Bachelorarbeit

Editor zur Erstellung interaktiver Inhalte ohne Programmierkenntnisse

von :	Friedrich Wessel
Angefertigt am:	Institut für Informatik
	Freie Universität Berlin
in Zusammenarbeit mit :	Happy Tuesday UG, Berlin
	Stenarts Ltd. – Stenarts Berlin
Betreut von:	Prof. Dr. Lutz Prechelt

Hiermit versichere ich, Friedrich Wessel, geboren am 03.10.1985 in Dresden, dass die vorliegende Bachelorarbeit von mir selbstständig verfasst wurde. Zur Erstellung wurden von mir keine anderen als die angegebenen Quellen und Hilfsmittel verwendet. Ich versichere, dass ich die Bachelorarbeit weder im In- noch Ausland bisher in irgendeiner Form als Prüfungsarbeit vorgelegt habe.

Berlin, den 30. Januar 2014

Unterschrift

Vorname, Nachname	Matrikelnummer

Inhaltsverzeichnis

Einleitung……..5
Vision
Struktur der Arbeit
Anmerkung zur Arbeit

„Was soll das sein?“ – Definition des Editors…….9
Ausgabe
Inhalt
Import von Inhalten
Wiederverwendung von Assets
Arbeitsfläche

„Was sind interaktive Inhalte?“ – Eine Definition……12
Definition: Interaktivität
Interaktivitätsgrad

„Welchen Inhalt erstellt man mit dem Editor?“ – Der Anwendungsfall……..16
„Kleiner Fuchs“
„Ria-Chibi-Camp“
„Haunted Island“

„Wer braucht den Editor?“ – Die Zielgruppe……17
Die Firmen
Happy Tuesday UG, Berlin
Stenarts Ltd. – Stenarts Berlin
Die Tester
Samuel Stephan - Stenarts Berlin
Kevin Blank - Happy Tuesday Berlin
Tino Englert - Happy Tuesday Berlin
Pauline Altmann - Freie Grafikdesignerin
Begründung der Auswahl

„Gibt es das nicht schon?“ – Die Softwaretests……..20
Software für Vergleichstests
Adobe Edge Animate CC
Hutongames Playmaker
Softwaretests
1. Nutzertest ➙Samuel Stephan, ➙„Adobe Edge Animate“
2. Nutzertest ➙Kevin Blank, ➙„Playmaker“ für Unity
Auswertung der Nutzertests

„Was muss der Editor können?“ – Das Featureset……..28
Nutzerinterviews
Ergebnis des 1. Interviews
Mögliche Anwendungsgebiete:
Beispielhafte Abläufe:
Benutzbarkeit
Workshops - „Präventiv Obduktion“
Ablauf
Zusammenfassende Featureliste
Anforderungen an den Editor

„Wie entwickelt man den Editor?“ – Das Softwaredesign……..37
Der Prototyp
Unity3D
Anmerkung zu Lizenzen
Einschränkungen für den Prototypen
Systemarchitektur
Timeline-Architektur
Flowgraph-Architektur
Behaviour-Tree-Architektur

„Funktioniert das auch?“ – Der Nutzertest……50
Beschreibung des Testfalls
Nutzertest: Samuel Stephan – EFI
Nutzertest: Pauline Altmann – EFI

„Wie geht es weiter?“ - Das Fazit……59

Danksagung

Einleitung

Am 31. Mai 1993 veröffentlichte das Magazin „*Newsweek*“ eine Ausgabe mit dem Thema „Interaktivität“. Unter dem Titel: »An interactive Life. It will put the world at your fingertips...« gab das Magazin einen Ausblick darauf, wie interaktive Inhalte in Zukunft aussehen könnten:

> *»...a huge amount of information available to anyone at the touch of a button, everything from airline schedules to esoteric scientific journals to video versions of off-off-off Broadway. (...) Watching a movie won't be a passive experience. At various points, you'll click on alternative story lines and create your individualized version of 'Terminator XII' (...) Videocamera owners could record news they see and put it on the universal network ... Viewers could select whatever they wanted just by pushing a button. (...) Instead of playing rented tapes on their VCRs, (...) [the customers] may be able to call up a movie from a library of thousands through a menu displayed on the TV.«* [1]

Zwanzig Jahre nach Veröffentlichung dieser Zukunftsvision sind viele der Vorhersagen für uns heute bereits Alltag geworden. Ganz selbstverständlich schlagen wir nahezu jede Information innerhalb von Minuten „im Netz“ nach. Wir buchen „online“ Flüge und Kinokarten. Plattformen wie „*YouTube*“ oder *„Vimeo“* ermöglichen jedem die Veröffentlichung selbstproduzierter Videos oder kompletter Serien. *„VideoOnDemand“* bezeichnet die Dienstleistung, welche die traditionelle Videothek noch nicht abgelöst hat, sich jedoch wachsender Beliebtheit erfreut und mittlerweile von vielen großen Unternehmen angeboten wird.[2] Für eine Zeit in der die Entwicklung des Internets ebenso wenig absehbar war, wie der Nachfolger der VHS, haben sich erstaunlich viele dieser Prognosen erfüllt.
Ein Punkt jedoch bleibt bis heute nahezu unberührt: Das Erlebnis alternativer Handlungsstränge eines Blockbusters wie „Terminator“. Dank des zusätzlichen Speicherplatzes auf DVD und Blue-Ray werden vereinzelt alternative Enden des Films verfügbar[3]. Vielfältige Interaktionspunkte (*„various points“*) zur Auswahl alternativer Handlungsstränge, mit dem Ziel einer personalisierten Version (*„individualized version“*) bleiben jedoch eine Zukunftsvision. Filme, sowie die Vielzahl „klassischer“ Medien, schöpfen bis dato den Mehrwert der technischen Möglichkeiten bei der Umsetzung auf digitalen Plattformen nicht aus. Die Bandbreite reicht von digitalen Kinderbüchern, Belletristik und Comics bis hin zu wissenschaftlichen Veröffentlichungen. Fast alle der genannten Inhalte werden heute zusätzlich digital publiziert und konsumiert. Die Inhalte werden jedoch wie vor 10 Jahren für die Druckausgabe konzipiert.

