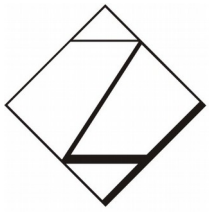

Westsächsische Hochschule Zwickau
University of Applied Sciences

Bachelorarbeit

zur Erlangung des Grades „Bachelor of Science“ im Fach Informatik

Performanceanalyse und -optimierung von Browseranwendungen für mobile Endgeräte am Beispiel eines HTML5 basierenden Browserspiels

Benkenstein, Martin

geboren am 07.07.1984 in Querfurt

Matrikelnummer 27928

Studiengang Informatik

Studienrichtung Systementwicklung

Seminargruppennummer 092079/SEW

Westsächsische Hochschule Zwickau

Fakultät Physikalische Technik / Informatik

Fachgruppe Informatik

Betreuer, Einrichtung

Prof. Dr. W. Golubski, WH Zwickau (FH)

Dipl. Designer Constantin Andiel, Coneckt GbR

Abgabetermin: 25.02.2015

Autorenreferat

HTML5 und mobile Geräte wie Smartphones und Tablets sind zwei Bereiche, welche in den vergangenen Jahren eine deutliche Weiterentwicklung erfahren haben. Browserentwickler stellen ihre Software auch für mobile Betriebbsysteme zur Verfügung und dank der wachsenden Möglichkeiten aktueller Webtechnologie wird es zusehens attraktiver, Software nach dem Motto "Write once, run anywhere" auf Webbasis zu entwickeln und damit viele Hard- und Softwareplattformen zu erreichen.

Die Andersartigkeit mobilen Geräte verglichen mit PCs, gekennzeichnet beispielsweise durch Hardwareausstattung und -leistung, Bildschirmgröße und Touchbedienung, erzeugt jedoch neue Problemfelder. Diese Arbeit beschäftig sich exemplarisch anhand des HTML5-Spiels "Letti ABC" mit dem Aspekt der vergleichweise geringen Hardwareleistung und damit verbundenen Performance-einschränkung. Es werden Analysemethoden für Webanwendungen vorgestellt und Lösungen unter anderem für die Bereiche Renderperformance, Audio und Speichernutzung aufgezeigt.

Im Ergebnis dieser Arbeit ist das Spiel sowohl auf PCs als auch auf mobilen Geräten gleichermaßen spielbar.

Betreuende Einrichtung

Das Unternehmen "coneckt GbR" ist in Reichenbach ansässig und wurde im April 2012 mit dem Ziel gegründet Kinder und Jugendliche für Produkte und Kommunikationen zu begeistern, die Medienkompetenz vermitteln und helfen, den richtigen Umgang mit den neuen Medien zu finden. Namensgebend sind ihre Gründer Constantin Andiel, sowie Annett Wohlfahrth-Behnecke und Pascal Weidenmüller von Agentur Eckpunkt, sie haben sich zusammengeschlossen um das durch die Initiative "Ein Netz für Kinder"[1] geförderte Projekt "Letti ABC" zu realisieren.

Danksagung

Mein Dank gilt meinen Freunden Michael Klenner und Stefanie Schmidt, die mich durch vielseitige Hilfe und Zeitaufwand unterstützt haben, außerdem natürlich der coneckt GbR mit allen Beteiligten, die mir diese Arbeit überhaupt erst ermöglichten und mir in stressigen Situationen den Rücken freigehalten haben.

Weiterhin gebührt großer Dank meiner Cousine Ramona Benkenstein und ihrem Freund Christian Weise für ein umfangreiches Lektorat und die vielen gestalterischen Hinweise, und meinen Eltern, auf deren Unterstützung ich mich immer verlassen kann.

Außerdem möchte ich mich bei der Westsächsischen Hochschule, insbesondere bei Frau Prof. Dr. Häber und Herrn Prof. Dr. Golubski für alle Bemühungen bedanken, mir trotz der zeitlich knappen Umstände so unkompliziert das Schreiben dieser Arbeit ermöglicht zu haben.

Nicht zuletzt bedanke ich mich auch bei allen nicht namentlich genannten Personen, die mich in und meine Launen in dieser anstrengenden Zeit ertragen haben.

1 http://enfk.de/

Inhaltsverzeichnis

Abkürzungsverzeichnis

API	*Application Programming Interface*, Teil einer Software zur Anbindung anderer Programme
CPU	*Central Processing Unit*, der Hauptprozessor eines Computers
CSS	*Cascading Style Sheets*, zentrale Gestaltungssprache für HTML-Dokumente
DOM	*Document Object Model*, Schnittstellenspezifikation für den Zugriff auf die Baumstruktur von HTML-Dokumenten
FPS	*Frames per Second*, Maßeinheit für die Bildwiederholungsfrequenz
GPU	*Graphics Processing Unit*, ein auf grafische Rechenoperationen spezialisierter Prozessor
HTML	*Hypertext Markup Language*, Auszeichnungssprache zur semantischen Strukturierung digitaler Dokumente
JS	*Javascript*, ursprünglich für HTML-Dokumente entwickelte Skriptsprache
W3C	*World Wide Web Consortium*, das Gremium zur Standardisierung von Webtechnologie

Abbildungsverzeichnis

Tabellenverzeichnis

Anlagenverzeichnis

1 Einleitung

Der nach langer Entwicklungszeit im Oktober 2014 vollendete Webstandard HTML5 erfreut sich gemeinsam mit weiteren parallel entwickelten Webtechnologien immer größerer Beliebtheit. Zusammen ermöglichen sie mittlerweile eine breite Palette an Funktionalität, die über das Darstellen von Text und Bildern auf Webseiten weit hinausgeht. Multimedia-Funktionen wie Audio und Video sowie eine wachsende Viefalt an Javascript-Schnittstellen positionieren Web-Anwendungen immer mehr als Alternative zu nativer Software. Ein großer Vorteil gegenüber dieser ist eine betriebssystem- und plattformübergreifende Unterstützung, welche bis auf einen aktuellen Browser keine besonderen Voraussetzungen mit sich bringt. Der von Sun Microsystems (heute Oracle) geprägte Slogan "Write once, run anywhere" für ihre Plattform Java findet in diesem Bereich ebenfalls seine Umsetzung.

Zeitgleich gewinnen mobile Geräte in Form von Smartphones und Tablets immer weiter an Bedeutung. Deren sowohl hard- als auch softwaretechnische Entwicklung ist zwar gar nicht so neu, hat aber seit der Einführung des Apple iPhone 2007 einen deutlichen Schub erfahren. Waren noch vor wenigen Jahren derartige Geräte teuer und wenig verbreitet, so sind sie heutzutage nicht mehr aus Geschäfts- und Privatbereich wegzudenken.

Die Browserentwickler haben das Potential dieser Geräte erkannt und stellen entsprechend angepasste Mobilversionen ihrer Software zur Verfügung, die die Lücken zu ihrem PC-Pendant zusehends schließen. Ebenso wird von ihnen immer mehr HTML5- und Webtechnologie implementiert. So ist es folgerichtig, dass dem Webbereich der Softwareentwicklung eine wachsende Bedeutung zukommt, da sich mit verhältnismäßig geringem Aufwand eine Vielzahl von Anwendern und verschiedenen Endgeräten erreichen lässt, und sich durch die zusehens breitere technische Aufstellung von Webtechnologie immer mehr Probleme effizient lösen lassen.

Dies stellt jedoch sowohl die Browser- als auch die Webentwickler vor neue Herausforderungen, denn mobile Geräte unterscheiden sich teils signifikant von PCs. Eigenheiten wie eine primär durch einen Touchscreen realisierte Bedienung, deutlich geringere Bildschirmgrößen und nicht zuletzt die - verglichen mit einem PC - eher leistungsschwache Hardware müssen berücksichtigt werden und benötigen

Anpassungen und Optimierungen für solche Gegebenheiten.

1.1 Problem und Zielstellung

Beispielgebend für eine solche Herausforderung aus Sicht der Webentwicklung ist in dieser Arbeit das Spiel "Letti ABC". Es handelt sich hierbei um eine Browseranwendung, welche Kindern im Grundschulalter spielerisch die Herkunft der lateinischen Buchstaben vermitteln soll und intensiven Gebrauch von aktueller Webtechnologie macht. Auf einem durchschnittlichen PC mit DualCore-Prozessor, OnBoard-Grafikkarte mit 3D-Beschleunigung und 4 GB Arbeitsspeicher erreicht diese Browserspiel eine gute Performance und damit Spielbarkeit. Optimiert wurde es auf der PC-Plattform für Google Chrome und ist damit dank der Verfügbarkeit dieses Browser für Windows, MacOS und auch Linux auf all diesen Betriebssystemen spielbar.

Eine weitere Anforderung des dem Spiel zugrunde liegenden Projektes ist jedoch auch die Funktionsfähigkeit auf mobilen Geräten wie Tablets und Smartphones. Diese ist jedoch stark eingeschränkt, ruckelnde Animationen machen Reaktionsspiele praktisch unspielbar. Die Bewegung in der Spielwelt ist durch eine optische Rotation realisiert und ebenfalls von dieser Problematik betroffen, Navigationsbefehle werden teils mit erheblicher Verzögerung und schlechter Performance umgesetzt und vermittelt nicht den Eindruck einer flüssigen Drehung, wie das auf dem PC der Fall ist. Die für den Lerneffekt und die Zielgruppe essentielle Audio-Ausgabe funktioniert stellenweise überhaupt nicht und das Gesamtbild ist obendrein von regelmäßigen Komplettabstürzen des Browsers geprägt.

Das Ziel dieser Arbeit ist es demzufolge durch Beseitigung der genannten Probleme die Spielbarkeit ebenso auf Mobilgeräten zu erreichen und ein mit der PC-Version vergleichbares Spielerlebnis zu gewährleisten.

1.2 Aufbau der Arbeit

Diese Arbeit ist in drei wesentliche Bestandteile untergliedert. Kapitel 2 liefert mit der Erklärung der Begriffe HTML5, Mobilgerät und Performance, und einer Beschreibung des Browser-Rendervorgangs das grundlegende Wissen.

Im Kapitel 3 folgt eine kurze Beschreibung des Spiels als Problembeispiel, die Erläuterung der Problemanalyse, sowie eine Beschreibung der benutzten Methoden und Werkzeuge. Des Weiteren werden technische Details der Geräte der Testumgebung gelistet und die Grenzen der Testbarkeit in dieser Arbeit aufgezeigt.

Das Kapitel 4 beschreibt die erkannten Probleme im Detail und liefert dazu passende Lösungsansätze. Es geht ein auf Animationsschleifen, die Animation von CSS-Eigenschaften und ihre Folgen für das Browser-Rendering, Verbesserung von Speichernutzung, und Limitierungen im Audio- und Videobereich. Zuletzt wird hier noch eine Möglichkeit der Unterscheidung zwischen PC und Mobilgerät vorgestellt, außerdem werden kurz zwei Frameworks beschrieben, welche die Optimierung bestimmter Problembereiche vereinfachen.

Den Abschluss der Arbeit bilden die Zusammenfassung und ein kurzer Ausblick.

2 Begriffe und Grundlagen

2.1 HTML5

HTML5 ist der Nachfolger der 1999 standardisierten Auszeichnungssprache HTML 4.01, welche bis auf Fehlerkorrekturen mit dem 1997 festgelegten Standard HTML 4 identlisch ist. Er wurde im Oktober 2014 vom World Wide Web Consortium, kurz W3C, als final verabschiedet und löst damit nicht nur HTML 4.01 ab, sondern auch XHTML als ursprünglich angedachten Nachfolger von HTML 4.01. Dass seine Entwicklung 15 Jahre dauerte, hat viele Gründe, deren detaillierte Erläuterung den Rahmen sprengen würden. Da der HTML5-Begriff aber allgegenwärtig ist und teilweise schwammig gebraucht wird, folgt an dieser Stelle dennoch der Versuch einer Kurzfassung seiner Herkunft und der verschiedenen Gebrauchsformen. Der interessierte Leser sei außerdem auf das einleitende Kapitel[2] von [HTML11] verwiesen, es dient als Quelle für den folgenden Absatz.

Die Entwicklung und Definition von HTML5 wird neben dem W3C auch noch von der Web Hypertext Application Technology Working Group, kurz WHATWG, vorangetrieben. Diese wurde 2004 von den Unternehmen Mozilla Foundation, Opera Software und Apple ins Leben gerufen, da sie mit dem vom W3C bevorzugten Standard XHTML (Umfang zunächst identisch mit HTML 4.01[3]) und auch der als bürokratisch empfundenen Arbeitsweise des W3C nicht einverstanden waren[4]. Sie nannten ihren Entwurf der HTML4-Weiterentwicklung zunächst Web Applications 1.0, später HTML5. Dieser war im Gegensatz zu XHTML rückwärtskompatibel. Da die WHATWG-Unternehmen ihre Marktmacht als Browser-Entwickler nutzen konnten und mit der Implementierung ihrer Idee des Standards die Praxis bestimmten, und außerdem der Internet Explorer bis zur Version 8 nicht XHTML-kompatibel war (IE9 erschien erst 2011), setzten sich die Ideen der WHATWG schließlich durch. So übernahm die W3C den HTML5-Entwurf der WHATWG von 2007 und entwickelte auf dieser Basis parallel zur WHATWG und zum eigenen XHTML-Standard noch ihre eigene HTML5-Version weiter. 2009 wurde vom W3C schließlich auch XHTML2 fallengelassen und so war HTML5 praktisch der einzige Nachfolger für HTML4. Im Jahr 2012 beschlossen beide Arbeitsgruppen, dass das W3C einen finalen, festen

2 [HTML11] S. 23-35
3 [WikDe15] Abschnitt: *XHTML 1.0: Übergang von HTML zu XHTML*
4 [HTML11] S.30

Standard vorantreibt, während die WHATWG einen "Living Standard" weiterentwickelt, in den kontinuierlich Neuerungen einfließen und der nur HTML heißt[5]. Der von W3C angestrebte finale Stand wurde schließlich im Oktober 2014 erreicht.

