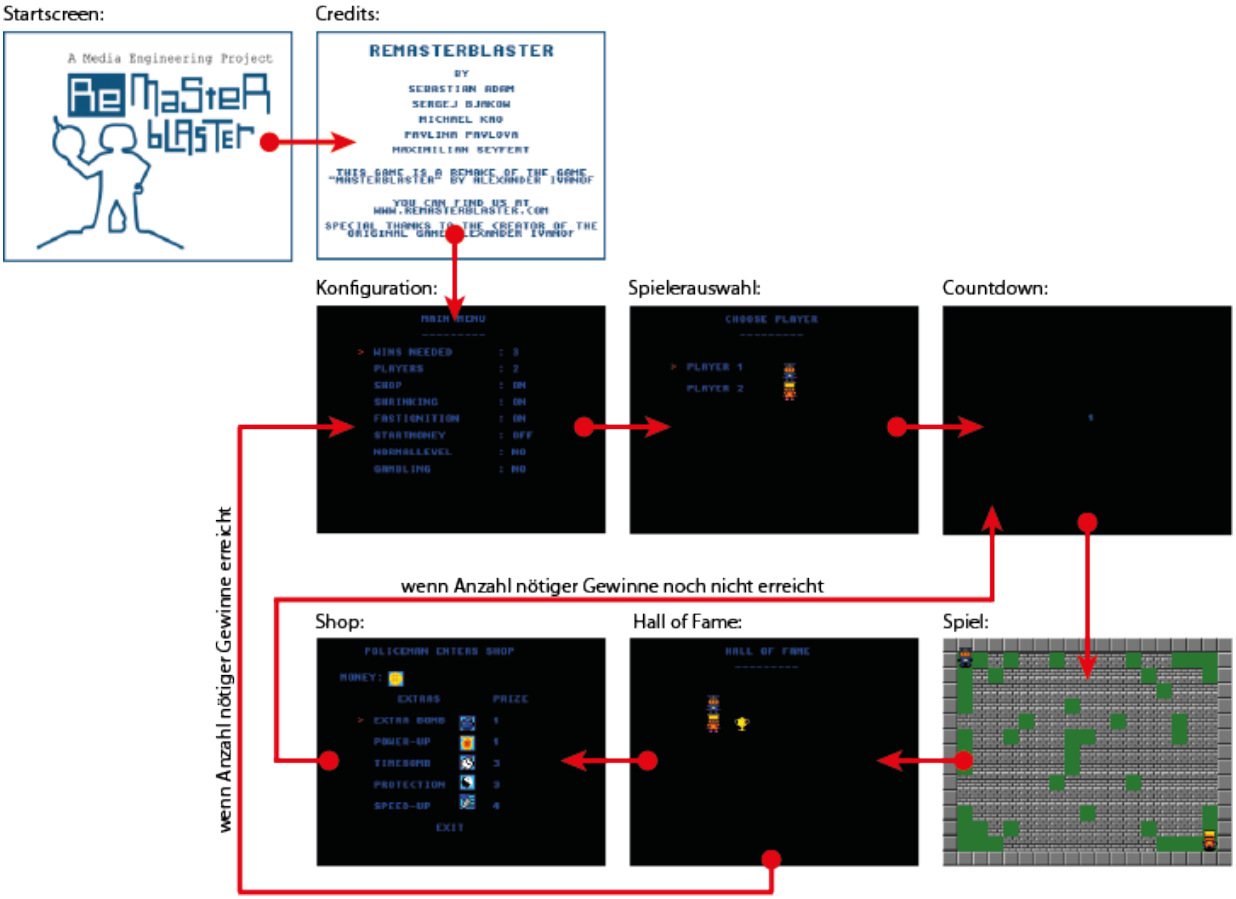

Abbildung 9: Ablauf der einzelnen Szenen

Anfangs hatten wir alle Szenen einzeln definiert, um schneller und einfacher ihre Funktionalität testen zu können. Bei dem Zusammenfügen aller Szenen und die erste Verbindung zwischen Spiel und GUI ist das erste richtig große Problem aufgetreten - unsere Versionen waren nicht kompatibel.