---

[1] Aus: JENS F. JENSEN - 'Interactivity' Tracking a New Concept in Media and Communication Studies Nordicom Review, 19(1), 1998, PDF, zitiert aus: Kantrowitz, Barbara et al., (1993) "An Interactive Life", in Newsweek, Mai 31. 1993
[2] Vergl. Studio des "House of Research", laut: http://www.online-videotheken-vergleich.com/entwicklung-umsatz-video-on-demand-2004-2012/ abgerufen am 09.11.2014
[3] Eine Liste von Filmen die auf DVD über alternativen Enden verfügen, ist bei Wikipedia zu finden: http://en.wikipedia.org/wiki/Alternate_ending#Examples_in_film

Technisch existieren keine Hürden, einen Roman für den digitalen Markt von Anfang an mit verschiedenen Handlungssträngen und Entscheidungspunkten zu versehen.
In analoger Form existieren solche Formen der Geschichtenerzählung seit den 1980er Jahren[4]. Der Leser setzt hierbei die Geschichte immer auf anderen Seiten des Buchs fort, je nachdem wie er möchte, dass sich der Protagonist der Geschichte verhält. Somit waren schon vor 20 Jahren einige Bücher interaktiver als der Großteil der heutigen E-Books.

Bei wissenschaftlichen Veröffentlichungen bleibt es meist dem Leser überlassen, notwendige Grundlagen nachzuschlagen. Solange die Masse der präsentierten Inhalte die Übersichtlichkeit gefährdet, wie es bei den auf zwei Dimensionen beschränkten Druckmedien der Fall ist, ist diese Herangehensweise verständlich. Bei der digitalen Präsentation können jedoch mit einfachen Mitteln viele Hintergrundinformationen auf wenig Raum untergebracht werden. So lässt sich durch das Verstecken und Anzeigen von Text interaktiv die Informationsdichte an die Bedürfnisse des Lesers anpassen.
Weitere Möglichkeiten der Informationsvisualisierung lassen sich auch mit Hilfe der Skalierung der Inhalte realisieren. Technologiespielereien wie die Website *„zoomquilt“* [5] (↘01_Zoomquilt.mp4), nutzen dieses Verfahren auf eindrucksvolle Weise. Hier wird immer weiter in ein Bild hinein gezoomt. In den Details manifestieren sich immer wieder neue komplette Bilder, in welche wiederum hineingezoomt werden kann.

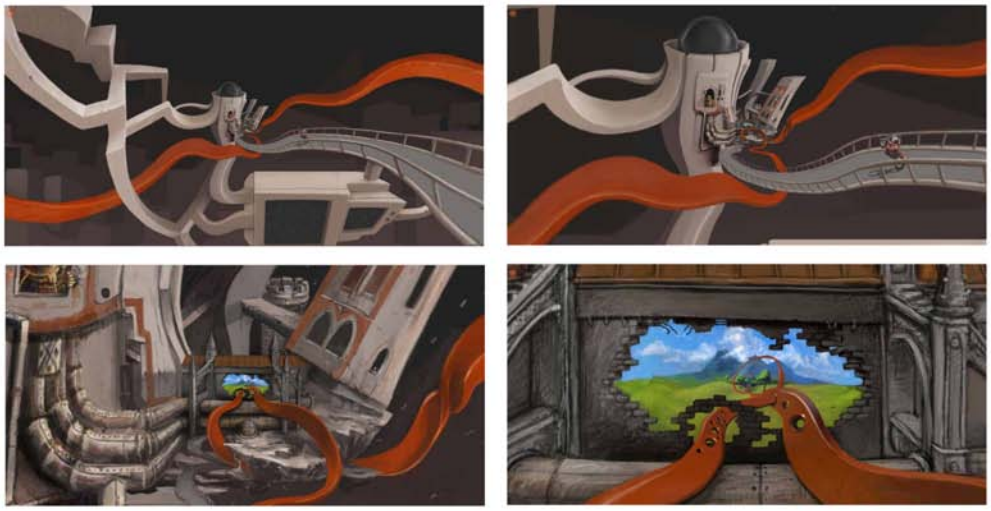


Abbildung 01: Screenshots von der Website zoomquilt.org

Tablets und Smartphones haben in den letzten Jahren Einzug in viele Haushalte gefunden, wo sie vielseitig und vermehrt zur Unterhaltung genutzt werden. Vielen Formaten ist es aber noch nicht gelungen den »Distributionsapparat in einen Kommunikationsapparat zu wandeln« wie es Berthold Brecht schon 1932 für das Radio gefordert hat.[6]

---

[4] Vergl. http://de.wikipedia.org/wiki/Spielbuch abgerufen am 09.11.2014
[5] http://www.zoomquilt.org/, abgerufen am 09.11.2014

[6] Bertolt Brecht, „Der Rundfunk als Kommunikationsapparat. Rede über die Funktion des Rundfunks“, Schriften zur Literatur und Kunst I., Frankfurt am Main: Suhrkamp Verlag 1967, S.132-140.

Die Konzeption der genannten Inhalte ist die Aufgabe von kreativen Autoren, Regisseuren und Illustratoren. Die Umsetzung erfordert jedoch fundierte Programmierkenntnisse und setzt folglich die Zusammenarbeit verschiedener Spezialistengruppen voraus. Hierdurch entsteht ein hohes Maß an Kommunikationsaufwand. Jegliche Änderungen am Inhalt, die nach Beginn der Produktion erdacht und veranlasst werden, verursachen hohe Kosten. Unvorteilhafterweise lassen sich diese Konzepte jedoch meistens erst nach der finalen Umsetzung richtig prüfen. Diese Faktoren behindern den kreativen Schaffensprozess und machen die Produktion von interaktiven Inhalten zu einer kostspieligen Unternehmung für hochgradig spezialisierte Teams.