Im allgemeinen Sprachgebrauch sind HTML5 und die damit verbundenen Technologien nicht eindeutig getrennt und schließen auch Bereiche ein, welche nicht direkt unter dem Begriff HTML5 spezifiziert sind. Die folgende Grafik verdeutlicht die Weitläufigkeit des Begriffes.

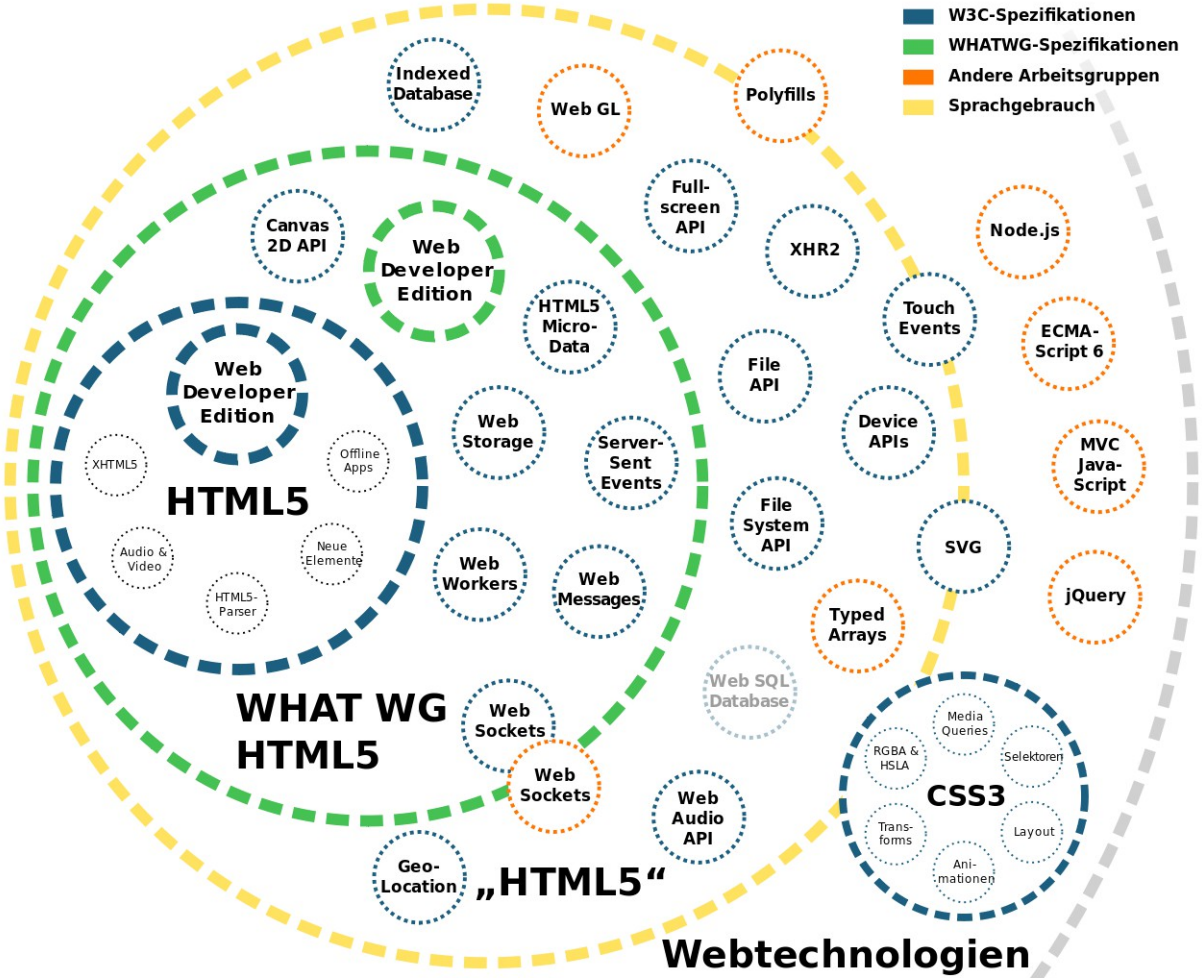

Abbildung 1: Der unterschiedliche Gebrauch des Begriffs HTML5, Quelle: [WikDe12]

Der W3C-HTML5 Standard führt neue Elemente wie `<section>`, `<header>` und `<footer>` zur Strukturierung ein, des Weiteren die Multimedia-Elemente `<audio>`, `<video>` und `<canvas>`. Letzteres ist eine digitale Leinwand, auf der auf Pixelbasis gezeichnet werden kann. Die Formular-Funktionalität wurde ebenfalls erweitert, so lässt sich beispielsweise der Typ des `<input>`-Elements genauer bestimmen,

5 [WikEn15a] Abschnitt: *Standardization process*

beispielsweise als Eingabefeld für eine Telefonnummer oder E-Mail-Adresse.

Im Kontext dieser Arbeit wird der HTML5-Begriff in einem größeren Rahmen genutzt, wodurch auch neuere Javascript-APIs und die Weiterentwicklung des CSS-Standards eingeschlossen werden. In Abbildung 1 auf Seite 5 ist dieser durch den gelben umrandeten Kreis gekennzeichnet. Relevant sind beispielsweise die Web Audio API und die transform-Eigenschaft des CSS3-Standards, auch wenn es rein formal nicht korrekt wäre, diese als HTML5-Technologie zu bezeichnen.

2.2 **Mobiles Gerät**

Für den Begriff Mobilgerät gestaltet es sich schwierig, eine einheitliche Definition zu finden. In dieser Arbeit sind damit Geräte der Kategorie Smartphone oder Tablet gemeint. Ihre Gemeinsamkeiten sind bestimmt durch Tragbarkeit, eine Bildschirmdiagonale zwischen etwa 3 und 10 Zoll, ein berührungsempfindliches Display, Kommunikationsschnittstellen, die Internetzugang ermöglichen, und ein Betriebssystem, welches durch sogenannte Apps, also zusätzliche Anwendungsprogramme, erweitert werden kann[6]. Die Firma ARM Limited, deren gleichnamiges Microprozessordesign in den meisten Mobilgeräten zum Einsatz kommt, beschreibt die Hardware solcher Geräte als hochintegiert und SingleChip-Design orientiert und spricht von einer guten Energie-Effizienz und damit Akkulaufzeit[7]. Ein Notebook hingegen wird trotz Tragbarkeit in diesem Zusammenhang wegen seiner leistungsstarken Hardware als PC kategorisiert.

Zusätzlich verfügt ein mobiles Gerät über die Fähigkeit, Audio und Video wiederzugeben. Es unterscheidet sich von einem PC durch eine deutlich schwächere Hardware-Ausstattung, sowohl bei der Prozessorleistung als auch bei der Speicherkapazität. Dominiert werden Mobilgeräte von den Betriebssystemen Android und iOS, welche zusammen fast 93% Marktanteil auf sich vereinen[8]. Auf beiden Plattformen ist zumindest ein moderner, HTML5-fähiger Browser verfügbar. Auch besitzen die Geräte in den meisten Fällen eine dedizierte GPU, sowie oft noch Beschleunigungssensoren, einen Kompass und einen GPS-Empfänger[9].

6 [MDM12] S.15
7 [ARM14]
8 [Stat15]
9 [WikEn15b]

2.3 **Performance**

Der Begriff Performance ist das eingedeutschte Substantiv des englischen Verbs *to perform,* welches durchführen, ausüben oder leisten bedeutet. Entsprechend wird Performance mit Durchführung oder Leistung übersetzt. Letzteres ist auch der Sinn, in welchem der Begriff meist im Zusammenhang mit Computer- und Softwaretechnik gebraucht wird. Er beschreibt die Leistungsfähigkeit eines Systems, und wird an praktisch jede Software als Anforderung gestellt. Bei einer klassischen Webanwendung geht es bei Leistung in erster Linie um Durchsatz in Form der verarbeiteten Anfragen pro Zeiteinheit und um die Antwortzeit, welche eine Anwendung für Bearbeitung einer einzelnen Anfrage benötigt[10]. Da die Verarbeitung von Anfragen auf dem Server geschieht, handelt es sich um die serverseitige Performance.

In dieser Arbeit ist dies jedoch nicht der entscheidende Faktor, da die Anwendung und jedes Minispiel zu Beginn komplett geladen werden und im Verlauf dieses Spiels keine weiteren Anfragen mehr an den Server gestellt werden müssen. Die zu erbringende Leistung wird somit primär clientseitig benötigt, also auf dem Endgerät, angefordert durch die Browser-Software.

Im Bezug auf die Performance gibt es bei einem Browser folgendende Kriterien[11]:

- **Ladezeit**: wie schnell kann eine Webseite aufgerufen werden
- **Ansprechverhalten**: wie schnell kann auf die Eingaben des Benutzers reagiert werden
- **Bildwiederholungsrate**: wie schnell kann der Bildschirmhalt erneut generiert und angezeigt werden, damit eventuelle Bewegungen bzw. Veränderungen dem Benutzer flüssig erscheinen
- **Speichernutzung**: wie wird der Speicher belastet, müssen andere Anwendungen geschlossen werden, oder bringt Speichermangel den Browser gar zum Absturz
- **Stromverbrauch**: wie stark belastet die angeforderte Rechenleistung die Stromversorgung

10 [DatWe03] S. 208
11 [DevMoz14]

Ausgehend von der Problemstellung soll der Begriff Performance an dieser Stelle zusätzlich noch etwas allgemeiner ausgelegt werden. Da bestimmte Anforderungen des Spiels wie die Audio-Ausgabe auf den Zielgeräten schlicht gar nicht erfüllt werden, also nicht durchführbar sind, erhält der Begriff zusätzlich noch eine Art binäres Maß im Sinne von "funktionstüchtig ja/nein".

2.4 Rendering im Browser

Der Rendervorgang eines HTML-Dokuments durchläuft die folgenden Schritte[12]:

1. Download der HTML-Daten (oder Laden von einem lokalen Datenträger)
2. Parsen des HTML-Codes in das Document Object Model (DOM)
3. Laden der CSS-Daten, Parsen der Informationen in das Cascaded Style Sheets Object Model (CSSOM)
4. Erstellen des Renderbaums durch Verknüpfung von DOM und CSSOM, an dieser Stelle wird jedes DOM-Objekt mit seinen Style, also der optischen Repräsentation verknüpft. Objekte mit dem Style 'display: none' werden nicht in den Render-Baum aufgenommen, ebensowenig generell unsichtbare Elemente wie `<head>`, `<script>` oder `<style>`.
5. **Layout** (oder Reflow), für jedes einzelne DOM-Objekt im Renderbaum wird Größe und Position berechnet. Relevante CSS-Eigenschaften: `width`, `height`, `position`, `left`, `top`, `margin`, `padding`, `float`, `line-height` usw[13].
6. **Paint**, die DOM-Objekte bekommen ihr finales Aussehen auf Pixelebene, relevante CSS-Eigenschaften: u.a. `color`, `background`, `box-shadow`, `border`, `visibility`, `outline`[14].
7. **Composite**, einzelne Layer werden erzeugt und in den Speicher der Grafikkarte geschrieben. CSS-Eigenschaften, für ein DOM-Objekt einen eingen Layer generieren, sind `opacity` und bestimmte Werte von `transform`. Jeder Layer wird im Grafikspeicher gesondert gespeichert. Die GPU setzt dann beim nächsten Bildaufbau diese zum Gesamtbild zusammen[15].

Die Schritte 1 - 4 sind für diese Arbeit von geringer Bedeutung, sie seien nur der

12 [DevGo14]
13 [H5Ro13] Abschnitt: Animating Layout Properties
14 [H5Ro13] Abschnitt: Animating Paint Properties
15 [Chro14]

Vollständigkeit halber erwähnt. Einzelne Spiele werden zu Beginn geladen, danach muss kein neuer HTML- und CSS-Code geparsed werden. Wichtig sind die hervorgehobenen letzten drei Schritte, da Animationen des Spiels durch die zeitgesteuerte Neuberechnung von CSS-Eigenschaften realisiert werden.

3 Methodik

Die prinzipielle Vorgehensweise beginnt mit der Analyse eines Problems durch Anwendung der in diesem Kapitel beschriebenen Methoden und Werkzeuge. Es folgt die Recherche und, sofern möglich, das Erstellen eines Testszenarios für einen ermittelten Lösungsansatz gefolgt von einer Erprobung in der Anwendung. Einzelne Tests sind stellenweise in den Unterpunkten des Kapitels Anpassung und Optimierung exemplarisch dargestellt.

An dieser Stelle ist es notwendig zu erwähnen, dass die Messung der Wirksamkeit von Optimierungen von vielen Faktoren abhängig ist und eine messbare Verbesserung nicht automatisch allgemeingültig sein muss. Nennenswerte Faktoren sind verschiedenartigste Hardware-Grundlagen, die Version des Betriebssystems, der Treiber und anderer Softwarekomponenten, und außerdem der Lastzustand des Geräts. Gerade im Bereich Webanwendungen stellt dies ein generelles Problem dar, da sich die Vielfalt der möglichen Endgeräte und Softwarezustände nicht ohne Weiteres empirisch abdecken lässt. Daher ist es im Rahmen dieser Arbeit nicht möglich, diesem Gesichtspunkt die notwendige Aufmerksamkeit zu widmen. Um aber zumindest die tendenzielle Funktionsfähigkeit zu gewährleisten, wurde eine im Kapitel Testumgebung näher beschriebene Gerätekonstellation zusammengestellt, die im Rahmen der verfügbaren Mittel auf eine möglichst breite Marktabdeckung abzielt. Außerdem ist darauf zu achten, dass auf den Geräten möglichst wenig Apps im Hintergrund laufen, und dass durch einen Neustart vor den Tests eine möglichst "saubere" Testumgebung zustande kommt.