Die Spielelogik von Michael und die GUI von Pavlina hatten unterschiedliche Crafty Versionen verwendet, was dazu geführt hat, dass alle Key-Anbindungen, Crafty Animationen usw. nicht dargestellt wurden.

Da die Spielelogik etwas mehr Codezeilen besitzt als die GUI hat Pavlina sich dazu bereit erklärt alle GUI Funktionen umzuschreiben und anzupassen.

Ursprünglich hatten wir uns drauf geeinigt, dass wir soweit wie möglich alles mit Crafty versuchen zu implementieren. Dies hat zu Problemen geführt. Da Crafty eine Sprite Engine ist und darauf aufgelegt ist, dass alle

Animationen und dynamische Inhalte in einer Main Scene passieren, so dass die Scene nicht neuaufgebaut werden muss.
Eine Animation ist nichts anderes als wiederholtes Zeichnen eines Objekts mit unterschiedlichen Parametern. Um die Aktualisierungen des Konfigurationsmenus darstellen zu können, müsste bei jedem Ändern die Szene neu gezeichnet werden. Crafty implementiert und zeigt seine Szenen intern auf eine aufwändige Art und Weise an, was dazu geführt hat, dass der Übergang nicht flüssig war und die es sichtbar war, wenn eine Szene neugeladen wird.
Deswegen war es naheliegend, dass wir Crafty nur für die Funktionen nehmen, die auch von der Engine vorgesehen sind und für alles andere reine JavaScript oder jQuery Funktionen.
Dies bedeutete für uns, dass wir den Code erneut schreiben und alle Szenen von Crafty in JS Canvas umdefinieren mussten.
Als die Verbindung zwischen GUI und Spiel stand war es einfach den kompletten Ablauf zu testen. Erst dann sind einige Logikfehler aufgetreten. Die GUI übergibt der Spiellogik eine Objektvariable und bekommt als Rückgabe auch ein Objekt zurück. Das bedeutet, dass es immer in der gleichen Variable geschrieben wurde, da in JavaScript Objekte und Arrays per Referenz und nicht per Value übergeben werden. Dieses Problem war schnell gelöst indem wir vor dem Aufruf die Variable einfach kopiert haben. Es sind oft so kleine Probleme aufgetreten, die aber schnell behoben waren, wenn man die Theorie dazu gelernt hat.

9.2 Parametrisieren und Final Touches

Als das Spiel voll funktionsfähig war, haben wir die Möglichkeit gefunden mehrere Game-Controller anzubinden. Wir haben die Spielersteuerung in der Konfiguration parametrisiert und für alle Spielaktionen immer die Steuerung des ersten Spielers übernommen, wie in dem Originalspiel.

10. Versionsverwaltung mit github

Bereits ganz zu Beginn des Projekts empfahlen uns Prof. Dr. Matthias Hopf und Prof. Dr. Stefan Röttger ein Versionierungssystem für die Umsetzung des Projekts zu verwenden. Hierfür wurde uns dann github[12] von den beiden Projektbetreuern nahe gelegt. Github ist ein auf git basierender web-basierter Hostingdienst für Softwareentwicklungsprojekte. Git selbst stellt wiederum das Versionsverwaltungs-System dar. Auf github werden sehr viele bekannte Open-Source Projekte gehostet, darunter beispielsweise auch jQuery, PHP, Ruby on Rails, etc.
Da das Projekt ReMasterBlaster ebenfalls als Open-Source Projekt angelegt wurde bot sich github als Hosting für unseren Quellcode und andere Dokumente natürlich ebenfalls an. Am Anfang war die Verwendung von github, mit der sich dann natürlich jeder auseinander setzen sollte, erst einmal sehr schwierig. Jeder musste sich selbst mit den Begrifflichkeiten „Repository", „commit" und „push" erst vertraut machen. Das Repository stellt im Wesentlichen den eigentlichen Projektordner dar in dem alle Dateien gespeichert werden. Ein „Commit" ist eine Beschreibung einer Änderung, die man an einer vorhandenen Datei vorgenommen hat. Mit einem „Push" synchronisiert man den lokalen Ordner auf seiner Festplatte mit dem Online-Speicherplatz auf github. Es gab zwar einige Tutorials und Dokumentationen bezüglich github, jedoch dauerte es dennoch eine gewisse Zeit bis sich das Team in die Verwendung und den Workflow dieser Entwicklungsmethode eingefunden hatte.