Vision

Das Ziel ist die direkte Umsetzung einer Idee zu einem Produkt. Kurze Iterationsschleifen mit direkter Beurteilungsmöglichkeit der Ergebnisse schaffen ein Umfeld für kreativ experimentelles Arbeiten. Ermöglicht wird dies durch eine Software mit deren Hilfe interaktive Inhalte ohne Programmierkenntnisse erstellt werden können. Ein Werkzeug welches Kreative in die Lage versetzt, ihre Ideen eigenständig umzusetzen. Benötigt wird ein :

„Editor zur Erstellung interaktiver Inhalte ohne Programmierkenntnisse.“

Struktur der Arbeit

Die Fragen: *„Was kann man sich unter einem Editor zum Erstellen interaktiver Inhalte vorstellen?“*, und *„Was ist überhaupt interaktive Software?“* sind der Beginn einer langen Fragenkette, die dieser Titel aufwirft. Diese Fragenkette stellt den Leitfaden durch die Arbeit dar. Am Beispiel eines Texteditors werden die wichtigsten Begriffe und der Aufbau des Editors skizziert. Nachdem wir eine Vorstellung von dem Editor erlangt haben, folgt ein Ausflug zur Definition und Bewertung interaktiver Inhalte. Zu diesem Zeitpunkt kennen wir den Editor und die Eigenschaften des ausgegebenen Inhaltes. Im Kapitel *„Welchen Inhalt erstellt man mit dem Editor?“* wird das Ausgabeformat des Editors an einigen Beispielen verdeutlicht.
Jetzt stellt sich die Frage: „Wer braucht den Editor?“. Ist die Nutzergruppe bekannt, liegt die Frage „Gibt es das nicht schon?“ nahe. Ich nehme die Antwort vorweg: „Nein, gibt es nicht!“. Deswegen werden anschließend die Wünsche der Nutzer thematisiert. *„Was muss der Editor können?“* ist eine umfangreiche Frage, bei der das Featureset und die Bedienbarkeit des Editors erarbeitet werden. Die Antwort auf die Frage: *„Wie entwickelt man den Editor?“*, lautet: „Stück für Stück“. Begonnen wird mit dem Prototypen, der das Ergebnis dieser Arbeit darstellt. In diesem Abschnitt werden das Softwaredesign und die Entwicklungsmethoden beleuchtet. Mit dem ersten Prototypen widmen wir uns wieder den Nutzern und ihrer skeptischen Frage *„Funktioniert das auch?“*. Auf Basis der Nutzertests, wird diese Frage im abschließenden Kapitel beantwortet.

Kapitelübersicht:

- „Was soll das sein?“ - Der Editor

- „Was sind interaktive Inhalte?“ – Eine Definition
- „Welchen Inhalt erstellt man mit dem Editor?“ – Der Anwendungsfall
- „Wer braucht den Editor?“ – Die Zielgruppe
- „Gibt es das nicht schon?“ – Die Softwaretests
- „Was muss der Editor können?“ – Das Featureset
- „Wie entwickelt man den Editor?“ - Das Softwaredesign
- „Funktioniert das auch?“ – Der Nutzertest
- „Wie geht es weiter?“ – Das Fazit

Anmerkung zur Arbeit

Beispiele für interaktive Abläufe lassen sich meistens nur umständlich in Form von Text und Bild darstellen. Um die Ideen kurz zu skizzieren, wird in der gedruckten Fassung dieser Arbeit auf Screenshots und kurze Beschreibungen zurückgegriffen.
Auf der beigelegten CD befinden sich zusätzlich Videos, die viele Beispiele eindeutiger darstellen können. Ich verweise an den entsprechenden Stellen im Text kurz auf das Video.
Des Weiteren befindet sich auf der CD die Arbeit als PDF, in dem Videos direkt eingebettet, und Textstellen verlinkt sind, um die interaktiven Möglichkeiten heutiger Texteditoren für die digitale Distribution zu nutzen.
Für die Ansicht empfehle ich *„Adobe Acrobat Reader“* und einen aktuellen *„Flashplayer“*. Um nach dem Sprung zu einer verlinkten Textstelle zur Ausgangsposition zurück zu kehren, bietet *„Adobe Acrobat“* die Funktion *Previous view* bzw. das Tastenkürzel [Alt+Left Arrow].
Verweise werde ich wie folgt kennzeichnen:
➳interner Verweis
⤡Verweise auf Dateien auf der CD
⤢Verweise ins Internet

Im eingeschränkten Rahmen dieser Arbeit werden die Nutzertests und Interviews exemplarisch - und nicht in repräsentativem Umfang - erstellt. Vorrangige Ausrichtung dieser Arbeit ist es deswegen, eine Basis für weitere Studien zu erarbeiten. So ließe sich zukünftig eine repräsentative Anzahl an Nutzertest mit dem erstellten Prototypen durchführen.
Zur Erarbeitung des Themas arbeite ich mit zwei Firmen aus Berlin zusammen. Sowohl ➳Happy Tuesday als auch ➳Stenarts Berlin arbeiten im Schwerpunkt an der Erstellung digitaler interaktiver Applikationen. Ausgewählte Mitarbeiter bilden die wesentliche Nutzergruppe meiner Tests.

Die Vielfalt der Möglichkeiten für interaktive Inhalte ist groß. Darum gilt mein Fokus visuellen Medien mit geringen Datenmengen, bei gleichzeitig hohem Interaktivitätsgrad. Als Beispieldomänen konzentriere ich mich daher auf digitale Kinderbücher und Comics. Beide Bereiche bieten aufgrund der komprimierten Erzählstruktur die Möglichkeit für viele kurz hintereinander liegende Interaktionspunkte. Zudem ist die Menge des benötigten Materials vergleichsweise gering.

Aufgrund der ähnlichen Erzählform und Struktur werden sich viele der erarbeiteten Grundkonzepte später auf die hier nicht untersuchten Bereiche, wie Spielfilm oder interaktive Romane, übertragen lassen.

„Was soll das sein?“ – Definition des Editors

Textverarbeitungsprogramme wie *„Microsoft Word“* oder *„Pages“* von Apple sind weit verbreitet und nahezu jedem Computerbenutzer bekannt[7]. Diese sogenannten Text-Editoren ermöglichen die Erstellung, Modifikation und Ausgabe von Text. Anhand dieses bekannten Beispieles erläutere ich im Folgenden den Aufbau des zu erstellenden „Editors für interaktive Inhalte“. Zur besseren Lesbarkeit werde ich den Editor im folgenden mit *„EFI“* abkürzen.

Ausgabe

Bei einem Texteditor ist das Ziel das Drucken einer Textseite oder Versenden einer E-Mail. Bei EFI ist die Ausgabe ein eigenständiges Programm, wie zum Beispiel ein digitales Kinderbuch oder ein Rätselspiel das auf einem Computer oder iPad ausgeführt wird. Der mit dem „Ausdrucken“ vergleichbare Prozess wird in der Softwareerstellung „bauen“[8] genannt. So wird auch EFI zum Abschluss die erstellte Arbeit zu einer eigenständigen Software bauen.