Des Weiteren lässt sich nicht auszuschließen, dass die Art der Messung einen Einfluss auf die zu messende Performance hat. Da sich in vielen Fällen auf die Chrome DevTools (Kapitel 3.3) bezogen wird, muss in Anbetracht des knappen Rahmens einer Bachelorarbeit ein etwas pragmatischer Ansatz gewählt werden: Bringt eine potenzielle Optimierung im isolierten Test eine messbare, und bei Übertragung in der Anwendung eine spürbare Verbesserung, so wird diese Optimierung als Erfolg gewertet.

In den folgenden Kapiteln werden das Spiel, Messmethoden, Werkzeuge und die Testumgebung näher erläutert.

3.1 Das Spiel Letti ABC

Grundlage dieser Arbeit ist das HTML5-basierte Kinderspiel "Letti ABC", welches in einer auf dem PC-Browser Google Chrome lauffähigen Variante vorliegt. Das vom Bundesministerium für Familie, Senioren, Frauen und Jugend und der Initiative "Ein Netz für Kinder" geförderte Projekt soll Kindern im Grundschulalter spielerisch die Geschichte der Formgebung der Buchstaben des lateinischen Alphabets vermitteln. Die Spielfigur Letti ist nach dem Zusammenstoß mit einem Kometen auf der Spielwelt gestrandet, dabei wurde das aus einzelnen Buchstaben bestehende Raumschiff zerstört. Die Buchstaben sind in der Welt verteilt und mit je einem Minispiel verbunden. Diesem ist die Erklärung der Herkunft des jeweiligen Buchstaben vorangestellt. Das erfolgreiche Spielen schaltet den Buchstaben frei, welcher dann wieder als Bauteil für das Raumschiff dient. Sind alle Minispiele gewonnen und damit das Raumschiff komplett, kann Letti die Spielwelt wieder verlassen.

Der folgende Screenshot vermittelt einen Eindruck von der Spielwelt. Es wurden schon einige Buchstaben freigeschaltet, entsprechende Teile der Welt sind eingefärbt. Zu sehen sind außerdem die begleitende Spielfigur Letti und sein noch unvollständiges Raumschiff.

Abbildung 2: Die Spielwelt mit Letti und dem unfertigen Raumschiff

Ein weiterer Screenshot zeigt eins der 24 Minispiele. Es handelt sich um ein Reaktionsspiel, wo im richtigen Moment durch einen Klick auf die Brille diese nach oben schnippt und damit das passende durchlaufende Augenpaar getroffen werden muss.

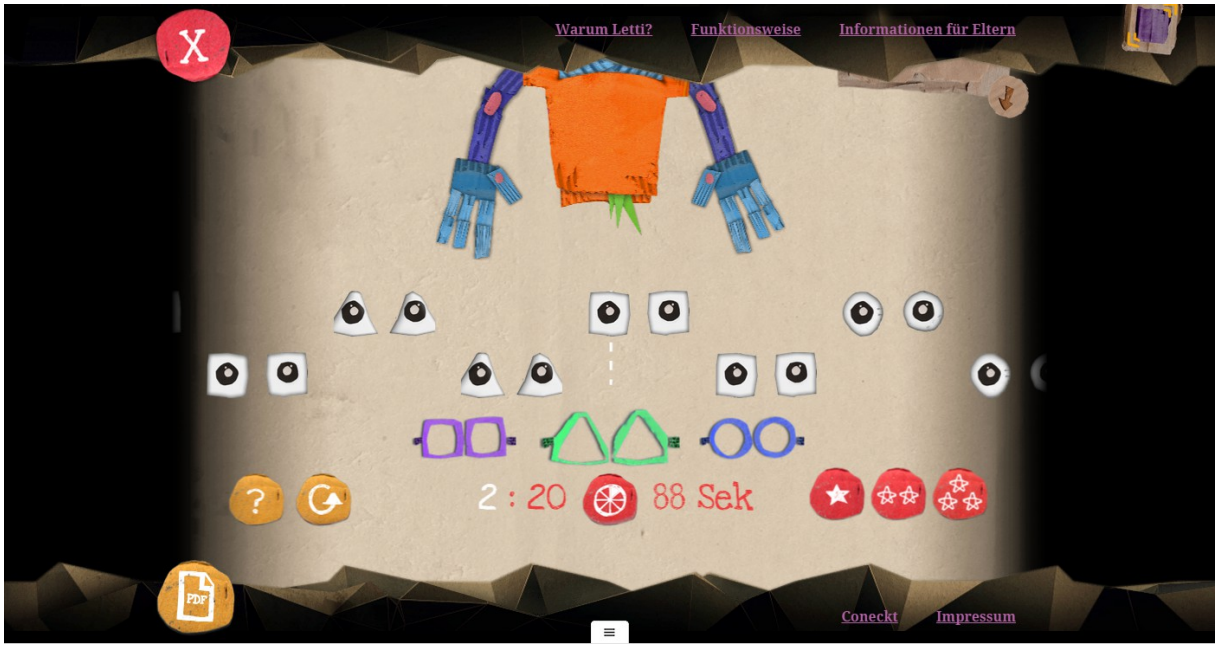

Abbildung 3: Ein Beispiel für ein Minispiel

3.2 **Profiling**

Als Profiling bezeichnet man in der Softwareentwicklung die Analyse von Programmen zur Laufzeit. Ziel ist das Ermitteln von Problembereichen, in erster Linie um kritische bzw. ineffiziente Codeabschnitte innerhalb von Programmen zu finden. Oft spricht man von sogenannten Flaschenhälsen. Diese müssen beseitigt werden.

Im Bereich der Webentwicklung können folgende Bereiche profiliert werden.

- **Ladezeit**: Durch Analyse der einzelnen Schritte, insbesondere der Datei-Anfragen beim Server, können Verzögerungen des initialen Seitenaufbaus untersucht werden.
- **Rechenzeit**: Der Zeitbedarf und die Häufigkeit von Funktionsaufrufen in Javascript werden gemessen.
- **Rendering**: Während der Manipulation von Elementen wird ermittelt, welche Prozesse hierdurch in der Render-Engine des Browsers ausgelöst werden,

und welche Teile davon wieviel Zeit in Anspruch nehmen. Technisch gesehen hängt dieser Aspekt mit dem Punkt Rechenzeit zusammen, jedoch handelt es sich um eine andere Betrachtungsperspektive, wie das folgende Kapitel zeigen wird.

- **Speicherverbrauch**: Über die Laufzeit wird betrachtet, wann eine Software bei welchen Prozessen wieviel Speicher verbraucht. Oft wird der Speicherbedarf dabei als Graph über einer Zeitachse ermittelt. Hierbei können neben dem generellen Speicherverbrauch auch eventuelle Speicher-Leaks erkannt werden.

In dieser Arbeit liegt der Schwerpunkt auf den Bereichen Renderung und Speicherbedarf. Der Punkt Ladezeit ist nebensächlich, da die Anwendung einmal aufgerufen wird und ab dann ohne Seitenwechsel im Browserfenster abläuft. Lediglich der Start eines Minispiels erfordert einen weiteren Ladevorgang. Die Rechenzeit ist hier ebenfalls nicht von Bedeutung, da bis auf die Animationen von Elementen keine aufwendigen Berechnungen notwendig sind, und sich das Laufzeitverhalten von Animationen vor allem im Rendering niederschlägt.

3.3 Chrome DevTools

Das zentrale Analyse-Werkzeug für diese Arbeit sind die mit Google Chrome mitgelieferten DevTools. Sie ermöglichen die Ermittlung von Daten aus allen im Profiling-Kapitel gelisteten Bereichen. Des Weiteren ist neben der Analyse von Seiten im lokalen Browser ohne Einschränkungen auch die Anwendung auf Chrome-Browser-Fenster möglich, die auf per USB verbundenen Android-Geräten laufen, sogenanntes Remote-Debugging[16].

Im Folgenden werden einige Funktionen der DevTools vorgestellt, die für die Analyse genutzt werden. Ein sehr universelles Werkzeug ist die Timeline. Hiermit lässt sich der Verlauf eines Programms, bzw. ein Ausschnitt davon aufzeichnen und untersuchen. Ein Abspeichern für spätere Analyse ist ebenfalls möglich.

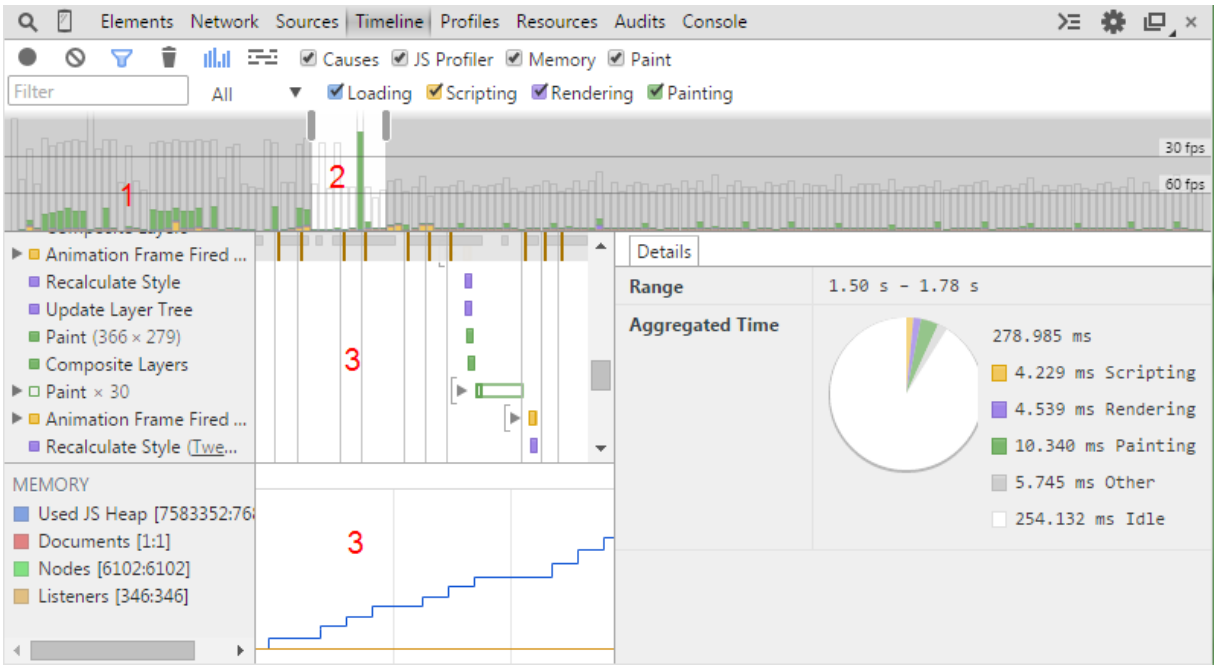

Abbildung 4: Aufgezeichnete Timeline

Im oberen Bereich wird der Verlauf der Framerate über den Aufnahmezeitraum angezeigt (1). Hier werden einzelne Render-Frames als Balken dargestellt, die farbige Füllung repräsentiert die Zeitanteile einzelner Render-Schritte, je höher der Balken bzw. dessen Inhalt, desto größer war der Zeitbedarf. Die Renderschritte wurden im Kapitel 2.4 näher erläutert. Der Verlauf lässt sich auf einen Bereich beschränken (2), entsprechend werden die darunter sichtbaren Details (3) auf diesen Zeitraum eingegrenzt. Diese zeigen die abgelaufenen Render-Schritte mit Zeitanteilen, des Weiteren (sofern aufgezeichnet) den Verlauf der Speicher-

16 https://developer.chrome.com/devtools/docs/remote-debugging

auslastung.

Innerhalb der Detaileinschränkung (1) der folgenden Abbildung lässt sich sehr präzise der Verlauf von Javascript-Funktionen und -Ereignissen verfolgen.

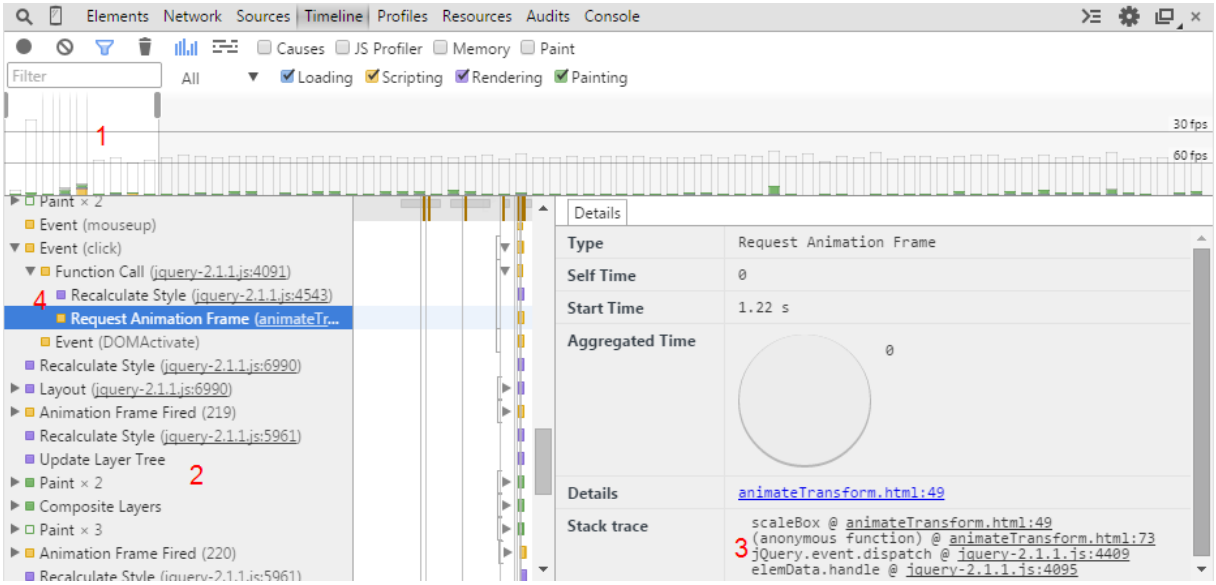

Abbildung 5: Javascript-Ereignisse innerhalb einer Timeline-Aufzeichnung

Hiermit wird es vereinfacht, den Auslöser bestimmter Renderprozesse einzugrenzen und ggf. zu optimieren (2)(3)(4).