[12] http://www.github.com

11. Soll – Ist Analyse

Die Soll-Ist Analyse des Projekts ReMasterBlaster stellt sich wie folgt dar:

Muss – Kriterien	▪ Spielbares Produkt von ReMasterBlaster ▪ Zwei steuerbare Spieler mittels Tastatureingabe ▪ Spieler muss Bomben legen und Wände sprengen können ▪ Gegenspieler wird bei einem Treffer von Spielfeld entfernt	
Soll – Kriterien	▪ Einbindung der Extras im Spiel (Goodies) ▪ Konfigurationsmenü zu Beginn des Spiels ▪ Szenenabfolge mit Credits, Config, Countdown, Spiel ▪ Hall of Fame der Gewinner einer Runde	
Kann – Kriterien	▪ Einbindung von mehreren spielbaren Charakteren ▪ zusätzliche Eingabegeräte verwendbar machen ▪ Shop um Extras nach einer Spielrunde kaufen zu können ▪ Sprite - Auswahl für die einzelnen Spielcharaktere	
Kann – Kriterien	• ALLE Goodies wie im Originalspiel implementieren	

In dieser Analyse zeigt sich, dass alle Muss- und Soll-Kriterien vollständig erfüllt werden konnten. Somit war der „Grunderfolg" des Projekts bereits gesichert. Jedoch stellten wir bereits während der Umsetzung fest, dass wir noch „Reserven" für zusätzliche Anforderungen bzw. so genannte „Kann-Kriterien" hatten. Es wurden zwar bereits im Pflichtenheft solche festgelegt, jedoch kamen dann im weiteren Projektverlauf noch weitere „on the fly" hinzu. Das waren natürlich immer die besten Momente bei einer Teambesprechung, wo die Ideen gerade so aus den Teilnehmern „raus sprudelten". Hier konnte man feststellen, dass das Team begeistert bei der Sache ist auch persönliches Engagement und Hingabe für die „gemeinsame

Sache" zeigte. Von diesen selbstgesteckten Kann-Zielen konnten wir dann auch den größten Teil gemeinsam bewältigen. Einzig und allein die Implementierung von wirklich allen Goodies, wie sie auch im Originalspiel vorhanden sind, blieb uns mangels vorhandener Zeit verwehrt. Doch wir sind uns sicher, dass auch die restlichen Goodies mit ausreichend vorhandener Zeit definitiv möglich gewesen wären was die technische und organisatorische Umsetzung betrifft. Somit sind wir mit der Soll – Ist Analyse des Projekts „ReMasterBlaster" mehr als zufrieden, da wir anfangs noch davon ausgegangen sind nur einen kleinen Teil dieser Ziele wirklich erreichen zu können.

12. Resümee

Abschließend können wir sagen, dass uns das Projekt über die gesamte Dauer sehr große Freude bereitet hat. Vor Allem die Resonanz sowohl von Professoren- als auch von Studentenseite bei der Abschlusspräsentation war durchweg positiv, was uns natürlich ganz besonders gefreut hat! Wir möchten uns auch auf diesem Wege noch einmal bei den beiden Projektbetreuern Prof. Dr. Stefan Röttger und Prof. Dr. Matthias Hopf für ihre Unterstützung bedanken.

Abbildungsverzeichnis