Inhalt

Wie es bei *„Microsoft Word“* (engl. Wort) schon im Namen enthalten ist, werden im Texteditor Wörter verarbeitet. Diese bilden zusammen den Text und die Seite, die wir später ausdrucken können.
Interaktive Inhalte sind deutlich komplexer als Wörter. Ein Wort setzt sich aus einzelnen Buchstaben zusammen. In einer festen Reihenfolge angeordnet, bilden sie ein bestimmtes Wort, welches sich immer wieder verwenden lässt.
In EFI bezeichne ich solche zusammengesetzten und wiederverwendbaren Komponenten als *Assets*[9]. Die Einzelteile sind größtenteils Mediendaten wie Bilder, Filme und Musik.
Aufgrund des hohen Aufwandes können *Assets* nicht bei jeder Benutzung neu erstellt werden. Um die *Assets* wiederzuverwenden, werden sie bei EFI in einer Bibliothek bereitgestellt. Diese Bibliothek ist in EFI schnell zu erreichen.

Import von Inhalten

In der Bibliothek befindliche *Assets* werden nicht in EFI erstellt. Die Komplexität der Inhalte erfordert spezialisierte Programme. Bilder können zum Beispiel von einer Bildbearbeitungssoftware wie *„Adobe Photoshop“* erstellt und abgespeichert werden. Daten werden normalerweise auf der Festplatte des Computers in einem Ordner gespeichert. EFI wird

---

[7] Vergl.: Der Verbreitung von Office Programmen in Deutschland 2010: http://www.webmasterpro.de/portal/news/2010/01/25/verbreitung-von-office-programmen-openoffice-ueber-21.html - abgerufen: 20.11.2013

[8] Erstellungsprozess oder Build-Prozess (von englisch *to build* „bauen“) bezeichnet in der Programmierung einen Vorgang, durch den ein fertiges Anwendungsprogramm automatisch erzeugt wird. Laut Wikipedia: http://de.wikipedia.org/wiki/Erstellungsprozess - Abgerufen 20.11.2013

[9] Etwas beliebiges von materielem Wert oder Nützlichkeit - nach : http://what-is-the-definition-of.com/asset.html - abgerufen 20.11.2013

einen speziellen Bibliotheksordner besitzen. Alle darin abspeicherten Daten, erscheinen in der Bibliothek als *Asset* und können beliebig oft dupliziert werden.

Wiederverwendung von *Assets*

In einem Textverarbeitungsprogramm können wir Wörter zu Sätzen und ganzen Absätzen zusammensetzen. Absätze lassen sich duplizieren und mehrfach im Text verwenden.
Nach dem gleichen Muster lassen sich in EFI *Assets* miteinander zu neuen *Assets* kombinieren. Zum Beispiel kann man verschiedene Bilder zueinander anordnen, und die gesamte Anordnung wieder als *Asset* Speichern. In Zukunft kann diese Anordnung von Bildern direkt aus der Bibliothek dupliziert werden. Diese Kombination von *Assets* wird in EFI *Prefab* (von dem engl. Pre-Farbricated - vorproduziert) genannt.

Was aber wenn sich in einem Satz den wir soeben mehrfach kopiert haben, ein Tippfehler eingeschlichen hat ? Im Texteditor müssten wir jetzt mit Hilfe der Funktion „Suchen“ an alle Stellen springen und den Fehler beheben. In EFI sehen alle duplizierten Assets immer aus, wie die Ursprungsvariante in der Bibliothek. Sollen alle Duplikate angepasst werden, muss nur das Asset in der Bibliothek angepasst werden und alle Duplikate werden automatisch aktualisiert. Ein solches Verhalten wird als „Referenzieren“ bezeichnet, da eingesetzte *Assets* lediglich eine Referenz auf ein Bibliothekselement sind.

Arbeitsfläche

Bleibt noch die Arbeitsfläche, auf der wir aus unseren *Assets* unser Ausgabeprodukt erstellen. Ein Texteditor verfügt über eine Arbeitsansicht, in welcher Text geschrieben und bearbeitet wird. Die Seiten- oder Druckansicht dient der Vorschau des Ausgaberesultats - der zu druckenden Seite.

In EFI wird es ebenfalls eine Arbeitsansicht geben. Da die Inhalte bewegt sind und Interaktion mit dem Nutzer (Publikum) zulassen, wird hier von einer ›Stage‹ (engl. Bühne) gesprochen.
Die Arbeitsfläche wird daher ›Stage view‹ (engl. Bühnenansicht) genannt. Hier arbeiten wir während den Proben und haben die Möglichkeit jederzeit anzuhalten, Abschnitte zu überspringen oder unsere Akteure (die *Assets*) umzuplatzieren und neu anzuweisen.
Das Äquivalent zur „Druckansicht“ in EFI wird die ›Play view‹ (engl. Spielansicht) genannt. Hier wird das Schauspiel aufgeführt. Wir sehen hier unser Produkt so wie es der Zuschauer später sieht. Vergleichbar mit einer Generalprobe. Entgegen den Einschränkungen eines Theaterregisseurs können wir in EFI beliebig oft zwischen diesen beiden Ansichten wechseln und unser Produkt immer weiter anpassen und perfektionieren.

Mit Hilfe der *Assets* lässt sich festlegen „Was“ wir anzeigen können. Doch wie bei einem Film oder Theaterschauspiel benötigen wir für die Darstellung interaktiver Inhalte zusätzlich noch Kontrolle über das „Wann“ und „Wie“. Wir benötigen also die Möglichkeit, den zeitlichen Ablauf und die Bewegung der *Assets* festzulegen.
Gesteuert wird die Darstellung von Assets in EFI durch Effekte. Diese werden auf einer ›Timeline‹ (engl. Zeitleiste) angeordnet und dadurch zu einem bestimmten Zeitpunkt ausgelöst.

Es wird zum Beispiel einen Effekt geben, der Bilder anzeigt. Die vorhandenen Effekte werden durch EFI vordefiniert, und werden durch ein eigenes Menü erstellt.