Ebenfalls hilfreich ist die Möglichkeit, dauerhaft eine FPS-Anzeige zu aktivieren, welche Frameratеverlauf und die Speicherauslastung zeigt. Sie unterstützt den Entwickler, indem sie ihm dauerhaft die Möglichkeit bietet, eventuelle Performanceveränderungen bei der Benutzung der Anwendung zu erkennen, ohne dass explizit eine Analyse gestartet bzw. aufgezeichnet werden muss. Somit kann sie Indizien liefern, welcher Vorgang in der Anwendungslogik für eine tiefere Analyse interessant ist.

Abbildung 6: FPS-Anzeige

Um den Speicherverbrauch einer HTML-Anwendung zu analysieren, bieten die DevTools eine Heap-Snapshot-Funktion. Mit dieser ist es möglich, Momentanaufnahmen des Speichers festzuhalten und diese zu analysieren.

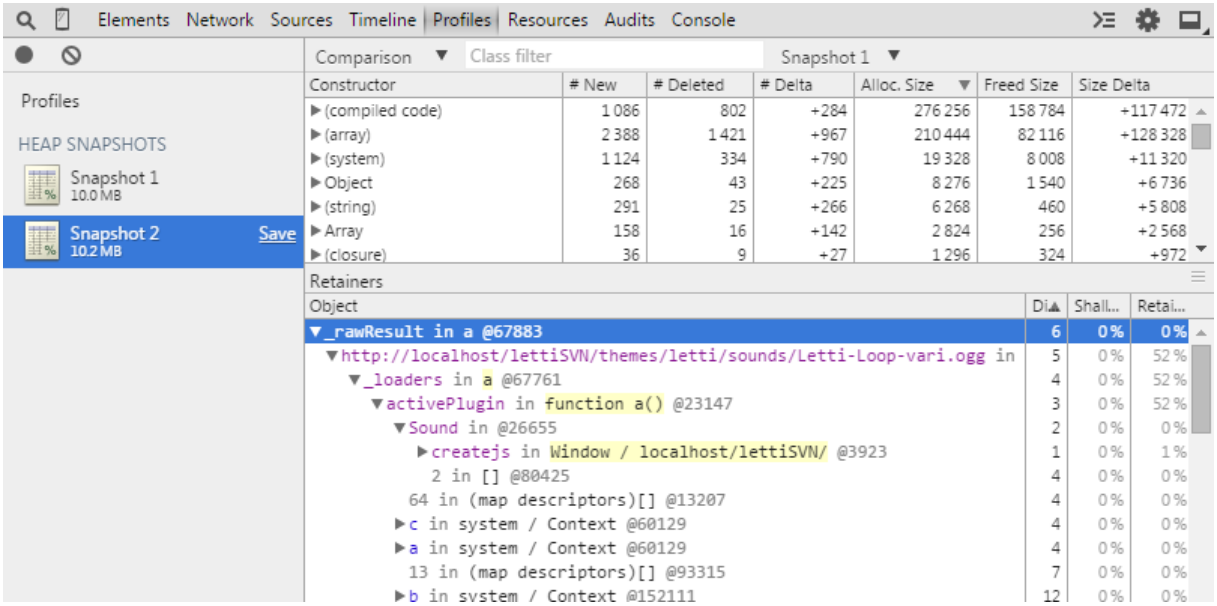

Abbildung 7: Vergleich zweier Heap-Snapshots

Darüber hinaus ist es möglich, mehrere Aufnahmen miteinander zu vergleichen und neu erstellte sowie gelöschte Speicherobjekte zu ermitteln. Abbildung 7 zeigt einen solchen Vergleich. Der obere Bereich stellt die Differenzen zwischen den Snapshots, sortiert nach Klassen, dar und im unteren Bereich lässt sich ein einzelnes Speicher-Objekt näher untersuchen. Eine baumähnliche Struktur veranschaulicht die Referenzierungen des Objektes und lässt ggf. Rückschlüsse zu, warum der Garbage Collector das Objekt nicht bereinigen kann.

3.4 **Testumgebung**

Wie im einleitenden Teil des Kapitels Methodik schon erwähnt, wurde eine Testumgebung zusammengestellt, die ein möglichst breites Spektrum der sich in Benutzung befindenden Geräte und Betriebssysteme abdecken soll. Sie besteht aus vier Tablets und zwei Smartphones, welche in der folgenden Tabelle gelistet sind. Ein aktuelles Windows-Phone-Gerät stand leider nicht zur Verfügung.

	Google Nexus 7	**Samsung Galaxy Tab 7 Rev2**	**Apple iPad 2**	**Apple iPad 3**
Hersteller	ASUS	Samsung	Apple	Apple
Modell-Nr.	Nexus 7 2013	P3100	A1395	A1416
Release	2013	2012	2011	2012
CPU	Krait 300 4x1,5 GHz	Cortex-A9 2x1 GHz	ARM-Cortex A9 2x1 GHz	
GPU	Adreno 320	PowerVR SGX540	PowerVR SGX543MP2	PowerVR SGX543MP4
RAM	2 GB	1 GB	512 MB	1 GB
Auflösung	1920x1200	1024x600	1024x768	2048x1536
Betriebssystem	Android 4.4.3	Android 4.2.2	iOS 8.1	
Browser	Google Chrome 40.0.2214.109		Safari 8	

Tabelle 3.1: Für Tests genutzte Tablets

	Motorola DROID 4	**Sony Xperia pro**
Hersteller	Motorola	Sony
Modell-Nr.	XT894	MK16i
Release	2012	2011
CPU	Cortex-A9 2x1,2 GHz	Qualcomm MSM8255 1 GHz
GPU	PowerVR SGX540	Adreno 205
RAM	1 GB	512 MB
Auflösung	960x540	854x480
Betriebssystem	Android 4.4.4	Android 4.0
Browser	Google Chrome 40.0.2214.109	

Tabelle 3.2: Für Tests genutzte Smartphones

Die verwendete Hardware entspricht in vielen Fällen nicht mehr dem aktuellen Stand. Dies ist jedoch kein Nachteil, da im Allgemeinen die Performance mit der Hardware skaliert und so davon auszugehen ist, dass eine für ältere Geräte optimierte Software auf neuer Hardware tendenziell sogar besser funktioniert.

Problematisch ist das Debugging und Profiling der Apple-Geräte auf einem Windows-PC. Das im Kapitel Chrome DevTools erwähnte Remote-Debugging ist mit der iOS-Version vom Chrome nicht möglich. Es handelt sich bei dieser Chrome-Variante nur um eine Art Frontend[17] für die Browser-Engine des Standardbrowsers Safari, auf die mittels der UIWebview-Technik zugegriffen wird, um HTML-Dokumente zu rendern. Remote-Debugging des Safari auf iOS ist zwar ähnlich dem Beispiel der Chrome DevTools möglich, dafür wird aber eine aktuelle Version des Desktop-Safari benötigt. Dessen Entwicklung wurde für Windows nach Version 5 eingestellt, und ein Apple-Gerät nebst aktuellem MacOS-Safari stand im Rahmen dieser Arbeit nicht zur Verfügung. Aus diesem Grund wurden die Tests der Auswirkung von Optimerungen auf Apple-Geräten auf die Auswirkung einzelner Optimierungen, sofern nicht anders möglich, auf die reale Anwendung beschränkt.

17 [DevChro]

4 Anpassung und Optimierung

In diesem Kapitel werden die aufgetretenen Probleme und ihre Lösung exemplarisch dargestellt. Die Code-Beispiele verwenden stellenweise das jQuery-Framework[18], zu erkennen am `$()`-Syntax.

4.1 Animationen

4.1.1 Animationsschleifen

Eine Animationsschleife in Javascript folgte in etwa diesem Schema:

```
1    function doAnimation() {
2      setTimeout(doAnimation, 1000/60);
3      renderAnimations();
4    }
5    doAnimation();
```

Innerhalb der Funktion `doAnimation` legt diese zunächst ihren nächsten Aufruf fest, im Beispiel mit einer Verzögerung von 1000/60=16,67ms, also 60 Aufrufen pro Sekunde. Im Anschluss an diesen verzögerten Selbstaufruf werden durch `renderAnimations` die eigentlichen Animationen berechnet. Diese Bildwiederholrate von 60 frames per second wurde nicht willkürlich gewählt, sondern ist für Computerspiele ein anzustrebendes Ideal[19].

Dieser Algorithmus hat ein paar Probleme[20]. Er rendert mit der festgelegten Wiederholungsrate, unabhängig davon, ob der Browser in dem Moment überhaupt den Bildschirm neu zeichnen wird oder nicht. Wird also nach dem Aufruf von `doAnimation` nicht gezeichnet, ist die von `renderAnimations` beanspruchte CPU-Rechenzeit verschwendete Leistung, da das Ergebnis zunächst nicht dargestellt wird. Dies tritt beispielsweise dann auf, wenn die gewählte Wiederholungsrate oberhalb der Bildwiederholungsrate des Ausgabegeräts liegt. In so einem Fall ist es möglich, dass zwei Animationsschritte berechnet werden, jedoch erst nach dem zweiten das Neuzeichnen des Bildschirms eintritt. Zusätzlich zu vergeudeter CPU-Zeit erscheint durch das Nichtzeichnen, also Überspringen eines Zwischenschritts die Animation ungleichmäßig. Des Weiteren wird die wiederholte Berechnung selbst dann ausgeführt, wenn das Fenster oder der Tab des Browsers gar nicht sichtbar ist, weil sie in einem anderen Tab oder minimiertem Fenster abläuft.

18 http://jquery.com/
19 [PcGHw12]
20 [DevOp13]

Durch eine starre Wiederholungsrate besteht außerdem die Gefahr, dass `doAnimation` erneut aufgerufen wird, obwohl `renderAnimations` seit dem letzten Aufruf noch gar nicht mit der Berechnung fertig ist. Dies wäre im obrigen Beispiel der Fall, wenn renderAnimations länger als 16,67 ms für seine Ausführung benötigt. Diese Überschneidung kann zu Inkonsistenzen führen, welche im Extremfall den Browser zum Absturz bringen.

Betroffen von dieser Problematik sind natürlich nicht exklusiv mobile Geräte, aber hier schlägt sich die Verschwendung von Rechenzeit deutlicher nieder. Knappe Hardware-Ressourcen sollen möglichst effizient genutzt werden, des Weiteren erhöhen unnötige Berechnungsprozesse den Stromverbrauch und wirken sich damit negativ auf die Akkulaufzeit aus.

Um diese Probleme zu lösen wurde, zuerst von Mozilla, dann gefolgt von weiteren Browserherstellern, die **requestAnimationFrame-API** eingeführt. Diese ist heute standardisiert[21] und in allen aktuellen Browsern verfügbar[22]. Sie nimmt sich der geschilderten Problemen an, indem sie genau dann einen Callback aufruft, wenn der Browser bereit ist, den Bildschirm neu zu zeichnen. Wann dies geschieht, entscheidet die Browser-Engine nach Kriterien, die in [W3tc13] näher beschrieben sind. So ist der Browser beispielsweise in der Lage, seine Bildwiederholungsrate mit der des Ausgabegerätes zu synchronisieren, des Weiteren wird kein neuer Frame gezeichnet und ein Callback ausgeführt, solange die Berechnung des alten noch nicht abgeschlossen ist.

Die Umstellung ist einfach, da die Syntax im Wesentlichen gleich ist. Die Animationsschleife mit der requestAnimationFrame-API sieht nun folgendermaßen aus:

```
1    function doAnimation() {
2      requestAnimationFrame(doAnimation);
3      renderAnimations();
4    }
5    doAnimation();
```

Der für setTimeout notwendige Zeit-Parameter ist verschwunden, da nun der Browser selbst den geeigneten Zeitpunkt wählt.

21 [W3tc13]
22 http://caniuse.com/#feat=requestanimationframe

Für einen Vergleich beider Techniken wurde folgender JS-Testcode erstellt, welcher eine zeitgesteuerte, framerate-unabhängige Animation realisiert. Er animiert im Zeitraum von zwei Sekunden die Bewegung einer Box über eine Distanz von 1000 Pixel. Das komplette HTML-Dokument ist in Anlage B zu finden.

```
1    var type = 'setTimeout',
2      fps = 60,
3      left = 0,
4      callCount = 0,
5      aniDist = 1000, //Animationsdistanz 1000px
6      aniTime = 2000, //Animationszeit 2 Sekunden
7      lastTimeStamp = Date.now();
8
9    function moveBox() {
10     if (left > aniDist) {
11         console.log('moveBox() callCount with '+type+': '+callCount);
12         return;
13     } else {
14         if (type == 'setTimeout')
15             setTimeout(moveBox, 1000 / fps);
16         else if (type == 'requestAnimationframe')
17             requestAnimationFrame(moveBox)
18     }
19     callCount++;
20     var tsNeu = Date.now(),
21         diff = aniDist * (tsNeu - lastTimeStamp) / aniTime;
22     lastTimeStamp = ts;
23     left = left + diff;
24     $('#box').css('margin-left', left + 'px');
25   }
```

In diesem Beispiel wird durch den einmaligen Aufruf von `moveBox()` die Animation gestartet. Zeile 10 definiert die Abbruchbedingung für das Ende der Animation und die Ausgabe der mit `callCount` gezählten Funktionsaufrufe. In Zeile 14 bzw. 16 wird entschieden, welcher Animationsschleifen-Typ `type` gewählt werden soll, Zeile 21 bis 23 berechnet die neue Position der Box in Abhängigkeit der aktuellen Zeit, diese Position wird anschließend an das Element übertragen.