Wie ich später noch ausführen werde, reicht Bewegung und Darstellung von *Assets* allein nicht aus, um von interaktivem Inhalt zu sprechen. Wie beim Improvisationstheater wollen wir unserem Publikum die Möglichkeit geben auf die Darstellung einzuwirken. Es müssen daher mögliche Interaktionpunkte definiert werden, und alle genannten Darstellungen abhängig von diesen Eingaben gemacht werden.
Alle Effekte werden Startbedingungen erhalten. Diese Startbedingung können unter anderem Nutzereingaben sein. Zum Beispiel wird es einen Effekt geben, der auf ›Klick‹ eine Grafik anzeigt.

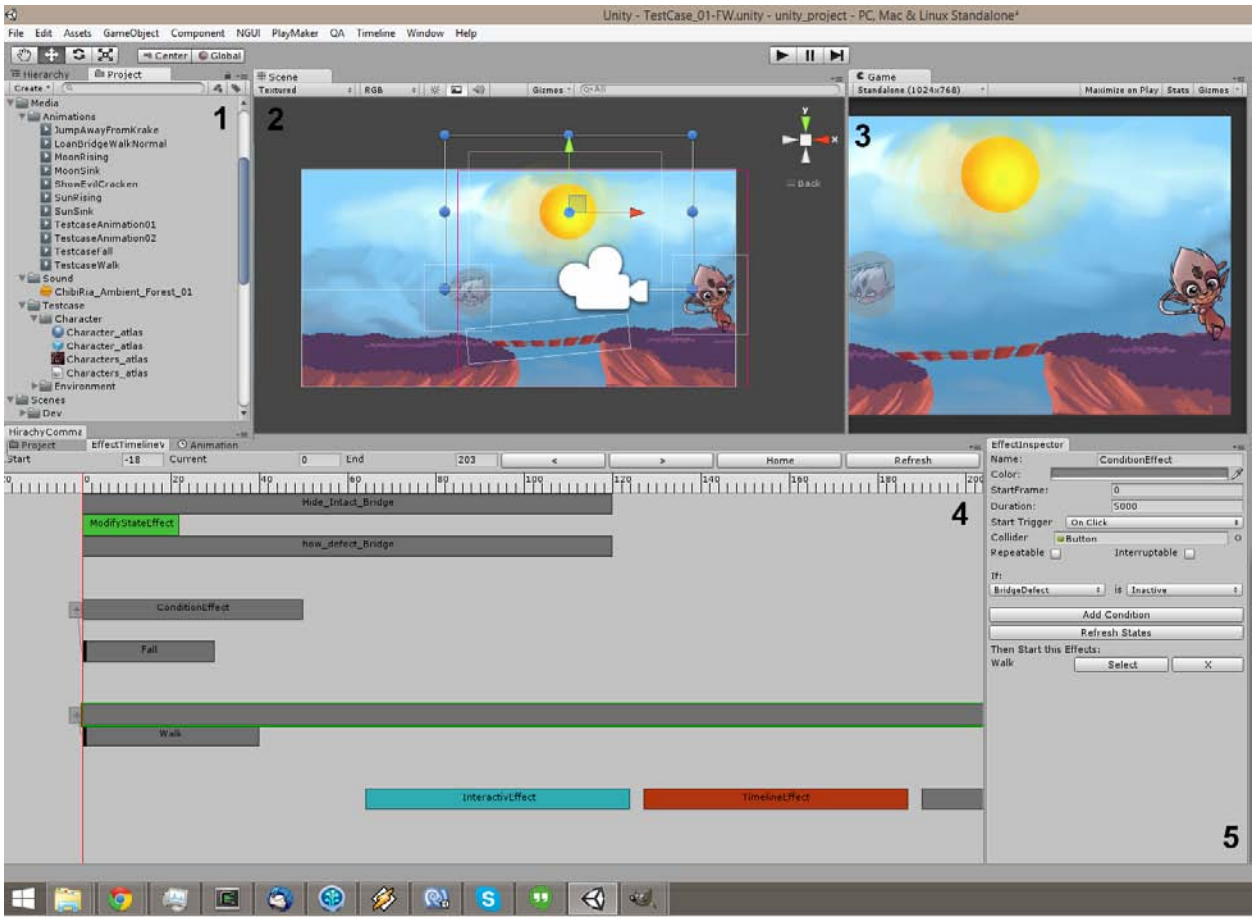


Abbildung 02: Screenshot EFI

Auf dem Screenshot ist der Prototyp von EFI zu sehen.

- 1: Assetbibliothek
- 2: Stage view
- 3: Play view
- 4: Timeline mit Effekten
- 5: Inspector (zum Einstellen der Effekteigenschaften)

„Was sind interaktive Inhalte?“ – Eine Definition

Definition: Interaktivität

»Interactivity is a widely used term with an intuitive appeal, but it is an underdefined concept[...]«[10] Laut Jensen gehört Interaktivität zusammen mit Begriffen wie „Multimedia“, „Medienkonvergenz“ und „Digitalisierung“, zu den wichtigsten Schlagwörtern im Bereich digitaler Inhalte. Seiner Meinung nach ist das Konzept überladen mit positiven Konnotationen im Zusammenhang mit technologischem Fortschritt und der individuellen Entscheidungsmöglichkeit.[11] Wie bei vielen dieser Schlagwörter ist die genaue Definition nicht offensichtlich. Für die Bewertung der Anwendungsfälle bedarf es somit der Festlegung einer für diese Arbeit gültige Definition.

Die Bedeutung des Wortes Interaktivität stammt aus dem lateinischen: ›inter agere‹ : deutsch „wechselseitig“ und „aufeinander bezogen“. Es weist auf eine Wechselbeziehung zwischen zwei oder mehreren beliebigen Größen hin, bei der implizit Informationen ausgetauscht werden.[12] Von dieser Wortbedeutung ausgehend unterscheiden sich die Definitionen je nach betrachtetem Fachgebiet. Im naheliegenden Umfeld der Informatik versteht man unter Interaktivität vorrangig die kontinuierliche Einflussnahme eines Menschen auf den Programmablauf einer Maschine. In diesem Sinne versteht man bereits den Dialog zum Fortführen oder Abbrechen einer Programmausführung als interaktives Bedienelement.[13] Diese Definition umfasst eine Vielzahl von Anwendung, die wir nach unserem intuitiven Verständnis, wie es Rafaeli beschreibt, nicht als interaktiv empfinden. Hieraus resultiert die Notwendigkeit die Definition auf andere Fachbereiche auszuweiten.