Die Ausgabe endete mit folgenden Ergebnissen:

```
moveBox() calls with setTimeout: 120
moveBox() calls with requestAnimationframe: 103
```

Dies zeigt die Wirksamkeit dieser Methode. Trotz dass sich die Anzahl der Aufrufe um knapp ein Sechstel geringer ausfällt, wird die Bewegung im Browserfenster gleichermaßen flüssig dargestellt. Bei diesem Test war die Bildwiederholfrequenz des Monitors auf 50 Hz eingestellt und der Browser war in der Lage, entsprechende

Berechnungsschritte einzusparen.

4.1.2 Animation durch Veränderung von CSS-Eigenschaften

Dieses Kapitel nimmt Bezug auf die in Kapitel 2.4 beschrieben Rendering-Schritte. Wie dort bereits dargelegt, sind die letzten drei Schritte Layout, Paint und Composite von Bedeutung, da Animationen durch die zeitgesteuerte Neuberechnung von CSS-Eigenschaften realisiert werden. Deren Veränderung macht das erneute Durchlaufen von mindestens einem der drei letzten Schritte notwendig. Es ist also für die Performance von Bedeutung, welche Eigenschaft für eine Animation benutzt wird, da die Kosten an Rechenzeit jeweils unterschiedlich sind.

Dem **Composite**-Schritt kommt hier eine besondere Bedeutung zu, da er auf der Grafikeinheit ausgeführt werden kann. Da diese für derartige Aufgaben optimiert ist, kann sie die notwendigen Operationen deutlich schneller erledigen als die CPU. Man spricht in so einem Fall von Hardwarebeschleunigung. Dies macht es erstrebenswert, Animationen möglichst auf CSS-Eigenschaften zu beschränken, die eigene Composite-Layer erzeugen, das sind `opacity` und `transform`, letztere jedoch nur mit den Werten `translateZ`, `translate3d`, `rotateZ`, `rotate3d`, `scaleZ`, `scale3d` und `matrix3d`, wie sich herausstellte. Die endgültige Darstellung der so erzeugten Layer kann bei Beschränkung der Manipulation dieser Werte komplett auf der Grafikkarte ausgeführt werden, ohne die CPU-intensiven Schritte Layout und/oder Paint auszulösen.

In der Ausgangssituation war dies nicht der Fall. Die Bewegung von Elementen wurde zumeist über die Animation der CSS-Eigenschaften `top` bzw. `bottom`, und `left` bzw. `right` realisiert. Weiterhin wurde die Skalierung von Elementen entweder durch Anpassung der Dimensionen und Streckung des Hintergrundbildes mittels background-size vorgenommen, oder es wurde die 2D-Variante `transform: scale` genutzt, welche nicht auf der GPU ausgeführt wird.

Die Gegenüberstellung auf der nächsten Seite verdeutlicht den Unterschied. Das zugrundeliegende HTML-Dokument ist als Anlage C beigefügt.

Zunächst wird ein Objekt von left: 0px nach left: 500px animiert:

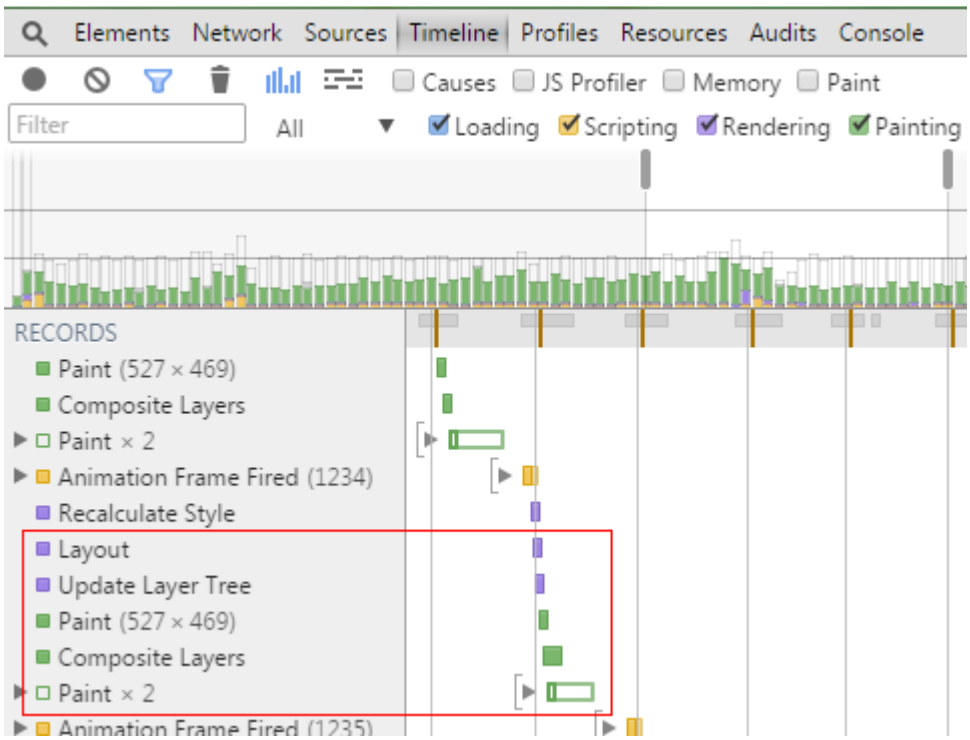

Abbildung 8: Animationsverlauf über Animation der CSS-Eigenschaft 'left'

Zu sehen ist, dass alle drei Schritte Layout, Paint und Composite durchlaufen werden. Anschließend wird die gleiche Bewegung realisiert, indem `transform3d(0,0,0)` nach `transform3d(500px,0,0)` animiert wird:

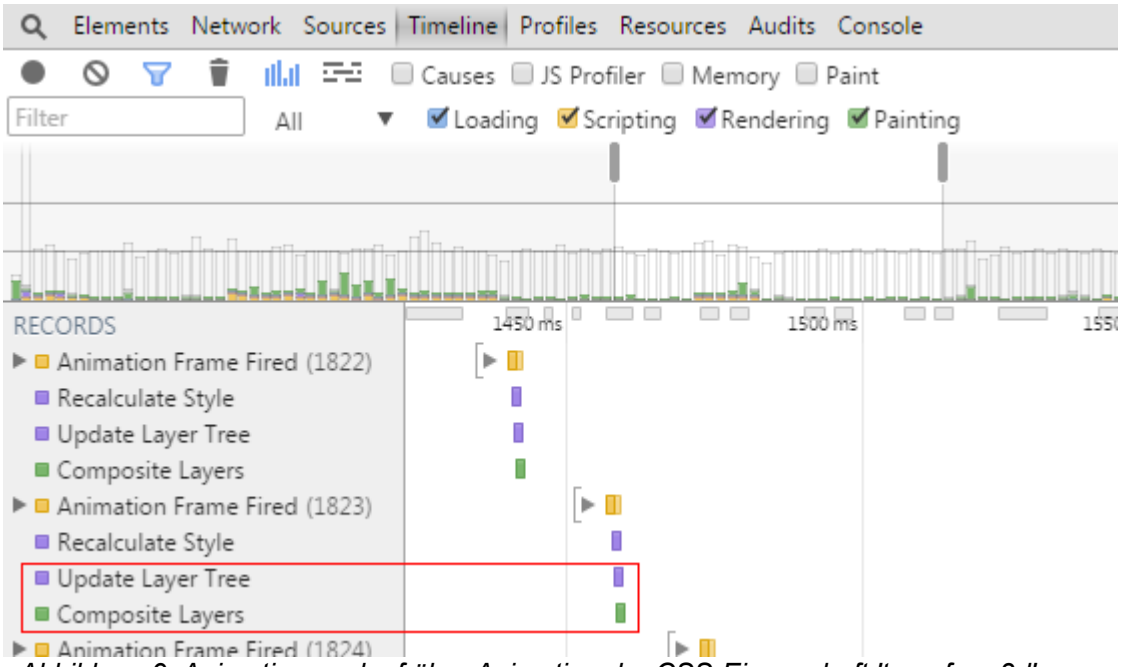

Abbildung 9: Animationsverlauf über Animation der CSS-Eigenschaft 'transform3d'

Die Schritte Layout und Paint sind verschwunden, die notwendige Berechnungszeit

eines Frames ist deutlich sichtbar gesunken. Außerdem wird die GPU-Speicherauslastung bei Verwendung von `transform3d` mit 4,4 MB angegeben, bei Animation von `left` dagegen mit 6,6 MB.

Zwar erreichten im Testbeispiel beide Animationen noch flüssige 60fps, da jedoch in der fertigen Anwendung meist wesentlich mehr als ein Element gleichzeitig animiert wird, ist hier der Unterschied viel deutlicher. Nach der Übertragung des transform-Ansatzes in das Spiel zeigen sich nun flüssige Animationsabläufe, welche zuvor nicht möglich waren. Anzumerken ist noch, dass dies natürlich nicht mit allen Animationen möglich ist und sich Layout- und Paint-Schritte nicht komplett eleminieren lassen.

4.2 Speichernutzung

4.2.1 Speicherverbrauch und Ladezeiten

Die ursprüngliche Anwendung ist für die FullHD-Auflösung, also 1920x1080 ausgelegt. Dies betrifft sowohl das Layout einzelner Teile wie der Spielwelt, das Menu, die Fortschrittsanzeige in Form des Raumschiffs und der Spielfigur Letti, als auch deren Texturen, welche in den allermeisten Fällen durch ein Hintergrundbild eines HTML-Elements, also mittels der CSS-Eigenschaft `background` realisiert sind. Die Anpassung an die verschiedenen Bildschirmauflösungen erfolgt, indem relevante Teile wie Spielwelt, Spielfigur und einzelne Spiele mittels der CSS-Eigenschaft `transform: scale()` (bzw seit der Erkenntnis aus Kapitel 4.1.2 `scale3d()`) skaliert werden, um eine optimale Ausnutzung des Browserfensters zu erreichen.

Im direkten Zusammenhang mit der Texturgröße steht der Speicherverbrauch, je mehr Pixel, desto größer auch der Speicherbedarf. Abstürze, insbesondere auf iOS, und eine anschließende Analyse des Crash-Logs ergab, dass nicht ausreichender Speicher dafür verantwortlich war.

Es ist daher notwendig, den Speicherbedarf zu verringern, und den größten Anteil daran haben Grafiken. Auch werden durch die Skalierung auf die kleineren Auflösungen von Mobilgeräten Texturen ohnehin selten in ihrer vollen Größe gezeigt, so dass dieser Schritt selbst ohne Abstürze eine sinnvolle Optimierung darstellt.

Ein weiterer Punkt, der eine Verkleinerung der Texturen sinnvoll macht, ist der Bandbreitenbedarf beim Laden der Anwendung. Nutzen mobile Geräte den Internetzugang über Mobilfunknetze, ist der Online-Tarif zumeist mit einem

Volumenlimit verbunden, welches bei Überschreitung zu einer starken Drosselung der Bandbreite führt. Außerdem erreichen Mobilfunknetze insbesondere zu Stoßzeiten oft nicht die Bandbreite eines festen Internetanschlusses, und somit ist ein positiver Nebeneffekt die Verringerung der Ladezeiten, auch wenn diese in den vorangegangenen Kapiteln als wenig relevant eingestuft wurden.

Elemente müssen also nebst ihrer Texturgrafik verkleinert werden. Dies ist mit einem relativ großem Aufwand verbunden, da nicht nur Grafiken verkleinert werden müssen (dieser Vorgang ist automatisierbar), sondern gleichzeitig auch die Pixel-Dimensionen der einzelnen Layout-Elemente und ggf. ihre Positionierung im Layout anzupassen sind. Würden die Dimensionen nicht angepasst und stattdessen lediglich die kleineren Textur-Grafiken per `background-size` auf die alte Dimension skaliert, so wäre zwar Bandbreite gespart, jedoch hätte der im Kapitel 2.4 beschriebene Paint-Schritt die gleiche Anzahl an Pixeln zu bearbeiten und es käme zu keinerlei Ersparniss an Rechenzeit und Speicherverbrauch.

Da die hierfür nötigen Änderungen mit einiger Arbeit verbunden sind, wurde sich bei der Zielauflösung für Mobilgeräte an den gängigen Auflösungen von Smartphones orientiert und eine Halbierung der Seitenlängen beschlossen. Dies ergibt eine theoretische Viertelung der Texturgrößen. Eine weitere Abstufung zwischen Tablet und Smartphone wurde aufgrund des Aufwandes nicht vorgenommen. Angenommen ein Element war in der vollen Höhe von 1080 Pixeln angelegt, so hat es auf Mobilgeräten noch eine Höhe von 540 Pixeln, was ziemlich gut mit den Displayauflösungen gängiger Mobilgeräte[23] im Einklang steht. Hier ist zu beachten, dass die unter Fußnote 23 gelisteten Displayauflösungen sich auf einer Ausrichtung im Portrait-Modus beziehen, die Anwendung aber für die Benutzung im Landscape-Modus ausgelegt ist.