Im Bereich der Sozialwissenschaften wird Interaktion als das wechselseitig aufeinander bezogene Handeln von Akteuren definiert, also als das Geschehen zwischen Personen, die aufeinander reagieren, einander beeinflussen und steuern.[14] Die sozialwissenschaftliche Definition widerspricht somit der genannten Definition der Informatik, da sie explizit die Aktion zwischen Personen beschreibt. Bei den Resultaten die EFI erzeugt, handelt es sich hingegen um eine „Mensch-Maschine“-Interaktion. Der interessante Aspekt dieser Definition, die gegenseitige Beeinflussung der Teilnehmer, lässt sich trotzdem übernehmen. Daraus folgt, dass eine Zwei-Wege-Kommunikation stattfinden muss, um von interaktiver Software sprechen zu können.
Alle Eingaben des Nutzers an die Software können als „Hinweg“ betrachtet werden. Der „Rückweg“ bezeichnet dann folglich die Anzeige des gewünschten Inhalts durch das System. In diesem Fall wurde jedoch lediglich die „Handlungsweise“ der Software beeinflusst. Um von

[10] Vergl. Jensen, Interactivity, S.185 - Nach Sheizaf Rafaeli (1988)
[11] Vergl. Jensen, Interactivity, S.185
[12] Vergl. Wikipedia: abgerufen am 28,10,2013 , http://de.wikipedia.org/wiki/Interaktivit%C3%A4t
[13] Vergl. Jensen, Interactivity, S.190 und Wikipedia: abgerufen am 28,10,2013 , http://de.wikipedia.org/wiki/Interaktivit%C3%A4t
[14] Vergl. Wikipedia, abgerufen am 04.11.2013 http://de.wikipedia.org/wiki/Soziale_Interaktion

gegenseitiger Beeinflussung zu sprechen, müsste der nun dargestellte Inhalt umgekehrt wieder das Verhalten des Nutzers beeinflussen.

Mit Hilfe dieser Aussage lässt sich Interaktion über zwei Kriterien definieren:

1. Steuerung des Inhaltsverlaufs durch den Nutzer
2. Beeinflussung des Nutzerverhaltens durch die dargestellten Inhalte

Der zweite Punkt der Definition ermöglicht eine Einordnung in die Kategorie „interaktive Software“ lediglich auf Basis subjektiver Einschätzung. Es ist keine eindeutige Einordnung anhand von Softwarefunktionen.
Weiterhin kann frei nach der Aussage von Paul Watzlawick: »One cannot not communicate.«[15], argumentiert werden, dass auch eine bewusste „Nicht-Beeinflussung“ des Nutzers, die Reaktion des Nutzers beeinflusst. Somit verändert jeder dargestellte Inhalt das Verhalten des Nutzers. Daraus ergibt sich, dass der zweite Punkt der Definition automatisch erfüllt ist, sobald sich der dargestellte Inhalt auf Basis von Nutzereingaben verändert. Diese Auslegung der Definition führt dazu, dass viele Inhalte als interaktiv eingestuft werden, die unserem intuitiven Verständnis nicht entsprechen. Aus diesem Grund wird im Folgenden eine Möglichkeit vorgestellt, welche die Bestimmung des Interaktivitätsgrades ermöglicht. Dadurch sind Abgrenzungsfragen nicht mehr absolut, sondern graduell zu beantworten.

Interaktivitätsgrad

Mit der genannten Definition kann entschieden werden, ob es sich bei einem Anwendungsfall um interaktive Software handelt. Die Definition erlaubt hingegen keine qualitative Aussage über den Grad der Interaktivität. Jensen zeigt in seiner Ausführung unter anderem die Möglichkeit auf, Interaktivität nicht nur über Kriterien zu definieren, sondern über das Erreichen verschiedener Stufen, bezeichnet als »dimensional Concepts of Interactivity«.[16] Die einfachste Möglichkeit für eine solche Abstufung ergibt sich als Ableitung der bereits genannten Definition als eindimensionale Skala, mit folgenden Stufen (aufsteigend von wenig interaktiv zu hochgradig interaktiv):

Stufe 1: Lineares Abspielen des Inhaltes
Stufe 2: Wie Stufe 1 + definierte Stopps, zu denen der Nutzer springen kann
Stufe 3: Wie Stufe 2 + die Möglichkeit für den Nutzer den Inhalt zu verzweigen
Stufe 4: Wie Stufe 3 + Inhaltswiedergabe beeinflusst das Nutzerverhalten, durch dargestellten Inhalt

Stufe 1 umfasst alle klassischen Medien, die wir heute nicht mehr als interaktiv verstehen. Hierunter fällt unter anderem das Abspielen eines Spielfilms im Fernsehen oder Kino. Diese Stufe soll als „Nicht-interaktiv“ gelten.

---

[15] Vergl. wikiquote : http://de.wikiquote.org/wiki/Paul_Watzlawick abgerufen am 10.01.2014 zitiert nach , - *Paul Watzlawick, Janet H. Beavin, Don D. Jackson. Pragmatics of Human Communication. New York W. W. Norton 1967. p. 51*)
[16] Vergl. Jensen, Interactivity, S. 192