Im Ergebnis dieser Optimierung verkleinerten sich die Texturvolumina wie folgt:

- Spielwelt: 42,2 MB auf 10,9 MB, Reduktion um 75%
- Schiffsmodell: 1,08 MB auf 0,19 MB, Reduktion um 82%
- Spielfigur Letti: 2,23 MB auf 0,54 MB, Reduktion um 76%

Eine Verkleinerung der Texturen der einzelnen Minispiele wurde nicht vorgenommen,

23 http://screensiz.es/

da diese ohnehin relativ kleine Werte zwischen 0.5 MB und 2 MB haben und das Ergebnis in keinem Verhältnis zum notwendigen Zeitaufwand gestanden hätte. Letzterer wäre immens, da bei jedem einzelnen Minispiel jedes Element in seiner Dimension angepasst und seiner Position auf dem Spielfeld entsprechend hätte korrigiert werden müssen.

4.2.2 Speicherlecks

Im Verlauf der Optimerung wurde festgestellt, dass mit längerer Spieldauer die Performance aus nicht ersichtlichen Gründen einbricht. Dies wurde um so deutlicher, je mehr der verschiedenen Minispiele geladen und gespielt wurden. Die Vermutung eines Speicherlags lag nahe. Eine Timeline-Aufzeichnung mit den Chrome DevTools und der immer weiter ansteigende Verlauf der Speicherkurve erhärtete diesen Verdacht.

Daraufhin kam die Heap-Snapshot-Funktion zum Einsatz, um das Problem zu analysieren und einzugrenzen. Zunächst wurde ein Speicher-Abbild vor dem ersten Start des Spiels aufgenommen, anschließend das Spiel gespielt, beendet, und ein weiteres Abbild erstellt. Im Anschluss wird das Spiel erneut gespielt und nach dem Beenden ein drittes Abbild gemacht. Diese werden anschließend verglichen, die Abbildung 10 soll das verdeutlichen.

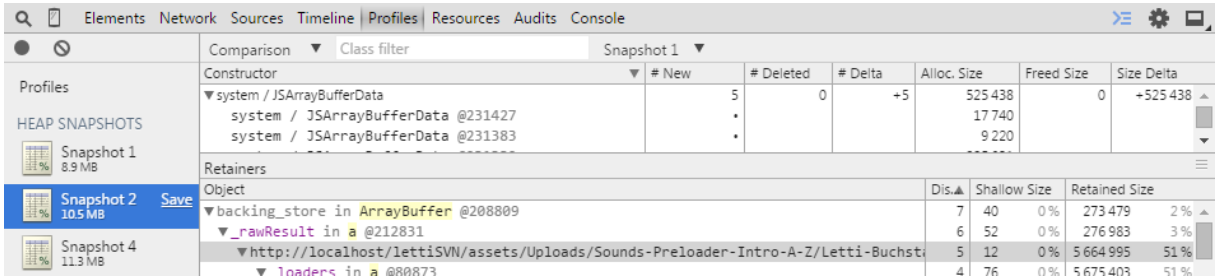

Abbildung 10: Heap-Snapshots zeigen Unterschied zwischen den Spielen

Die Ansicht zeigt den Vergleich von Snapshot 1 (vor dem ersten Spiel) mit Snapshot 2 (nach dem ersten Spiel). Exemplarisch wird eine Klasse gezeigt, deren Speicherverbrauch um 500 kB zugenommen hat. Die Prüfung des damit verbundenen Heap-Objektes ergab, dass es sich um die Audio-Datei handelt, die vor jeden Minispiel die Herkunft des Buchstabens erklärt. Weitere durch diesen Vergleich ermittelte "überflüssige" Objekte im Speicher sind HTML-Elemente, die nach dem Beenden eigentlich nicht mehr dargestellt werden.

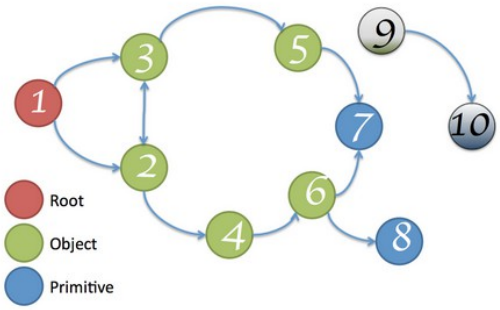

Abbildung 11: Speicher-Graph in Javascript, Quelle: [DevCPrf]

In Javascript kann Speicher nicht explizit freigegeben werden, zuständig für das Bereinigen nicht mehr genutzter Speicherbereiche ist ein Garbage Collection Algorithmus. Die Speicherverwaltung wird durch einen Graph wie in Abbildung 11 realisiert, in dem die Referenzierung einzelner Speicherelemente, beginnend von einem festen Ausgangsknoten, abgebildet wird. Jeder Knoten repräsentiert ein solches Element, also eine Speicherprimitive (Zahl, String oder Boolean) oder ein Objekt, jede Verbindung der Knoten eine Variablenreferenz auf ein Element. Ist nun ein Element nicht von der Wurzel aus erreichbar, besteht keine relevante Referenz mehr zu ihm und sein Speicher kann von der Speicherbereinigung freigegeben werden. Im Beispiel hält zwar Element 9 noch eine Referenz auf Element 10, jedoch existiert keine Referenz mehr auf Element 9, es sein Speicher kann daher bereinigt werden.

Um das Problem zu lösen, wurde im Code mehr auf Dereferenzierung geachtet, außerdem wurde nach dem Beenden des Spiels alle HTML-Objekte aus dem Spielbereich entfernt und von ihren Events bereinigt. Die Audio-Objekten müssen aufgrund der genutzten Technik gesondert freigegeben werden, siehe Kapitel .

Zuletzt muss gesagt werden, dass es sich hier nicht um eine Mobilgerät-spezifische Optimierungsmöglichkeit handelt. Jedoch erlangt sie aufgrund der limitierten Speicher-Ressourcen auf diesen Geräten größere Bedeutung. Bezeichnend hierfür ist schon, dass dieses Problem in der Anwendung auf Desktop-Geräten gar nicht erkannt wurde.

4.3 Multimedia

4.3.1 Audio

Ein essentieller Bestandteil des Spiels ist die Nutzung der Audioausgabe. Auf dem Desktop wurde hierfür das `<audio>`-HTML-Element genutzt. In der Desktop-Version funktioniert dies zufriedenstellend, jedoch ist dieses Element auf mobilen Geräten mit einigen Einschränkungen belastet, welche es für den gegebenen Anspruch praktisch unbrauchbar macht. Diese Einschränkungen unterscheiden sich teilweise zwischen den Mobil-Betriebssystem bzw. Browser-Versionen, es trifft nicht jede Einschränkung auf jeden Browser zu. Folgende Einschränkungen[24] sind problematisch:

- das `preload`-Attribut bzw. Preloading an sich wird nicht unterstützt, das Laden von Audio geschieht erst beim ersten Abspielen. Damit wird Abspielen zum richtigen Zeitpunkt ohne Latenz, zumindest beim ersten Versuch, unmöglich.
- Es kann nur ein `audio`-Element gleichzeitig abgespielt werden. Dies ist problematisch, weil die Spielwelt neben einer Hintergrundmelodie auch Elemente mit akustischer Ausgabe enthält, außerdem die Minispiele selbst Soundeffekte benutzen und die Spielfigur Letti eine Sprachausgabe hat.
- Das Abspielen kann nur durch ein User-Event gestartet werden, also im Kontext eines Click- oder Touch-Events. Das behindert gezieltes, gewollt zeitverzögertes Abspielen beispielsweise durch einen `setTimeout`-Callback, oder durch die Spiel-Logik.
- Audio kann nicht in Schleife abgespielt werden. Dies wird für die Hintergrundmelodie benötigt.
- Die Lautstärke einzelner Audio-Elemente kann nicht separat geändert werden. Dadurch kann die Lautstärke der Hintergrundmelodie in bestimmten Situationen nicht abgesenkt werden und stört, beispielsweise bei den einführenden Erklärungen der einzelnen Minispiele.

Die Lösung dieser Probleme liegt in der Nutzung der **Web Audio API**. Es handelt sich hierbei um eine Standardisierung, welche mittlerweile eine breite Browser-Unterstützung erfährt[25] und den oben aufgeführten Einschränkungen nicht unterworfen ist. Die Möglichkeiten dieser API[26] gehen weit über das Abspielen von

24 [Apl12]
25 http://caniuse.com/#feat=web-animation
26 [W3WebA]

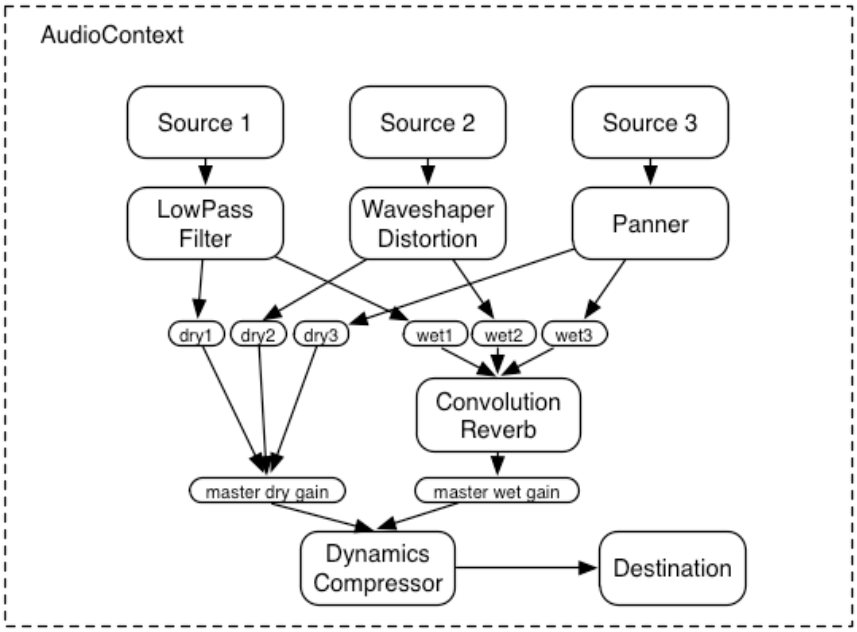

Abbildung 12: Beispiel für einen Audio Routing Graph, Quelle: [W3WebA]

Sounds hinaus. Ihr grundlegendes Prinzip ist die Erstellung eines Audio-Routing-Graphs, in dem eine Audio-Quelle durch verschiedene Bearbeitungsknoten zur Ausgabe geleitet wird. Abbildung 12 zeigt ein komplexer Beispiel mit mehreren Quellen und Bearbeitungsknoten.

Für die Lösung der vorliegenden Probleme wird diese Komplexität jedoch nicht benötigt. Es reicht ein Graph nach Schema der Abbildung 13. Der zwischen Quelle und Ziel eingefügte GainNode dient der Lautstärkeregelung. Auf ein Quellcode-Beispiel wird an dieser Stelle verzichtet, da für die Anwendung der Web Audio API ein Framework verwendet wird, siehe Kapitel 4.5.2.

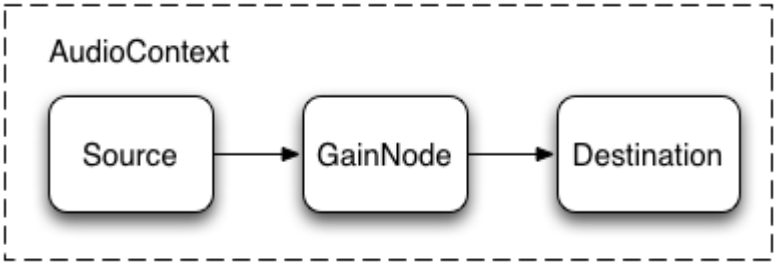

Abbildung 13: Audio-Graph für das Spiel, Quelle: [H5RoA13]

Eine kleine Einschränkung gibt es auch noch mit diesem Ansatz. Auf iOS-Geräten ist das gesamte Audio lautlos, bis es zum ersten Mal durch ein User-Event aktiviert wird. Dieses Problem wurde gelöst, indem ein auf dem Startbildschirm der Anwendung ein Play-Button plaziert wurde, welcher zur Kontrolle den Hintergrundsound startet und damit Audio auch für iOS aktiviert. Nach dieser einmaligen Aktion kann mit der API Sound zu beliebigen Zeitpunkten auf allen Geräten abgespielt werden.

4.3.2 Video

Zu Beginn des Spieles, genauer gesagt nur beim ersten Start, wird ein kurzes Video abgespielt, was dem Spieler die Geschichte erklären soll. Dieses wird durch das HTML-`<video>`-Element realisiert und war zu Anfang im Format 720p, also der Auflösung 1280x720, eingebunden. Jedoch zeigte dieses Video mit einer relativ hohen Bitrate von über 2800kBit/s selbst auf dem Nexus7 als leistungsstärkstes Gerät Ruckler, auf den Smartphones war die Performance noch schlechter. Da andere, native Apps, auf dem Gerät dieses Video ohne Probleme wiedergeben konnten, und dort selbst Videos mit FullHD-Auflösung problemlos waren, liegt die Vermutung nahe, dass das Video-Playback in Browsern noch Optimierungspotenzial besitzt, das Problem also beim Browser und nicht bei der Video-Datei zu suchen ist.

Um auch die Abspielbarkeit in Browsern zu gewährleisten, wurde das Video in ein Format mit geringerer Auflösung und Bitrate konvertiert. Auch in Anbetracht der in Kapitel 4.2.1 bechriebenen Datenvolumen- und Bandbreitenlimitierung bei Mobilgeräten wurde eine Auflösung gewählt, die deutlich unterhalb der Auflösung selbst von Smartphones liegt, 402x226. Diese ungewöhnlichen Zahlen ergaben sich aus Problemen mit dem Konvertierungstool, welche der Autor dieser Arbeit nicht zu verantworten hatte, sie machte jedoch keine Schwierigkeiten beim Abspielen. Das Ergebnis war eine auf 480kbps reduzierte Bitrate, und damit verbunden eine Dateigrößenreduzierung von 8,74 MB auf 1,49 MB.

Die im vorangegangenen Kapitel genannten Limitierungen treffen bei iOS in ähnlicher Form auch auf das `video`-Element zu[27]. So ist hier ebenfalls ein Start aus einem User-Event heraus notwendig, und das Video wird nicht vorgeladen. Im Gegensatz zum Audio stellt dies aber kein unüberwindbares Problem dar, da es wie eingangs erwähnt nur einmalig abgespielt werden muss. Das folgende Code-Beispiel zeigt, wie die für den Start notwendige Steuerung anfangs eingeblendet und beim Start wieder versteckt wird.

```
1    $('video').attr('controls', 'true');
2    $('video').on('play', function(){
3      $(this).removeAttr('controls');
4    });
```

27 [Apl12]

4.4 **Vereinfachung der Spielwelt**

Trotz der in Kapitel 4.2.1 ergriffenen Maßnahmen ließen sich die durch Speichermangel ausgelösten Abstürze nicht komplett eliminieren. Insbesondere das ältere iPad 2 mit seiner geringeren Arbeitsspeicher-Ausstattung fiel hierbei auf. Um das Problem mittels Ausschlussverfahren eingrenzen zu können, wurden einzelne Elemente mit der CSS-Eigenschaft `display: none` aus dem Renderbaum entfernt und das Ergebnis getestet. Es stellte sich heraus, dass beim kompletten Ausblenden der rotierbaren Spielwelt (Abbildung 1) die Abstürze ausblieben.

In Anbetracht der trotz reduzierter Texturgrößen immer noch bestehenden Komplexität des HTML- und CSS-Gerüsts der Spielwelt und des optischen Qualitätsverlustes wurde auf eine weitere Größenreduktion der Texturbasis verzichtet. Auch aufgrund des knappen Zeitbudgets wäre eine weitere Reduktion mit dem Risiko verbunden, dass dies das Problem trotzdem nicht lösen könnte, und die Ursache nicht allein im Texturvolumen zu finden sei. Stattdessen wurde in Absprache mit den Gestaltern der Welt ein radikal vereinfachter Entwurf erstellt, welcher nur noch einen Bruchteil an notwendigem Code und an Texturgrafiken (2,33MB) verwendete. Zwar musste hierfür der Javascript-Code für die Welt noch einmal umfassend angepasst werden, jedoch blieb dieser Aufwand in einem vertretbaren Rahmen. Der folgende Screenshot zeigt die für Mobilgeräte vereinfachte Welt. Die Buchstabenkacheln sind horizontal angeordnet und können mit einer Wischbewegung nach links und rechts verschoben werden.

Abbildung 14: Vereinfachte Spielwelt für Mobilgeräte

4.5 Praktische Erwägungen

4.5.1 Unterscheidung Desktop-/Mobilgerät

Die Lösung der im Kapitel 4 erfassten Probleme erforderte am manchen Stellen eine Unterschiedungsmöglichkeit, ob das Spiel gerade auf einem Mobil- oder auf einem Desktop-Gerät ausgeführt wird. So sind die Optimierungen der Animationen und der Audio-Problematik auf beiden Gerätekategorien sinnvoll und werden genutzt, im Gegensatz dazu sollen die mobil-spezifische Spieltwelt, die qualitativ verringerten Texturen und das Video nur auf entsprechenden Geräten genutzt werden.

Die Identifizierung unterschiedlicher Geräte wurde anhand des User Agent String des Browsers durchgeführt. Hierfür wurde auf die OpenSource-Bibliothek isMobile[28] zurückgegriffen, welche sich nach eine kurzen Sichtung des Quellcodes als übersichtlich und ausreichend herausstellte. Beim ersten Aufruf der Spiele-Website wird, bei Erkennung eines Mobilgerätes, der Spieler gefragt, ob er die entsprechende Variante nutzen möchte, seine Entscheidung wird in einem Cookie gespeichert. Dieser Cookie dient anschließend als Entscheidungsgrundlage, welcher Javascript-Code clientseitig genutzt und welcher HTML- und CSS-Code serverseitig geliefert wird. Letzteres wurde gemacht um um für die jeweilige Gerätekategorie irrelevante Daten gar nicht erst zu übertragen und damit Ladezeit und Datenvolumen zu sparen.

Die Anwendung von isMobile sieht beispielsweise so aus:

```
1   if (isMobile.any) {
2     $.cookie('useMobile', 'true', {expires: 365, path: '/'});
3   }
4   if (isMobile.android.device) {
5     //Android-spezifischer Code möglich
6   } else if (isMobile.apple.device) {
7     //iOS-spezifischer Code ebenso
8   }
```

An dieser Stelle muss gesagt werden, dass die Identifizierung anhand es User Agents, die sogenannte Browser Detection, nicht immer die beste Lösung ist. Der User Agent eines Browsers kann mit entsprechender Fachkenntnis und geringem Aufwand beliebig gesetzt werden, außerdem benötigt die Zuordnungsliste, welcher String zur Kategorie Mobilgerät gehört, beständige Pflege. Da in diesem Fall jedoch kein sicherheitsrelevanter Aspekt mit dieser Technik verbunden ist und im schlimmsten Fall ein Mobilgerät die Desktop-Version geliefert bekommt, ist dieser

28 https://github.com/kaimallea/isMobile

Ansatz vertretbar. Der Königsweg wäre die sogenannte Feature Detection, welche anhand des (Nicht-)Vorhandenseins eines Features, beispielsweise eine bestimmte Javascipt API oder ein HTML-Element, dieses benutzt oder ansonsten einen Workaround wählt. Jedoch ist es nicht ohne Weiteres möglich einen Aspekt wie Performance aus dieser Perspektive zuverlässig zu ermitteln.

4.5.2 Nutzung von Framework

Zur Umsetzung der in den vorangegangenen Kapiteln zusammengetragenen Problemlösungen wurde an mehreren Stellen auf Frameworks zurückgegriffen. Ihre Nutzung stellt allgemein einen Zugewinn an Produktivität dar, da man sich, allerdings abhängig vom Entwicklungsstand, meist auf eine gute Codebasis stützen kann, welche viel elementares Programmieren erspart. Auch haben die Entwickler von Frameworks meist vorhandene Browserbugs und Implementierungsunterschiede berücksichtigt und dem Nutzer bleibt eine solche Arbeit erspart.

Nachteilig daran ist, dass beim Arbeiten mit Frameworks die Gefahr besteht, Grundlagen zu verlernen bzw. gar nicht erst zu erlernen. Dies kann bei einer Nichtverfügbarkeit eines gewohnten Frameworks ein Nachteil sein.

Problematisch kann der Einsatz eines Frameworks auch werden, wenn dieses selber Bugs enthält, inkonsistentes Verhalten zeigt oder schlecht dokumentiert ist. Das Verstehen und die Fehlersuche in fremden Code gestaltet sich deutlich schwieriger als bei der eigenen, vertrauten Programmierweise.

Der Einsatz der beiden folgenden Frameworks zeigte solche Probleme jedoch nicht. Beide verfügen über eine gute Dokumentation, und insbesondere das Greensock Animation Platform Framework über eine aktive Forums-Gemeinschaft, an der sich auch die Entwickler selbst stark beteiligen und gute Hilfestellungen und Erklärungen liefern.

Greensock Animation Platform

Das Greensock Animation Platform Framework, kurs GSAP, ist dem Namen entsprechend ein primär für Animationen entwickeltes Framework. Neben einfachen Animationsmöglichkeiten bietet es gezielte Steuermöglichkeiten einer laufenden Animation wie Pausieren, rückwärts abspielen und die Änderung der Abspielgeschwindigkeit. Ein weiteres vielseitiges Werkzeug des Framework ist die

Timeline, welche es sehr vereinfacht Aimationen zu staffeln. Die Vielzahl der Features kann an dieser Stelle nicht dargestellt werden, auf der Homepage gibt es eine ausführliche Präsentation[29].

GSAP war zu Beginn dieser Arbeit bereits im Spiel im Einsatz, jedoch wurden dessen Möglichkeiten nicht ausreichend umgesetzt. Der Entwickler wirbt zwar mit sehr guter Performance, diese setzt jedoch eine korrekte Anwendung voraus. Bei genauerem Lesen der Dokumentationen wurde festgestellt, dass beide im Kapitel 4.1 ermittelten Problemlösungen hiermit angewendet werden können. Teilweise waren sie unbewusst schon in Verwendung, so benutzt GSAP die requestAnimationFrame API als Zeitgeber, jedoch wurde das Framework für viele Animationen gar nicht verwendet, obwohl es möglich gewesen wäre. An anderer Stelle wurde GSAP zwar genutzt, jedoch damit CSS-Eigenschaften wie left oder top animiert, wo Optimierungen nicht greifen können.

Das folgende Beispiel zeigt, wie mit GSAP `element` in einer Sekunde um 500 px in y-Richtung bewegt und gleichzeitig um 50° gedreht wird. Am Ende der Animation wird eine anonyme callback-Funktion aufgerufen.

```
1    TweenMax.to(element, 1, {
2      y: 500, rotate: 50, onComplete: function(){
3          console.log('complete');
4      }
5    });
```

Die notwendige Transform-Matrik berechnet das Framework automatisch und animiert das Element mittels der CSS-Eigenschaft `transform: matrix3d()`, so dass die Animation mit einem eigenen Composite-Layer durchgeführt wird.

Es wurde also die vorhandene Codebasis darauf hin optimiert GSAP konsequent auszureizen. In erster Linie bestand der Aufwand darin die richtigen CSS-Eigenschaften für Animationen zu nutzen, weiterhin wurden vorhandene, selbst programmierte Animationskonstrukte durch die Nutzung des Frameworks ersetzt.

CreateJS Framework

Die Nutzung der Web Audio API aus Kapitel 4.3.1 ist relativ komplex. Da für das Abspielen von Sounds nur ein kleiner Teil der Möglichkeiten dieser API benötigt wird, wurde ein Framework gesucht, welches diesen Teil gut umsetzt. Eine Recherche

29 https://greensock.com/jump-start-js

schränkte die Auswahl auf die beiden Frameworks howler.js[30] und SoundJS[31] als Teil des CreateJS-Framework[32] ein. Teil von CreateJS ist auch PreloadJS[33], welches im Rahmen dieser Recherche entdeckt wurde und weitere Probleme des Spiels außerhalb dieser Arbeit lösen konnte. Da SoundJS gut mit PreloadJS zusammenarbeitet und das Vorladen von Audio zwingend notwendig ist, fiel die Wahl eines Audio-Framework schließlich auf dieses.

Ein weiteres wichtiges Kriterium war auch die aus Kapitel 4.2.2 hervorgegangene Notwendigkeit Audio wieder aus dem Speicher entfernen zu können. Diese Funktion ist gegeben[34]. Eine Prüfung mit zwei Heap-Snapshots bestätigt, dass nach dem Entfernen der Speicher tatsichlich durch Garbage Collection freigegeben werden kann.

30 http://goldfirestudios.com/blog/104/howler.js-Modern-Web-Audio-Javascript-Library
31 http://www.createjs.com/SoundJS
32 http://www.createjs.com
33 http://www.createjs.com/PreloadJS
34 http://www.createjs.com/Docs/SoundJS/classes/Sound.html#method_removeSound

5 Fazit

5.1 Zusammenfassung

Mit dieser Arbeit konnnte die gesetzte Zielstellung, das gegebene Spiel auch auf mobilen Geräten performant zu machen, erreicht werden. Die im Rahmen der Testumgebung durchgeführte Analyse machte einige Probleme deutlich, die sich durch gezielte Optimierung beseitigen ließen. Das erreichte Spielerlebnis der Minispiele unterscheidet sich nicht mehr grundlegend von dem der Desktop-Version. Andere Gegebenheiten wie die Touchscreen-Bedienung erwiesen sich nicht als nachteilig.

Mehrere Problembereiche wurden erfolgreich optimiert. Mit dem größten Umsetzungsaufwand belastet, aber auch dem größten Performancegewinn verbunden, war die Verbesserung der Animationen, genauer gesagt die Anwendung der im Kapitel 4.1.2 erlangten Erkenntnis, dass die Beschränkung einer Animation auf bestimmte CSS-Eigenschaften einen deutlichen Unterschied ausmacht. Einen großen Beitrag zur Vereinfachung leistet hier das GSAP-Framework aus Kapitel 4.5.2. Weiterhin war die Umstellung der Audio-Ausgabe auf die Web Audio API auch für die PC-Version ein Gewinn, da diese API wesentlich mehr Möglichkeiten bietet als das `<audio>`-Element.

Die auf Speicherlecks zurückzuführenden Abstürze konnten durch die Optimierung des Programmcodes ebenfalls größtenteils beseitigt werden. Nur die beiden ältesten Geräte der Testumgebung, das iPad2 und das Sony Xperia Pro, beide mit lediglich 512 MB Arbeitsspeicher, verzeichnen weiterhin stellenweise Ruckler und vereinzelte speichermangelbedingte Abstürze. Diese Einschränkung wird jedoch durch die relativ schnelle Weiterentwicklung des Marktes für mobile Geräte zukünftig an Bedeutung verlieren.