Ab Stufe 2 kann man von interaktiven Inhalten sprechen. Als Beispiel dient hier eine DVD mit Menü, in welchem einzelne Szenen direkt angewählt oder übersprungen werden können. Auch wenn wahrscheinlich ein großer Teil der Nutzer dieses Menü noch nicht als interaktiv bezeichnen würde, ist eine Einflussnahme des Nutzers auf die Inhaltswiedergabe vorhanden. Somit besteht ein geringer Grad an Interaktivität.
Stufe 3 fügt die Verzweigung des Inhaltes hinzu. Dadurch eröffnet sich ein sehr breiter Bereich. Man stelle sich hier einen Spielfilm auf DVD vor, der zwei verschiedene Enden anbietet. Diese Art der Interaktion bezeichnet Rafaeli als »reactive communication«.[17] Das bedeutet, dass sich spätere Nachrichten (in unserem Fall Entscheidungspunkte) immer auf eine vorangegangene Nachricht beziehen müssen. Die Erweiterung dieser Form zur sogenannten »full interactivity«[18] verlangt, dass spätere Nachrichten sich immer auf eine Sequenz von vorangegangen Nachrichten beziehen. Ein Beispiel hierfür sind weitere Entscheidungspunkte zu einem früheren Zeitpunkt der Geschichte, welche die Auswahl vom Ende in Version zwei erst ermöglichen, oder verhindern. Beide Formen werden in der 3. Stufe zusammengefasst, da technisch zwischen Beiden kein relevanter Unterschied besteht.
Die 4. Stufe auf der Skala erfordert, die Rückwirkung des dargestellten Inhalts auf die Entscheidungen des Nutzers. Hier stellt sich das Problem der Abgrenzung zu Stufe 3.
Der Unterschied zwischen den ersten drei Stufen basiert auf den gebotenen Möglichkeiten der verwendeten Software zum Erstellen der Inhalte. Der Schritt von Stufe 3 auf Stufe 4 erfolgt hingegen ausschließlich auf Basis der Präsentation des Inhaltes. Editoren die den Bezug auf Nachrichtensequenzen unterstützen haben demnach das Potential die Kriterien von Stufe 4 zu erfüllen.
Als veranschaulichendes Beispiel für die Erfüllung von Stufe 4 dient eine Anekdote aus dem OpenWorld Rollenspiel *„The Elder Scrolls III: Morrowind“* von *„Bethesda Game Studio“*[19]:
In diesem Spiel boten die Städte das Bild einer Känguru farm. Alle ›Spieler-Avatare‹ bewegten sich hüpfend auf den Straßen und Plätzen. Der Grund dafür war, dass sich die körperliche Stärke des ›Avatars‹ unter anderem abhängig davon entwickelte, wie oft man sprang. Es ist leicht nachzuvollziehen, dass sich nahezu alle ›Avatare‹ hüpfend durch die Welt bewegten, um schneller den nächsten Stärkepunkt zu erlangen. Die Darstellung des relevanten Spielwertes „Stärke“ in Abhängigkeit des Nutzerverhaltens, beeinflusste die Nutzer dazu, eine ansonsten auf ebenen Straßen unsinnige Aktion wiederholt auszuführen.

Das nächste Beispiel befasst sich mit der Schwierigkeit der Abgrenzung:
Der Protagonist einer Geschichte befindet sich in einem Labyrinth. Er möchte den Ausgang finden. An einigen Kreuzungen kann der Nutzer entscheiden welchen Weg der Protagonist einschlägt und damit die Handlung beeinflussen.
Der Erste der möglichen Wege wird dunkel, verrottet und eingefallen präsentiert, der Zweite ist sauber und hell erleuchtet. Die Frage, ob man in einem solchen Fall schon von ausreichender Beeinflussung des Nutzers sprechen kann, um Stufe 4 zu erreichen, muss hier mit „nein“ beantwortet werden.

---

[17] Vergl. Jensen, Interactivity, S 194, Nach Rafaeli
[18] Vergl. Jensen, Interactivity, S194, Nach Rafaeli
[19] Siehe Wikipedia: http://en.wikipedia.org/wiki/The_Elder_Scrolls_III:_Morrowind

Im ersten Beispiel ist der Interaktionskreis geschlossen. Die Aktion des Nutzers verursacht eine veränderte Darstellung des Systems („Hinweg“). Die Veränderung verursacht nachfolgend eine Reaktion des Nutzers („Rückweg“). Im zweiten Beispiel erfolgt nur eine Aktion des Nutzers durch die Wahl des Weges.
Um Stufe 4 zu erfüllen muss demnach die geänderte Handlung des Nutzers durch direktes Feedback auf eine vorangegangen Handlungen geschehen.
Das genannte Beispiel im Labyrinth fällt solange unter Stufe 3, bis die Handlung des Nutzers, in diesem Fall die Entscheidung für den „dunklen Gang“, eine negative Auswirkung für den Nutzer darstellt, und er zukünftig den „hellen Gang“ wählt.

Daraus ergibt sich folgende geänderte vierte Stufe der Skala :
Stufe 4: Wie Stufe 3 + Inhaltswiedergabe beeinflusst das Nutzerverhalten, durch direktes Feedback auf seine gewählten Aktionen.

Die beschriebene 4-stufige Skala bietet eine grobe Vergleichsmöglichkeit von Inhalten bezüglich ihrer Interaktivität. Da wir uns im Bereich interaktiver Software bewegen, werden viele der betrachteten Beispiele sich in Stufe 3 oder 4 befinden. Dies macht einen Vergleich innerhalb einer Stufe nötig.
Eine Möglichkeit zum Vergleich bietet, laut Jensen, die von Brenda Laurel vorgestellte Messung in drei Dimensionen: Frequenz, Bandbreite und Signifikanz.[20]
Die „Frequenz“ gibt an wie häufig der Nutzer interagieren kann. Die „Bandbreite“ gibt an wieviele verschiedene Möglichkeiten zur Auswahl stehen, und die „Signifikanz“ gibt an wie stark der Einfluss der Entscheidung auf den weiteren Inhalt ist.
Diese dreidimensionale Messung lässt sich durch Addition der einzelnen Werte wieder als eindimensionales Maß einsetzen und kann zum Vergleich der Inhalte einer Stufe untereinander genutzt werden. Bei Frequenz und Bandbreite handelt es sich um quantitative Maße, die durch den Editor nicht eingeschränkt sind, sondern vom Nutzer umgesetzt werden. Deshalb dient das dreidimensionale Maß nur als Hilfsmaß zum Vergleich innerhalb einer Stufe.

Der Interaktivitätsgrad lässt sich wie folgt ausdrücken:
1. Hauptindex: Einordnung in die Stufen 1 bis 4
2. Nebenindex: Die Messung der addierten Werte auf der dreidimensionalen Skala.

[20] Vergl. Jensen, Interactivity, S. 196 Nach: Laurel, Brenda, (1991) Computers as Theatre, Reading, Mass.: Addison-Wesley.

„Welchen Inhalt erstellt man mit dem Editor?“ – Der Anwendungsfall

Interaktive Inhalte werden heute schon vielfältig erstellt. Die folgenden drei Beispiele illustrieren Ergebnisse, die zukünftig mit EFI erstellt werden könnten.