Eine notwendiger Kompromiss ist allerdings die im Kapitel 4.4 beschriebene radikale Vereinfachung der Spielwelt. Hiermit geht etwas vom Flair des spielerischen Gesamtkonzepts verloren, jedoch beeinflusst das nicht den Lerneffekt, welcher vermittelt werden soll.

Als insgesamt von geringer Bedeutung erwies sich die im Kapitel 4.2.1 erläuterte Reduktion der Texturgrößen. Das ist in erster Linie dadurch begründet, dass der

Bereich mit dem deutlichsten Gewinn, die Spielwelt, letztlich wie im vorigen Absatz erwähnt, auf einem anderen Weg optimiert wurde.

Die im Kapitel 4.5.2 beschriebenen Frameworks erwiesen sich bei korrekter Nutzung ihrer Möglichkeiten als sehr hilfreich und zuverlässig und erleichtern den Umgang mit komplexen Techniken wie vielgestaltigen Animationen und die Nutzung der Web Audio API erheblich.

Leider kann durch die Begrenzung der Testumgebung auf die Betriebssysteme Android und iOS keine Aussage zur Funktionstüchtigkeit auf Geräten mit Windows Phone oder dem Blackberry-Betriebssystem getroffen werden, insofern ist auch dieser Punkt noch eine mögliche Einschränkung.

5.2 **Ausblick**

Mit der zu erwartenden Zunahme der Leistungsfähigkeit mobile Geräte kann es in absehbarer Zukunft möglich sein, dass die PC-Spielwelt auch mobil realisiert werden kann. Um der großen und im Rahmen dieser Arbeit nicht testbaren Vielfalt an Geräten dennoch gerecht zu werden, wurde in der Anwendung ein Kontaktformular eingerichtet, mit dem die Benutzer detailliertes Feedback zu eventuellen Problemen geben können, welches als Grundlage für weitere Verbesserungen dienen soll.

Die Entwicklung der Browser-Software ist ebenfalls fortschreitend und durch Implementierung neuer Funktionalitäten ergeben sich Möglichkeiten für weitere Spiel-Features, die insbesondere für Mobilgeräte von Bedeutung sein können. So gibt es beispielsweise schon zum aktuellen Zeitpunkt eine breite Unterstützung[35] für das HTML cache manifest, mit dem sich das Spiel als Offline-App, also ohne Notwendigkeit eines Internetzugangs, realisieren liese.

Abschließend kann gesagt werden, dass die kontinuierliche Weiterentwicklung von Hard- und Software dem HTML5-Bereich und den mobilen Geräten viele Potentiale eröffnet, die nicht nur für den Bereich Spiele relevant sein werden.

35 http://caniuse.com/#feat=offline-apps

Quellenverzeichnis

[HTML11] Peter Krömer: *HTML5 : Webseiten innovativ und zukunftssicher* Open Source Press, München, 2. Auflage, 2011

[DatWe03] Stefano Ceri, Piero Fraternali, Aldo Bongio, Marco Brambilla, Sara Comai, Maristella Matera: *Designing data-intensive Web applications* Morgan Kaufmann Publishers, San Francisco, 2003

[MDM12] Heinrich Kersten, Gerhard Klett: *Mobile Device Management* mitp, Heidelberg, München, Landsberg, Frechen, Hamburg, 2012

[WikDe15] Wikipedia Deutsch: *Extensible Hypertext Markup Language* http://de.wikipedia.org/w/index.php?title=Extensible_Hypertext_Markup_Language&oldid=137834612 abgerufen am 20.02.2015

[WikDe12] Peter Krömer, JanRieke: *Datei:HTML5-Spezifikations-Übersicht.svg*, 11.04.2012 http://de.wikipedia.org/wiki/Datei:HTML5-Spezifikations-%C3%9Cbersicht.svg#filehistory abgerufen am 22.02.2015

[WikEn15a] Wikipedia Englisch: *Extensible Hypertext Markup Language* http://en.wikipedia.org/w/index.php?title=HTML5&oldid=647372227 abgerufen am 20.02.2015

[WikEn15b] Wikipedia Englisch: *Mobile device* http://en.wikipedia.org/w/index.php?title=HTML5&oldid=647372227 abgerufen am 22.02.2015

[ARM14] ARM Limited: *ARM in Mobile*, 2014 http://www.arm.com/markets/mobile/index.php abgerufen am 21.02.2015

[Stat15] statista: *Marktanteile der mobilen Betriebssysteme am Absatz von Smartphones in Deutschland von Oktober bis Dezember in den Jahren 2013 und 2014*, 2015 http://de.statista.com/statistik/daten/studie/198435/umfrage/marktanteile-der-smartphone-betriebssysteme-am-absatz-in-deutschland/ abgerufen am 21.02.2015

[DevMoz14] Mozilla Developer Network: *Performance fundamentals,* 2014 https://developer.mozilla.org/en-US/Apps/Build/Performance/Performance_fundamentals abgerufen am 21.02.2015

[DevGo14] Ilya Grigori: *Erstellung der Rendering-Baumstruktur, Layout, und Paint*, 18.09.2014
https://developers.google.com/web/fundamentals/performance/critical-rendering-path/render-tree-construction
abgerufen am 12.02.2015

[H5Ro13] Paul Lewis, Paul Irish: *High Performance Animations*, 07.11.2013
http://www.html5rocks.com/en/tutorials/speed/high-performance-animations/
abgerufen am 12.02.2015

[H5RoA13] Boris Smus: *Getting Started with Web Audio API // Changing the volume of a sound*, 29.10.2013
http://www.html5rocks.com/en/tutorials/webaudio/intro/#toc-volume
abgerufen am 17.02.2015

[Chro14] Tom Wiltzius, Vangelis Kokkevis & the Chrome Graphics team: *GPU Accelerated Compositing in Chrome*, Mai 2014
https://www.chromium.org/developers/design-documents/gpu-accelerated-compositing-in-chrome
abgerufen am 14.02.2015

[DevChro] Chrome Developer, *Google Chrome for iOS*
https://developer.chrome.com/multidevice/ios/overview
abgerufen am 22.02.2015

[DevCPrf] Chrome Developer, *JavaScript Memory Profiling // Retained size*
https://developer.chrome.com/devtools/docs/javascript-memory-profiling#retained-size
abgerufen am 16.02.2015

[W3WebA] Paul Adenot, Chris Wilson: *Web Audio API*, 18.02.2015
http://webaudio.github.io/web-audio-api/#modular-routing
abgerufen am 23.02.2015

[PcGHw12] Raffael Vötter: *Der 24-Fps-Mythos: Warum 24 Frames in Spielen nicht flüssig ist*, 09.11.2012
http://www.pcgameshardware.de/Spiele-Thema-239104/Specials/Wann-laufen-Spiele-fluessig-1034704/
abgerufen am 15.02.2015

[DevOp13] Luz Caballero: *Better Performance With requestAnimationFrame*, 12.06.2013
https://dev.opera.com/articles/better-performance-with-requestanimationframe/
abgerufen am 11.02.2015

[W3tc13] James Robinson, Cameron McCormack: *Timing control for script-based animations*, 31.10.2013
https://dvcs.w3.org/hg/webperf/raw-file/tip/specs/RequestAnimationFrame/Overview.html
abgerufen am 11.02.2015

[Apl12] Apple Inc., *Safari HTML5 Audio and Video Guide - iOS-Specific Considerations*, 13.12.2012
https://developer.apple.com/library/safari/documentation/AudioVideo/Conceptual/Using_HTML5_Audio_Video/Device-SpecificConsiderations/Device-SpecificConsiderations.html#//apple_ref/doc/uid/TP40009523-CH5-SW1
abgerufen am 16.02.2015

Anlagen

Anlage A: Datenträger

Der beigelegte Datenträger hat den folgenden Inhalt:

- im Wurzelverzeichnis liegt diese Arbeit als PDF-Dokument
- **Code** enthält das komplette Spiel in seiner fertigen, mobil-optimierten Version. Im Unterordner `themes/letti/` befindet sich der selbst entwickelte Teil der Anwendung, die restlichen Ordner sind Teil des Silverstripe-Frameworks[36]
- **Quellen** enthält die abgespeicherten Internetquellen, die Unterordner sind entsprechend dem Quellenverzeichnis benannt
- **Tests** enthält die HTML-Dokumente aus Anlage B (Unterordner `requestAnimationFrame`) und Anlage C (Unterordner `Animation-left-vs-transform3d`)

Anlage B: Quellcode f. Animationsschleifentest

```
1    <!DOCTYPE html>
2    <html>
3        <head>
4            <style type="text/css">
5                body {padding: 10px;}
6                #box {
7                    width: 100px;
8                    height: 100px;
9                    margin-top: 10px;
10                   background: green;}
11           </style>
12
13           <script src="jquery-2.1.1.js"></script>
14           <script type="text/javascript">
15               $(function(){
16                   var type = 'sto',
17                       fps = 60,
18                       aniDist = 1000, //Animationsdistanz 1000px
19                       aniTime = 2000, //Animationszeit 2 Sekunden
20                       left = 0,
21                       callCount = 0,
22                       startTime = Date.now();
23
24                   function moveBox() {
25                       callCount++;
26                       if (left > aniDist) {
27                           console.log('moveBox() callCount with
     '+type+': '+callCount);
```

36 http://www.silverstripe.org/

```
28                          alert('moveBox() callCount with '+type+':
    '+callCount);
29                          return;
30                      } else {
31                          if (type == 'sto')
32                              setTimeout(moveBox, 1000 / fps);
33                          else if (type == 'raf')
34                              requestAnimationFrame(moveBox)
35                      }
36                      var ts = Date.now();
37                      left = aniDist * (ts - startTime) / aniTime;
38                      $('#box').css('margin-left', left + 'px');
39                  }
40
41                  $('input').click(function(){
42                      callCount = left = 0;
43                      startTime = Date.now();
44                      type = $(this).attr('id');
45                      moveBox();
46                  });
47              })
48          </script>
49
50      </head>
51
52      <body>
53          <input id="sto" type="button" value="setTimeout" />
54          <input id="raf" type="button"
    value="requestAnimationframe" />
55
56          <div id="box"></div>
57
58      </body>
59  </html>
```

Anlage C: Quellcode f. Animation unterschiedl. CSS-Eigenschaften

```
60  <!DOCTYPE html>
61  <html>
62      <head>
63          <style type="text/css">
64              body {padding: 10px;}
65              #box {
66                  position: relative;
67                  width: 150px;
68                  height: 150px;
69                  margin-top: 10px;
70                  transform-origin: top left;
71                  background: url("bernd.jpg") center center no-
    repeat;
72                  background-size: cover;}
73          </style>
74
75          <script src="jquery-2.1.1.js"></script>
76          <script type="text/javascript">
```

```
77              $(function(){
78                  var typeMove,
79                      timeStamp,
80                      left = 0,
81                      aniTime = 5000, //Animationszeit 2 Sekunden
82                      aniDist = 500; //Animationsdistanz 1000px
83
84                  function moveBox(timestamp) {
85                      if (left > aniDist) {
86                          return;
87                      } else {
88                          requestAnimationFrame(moveBox)
89                      }
90                      var ts = performance.now(),
91                          diff = aniDist * (ts - timeStamp) / aniTime;
92                      timeStamp = ts;
93                      left = left + diff;
94                      if (typeMove == 'left')
95                          $('#box').css('left', left + 'px');
96                      else if (typeMove == 'transform')
97                          $('#box').css('transform', 'translate3d(' +
    left + 'px, 0, 0)');
98                  }
99
100                 var typeScale,
101                     scale,
102                     startWidth = $('#box').width(),
103                     startHeight = $('#box').height();
104
105                 function scaleBox(timestamp) {
106                     if (scale > 2) {
107                         return;
108                     } else {
109                         requestAnimationFrame(scaleBox)
110                     }
111                     var ts = performance.now(),
112                         diff = 1 * (ts - timeStamp) / aniTime;
113                     timeStamp = ts;
114                     scale = scale + diff;
115                     if (typeScale == 'widthHeight')
116                         $('#box').css({width: startWidth * scale,
    height: startHeight * scale});
117                     else if (typeScale == 'scale')
118                        $('#box').css('transform', 'scale3d(' + scale
    + ',' + scale + ',1)');
119                 }
120
121                 $('input[value~=move]').click(function(){
122                     left = 0;
123                     $('#box').removeAttr('style');
124                     timeStamp = performance.now();
125                     typeMove = $(this).attr('id');
126                     moveBox();
127                 });
128                 $('input[value~=scale]').click(function(){
129                     scale = 1;
130                     $('#box').removeAttr('style');
131                     timeStamp = performance.now();
132                     typeScale = $(this).attr('id');
```

```
133                     scaleBox();
134                 });
135             })
136         </script>
137     </head>
138 
139     <body>
140         <input id="left" type="button" value="move left" />
141         <input id="transform" type="button" value="move
    transform" />
142         <input id="widthHeight" type="button" value="scale
    widthHeight" />
143         <input id="scale" type="button" value="scale transform" />
144 
145         <div id="box"></div>
146     </body>
147 </html>
```

Selbstständigkeitserklärung gem. § 22 Abs. 5 BPO

Hiermit versichere ich, Martin Benkenstein, dass ich die vorliegende Bachelorarbeit mit dem Titel

Performanceanalyse und -optimierung von Browseranwendungen für mobile Endgeräte am Beispiel eines HTML5 basierenden Browserspiels

selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfmittel verwendet habe. Die Stellen der Arbeit, die in Wortlaut oder Sinn aus anderen Arbeiten entnommen wurden, sowie Bilder und Tabellen aus fremden Werken, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Weiterhin ist diese Arbeit noch nicht veröffentlicht oder als Prüfungsleistung vorgelegt worden.

_____ _____

Ort, Datum Unterschrift