„Kleiner Fuchs“ – iPad Applikation von: Fox & Sheep[21], erhältlich im Apple Appstore

Bei *„Kleiner Fuchs“* handelt es sich um eine iPad Applikation für Kinder zwischen 2 und 6 Jahren. In der Anwendung werden verschiedene Kinderlieder vorgespielt. Dazu wird eine bildschirmfüllende Illustration angezeigt. Die Elemente in der Illustration lassen sich durch ›Tap‹ oder ›Swipe‹ aktivieren und führen kurze Animationen aus. (⤡02_LittleFox.mp4)

Abbildung 03: „Der Mond ist aufgegangen“ - Screenshot aus „Kleiner Fuchs“

„Ria-Chibi-Camp“ – iPad Applikation von ➸Stenarts Berlin, in Entwicklung

Bei *„Ria-Chibi-Camp*“ handelt es sich um ein interaktives Comic für Kinder zwischen 7 und 14 Jahren. Die Bilder werden wie bei einem klassischen Comic in Panels dargestellt. Der Nutzer kann durch ›Tap‹ Elemente in den Bildern aktivieren oder einsammeln. Dadurch kann der Verlauf der Geschichte beeinflusst werden. (⤡03_Ria_Chibi_Camp.mp4)

[21] iTunes Link: https://itunes.apple.com/de/app/kleiner-fuchs-kinderlieder/id499541243?mt=8

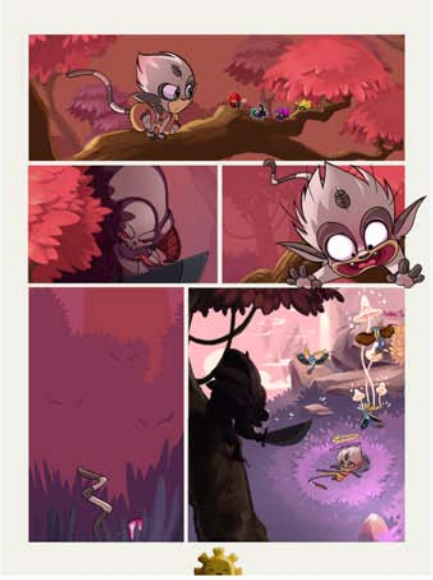


Abbildung 04: „Ria-Chibi-Camp“ – Seite 1

„*Haunted Island*“ - iPad Applikation von ➸Happy Tuesday, in Entwicklung

Bei *„Haunted Island“* handelt es sich um ein Strategiespiel für Tablets. Das Spiel beginnt mit einer Einführung in die Geschichte, verknüpft mit der Einführung des Spielers in die Bedienmechaniken (›Tutorial‹). Der Ablauf dieses ›Tutorials‹ erfordert eine zeitlich abgestimmte Interaktion mit dem Spieler, und wird im Erstellungsprozess in vielen Iterationen immer wieder umgestellt und verbessert. (⤡04_Haunted_Island.mp4)

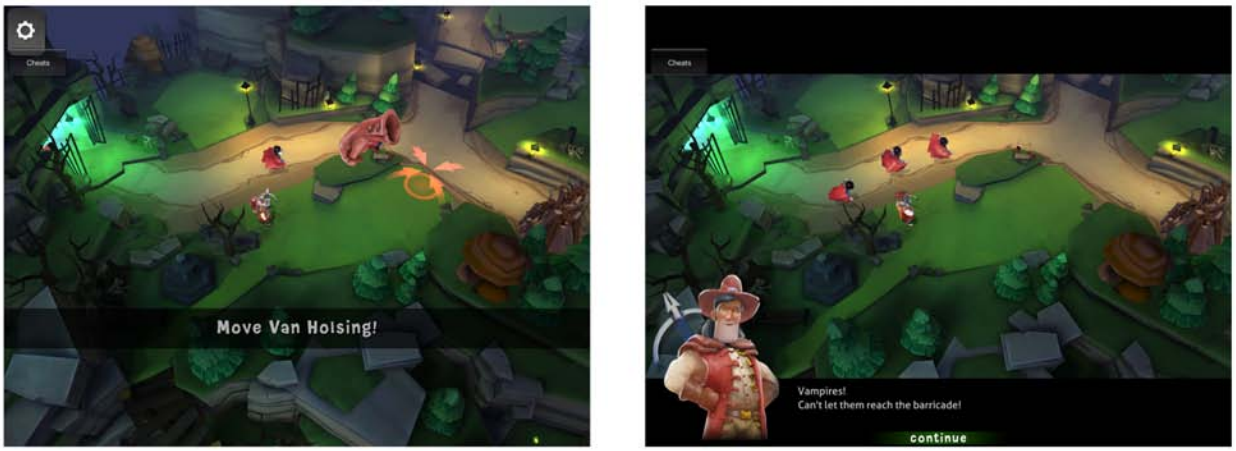


Abbildung 05: „Haunted Island“ – Tutorial 01

„Wer braucht den Editor?“ – Die Zielgruppe

EFIs Zielgruppe sind professionelle Kreative. Diese sind es gewohnt digital zu arbeiten und zu publizieren, Kreative, die ihre Inhalte interaktiv präsentieren möchten, aber nicht über Programmierkenntnisse verfügen. Berufsgruppen sind zum Beispiel: Animatoren, Illustratoren oder Game Designer.

Aktuelle Grafikprogramme wie *„Photoshop“*, *„Aftereffects“* oder *„3ds-Max“*, die zur Erstellung von 2D- oder 3D- Grafiken und Animationen benutzt werden, sind in ihrer Bedienung meist hoch komplex. Bei der Zielgruppe lässt sich somit eine gewisse Lernbereitschaft für komplexe Software voraussetzen.

Die folgenden Firmen und Mitarbeiter haben mir während der Erstellung dieser Arbeit als Interviewpartner, Teilnehmer an Workshops und Nutzertests, sowie mit Feedback zum Prototypen zur Seite gestanden.

Die Firmen

Happy Tuesday UG, Berlin

Happy Tuesday[22] ist ein junges ›Indie Studio‹ in Berlin Kreuzberg. Happy Tuesday kreiert spaßige ›mobile games‹ mit Suchtfaktor und »Cutting Edge Technologie«. Aktuell erstellt Happy Tuesday das Spiel Haunted Island.

Stenarts Ltd. – Stenarts Berlin

Seit vielen Jahren arbeitet Stenarts[23] an der Entwicklung und Umsetzung eigener Stoffe, wie zum Beispiel „RIA – Die Lichtklan-Chroniken“. Dabei verfolgt Stenarts das Ziel der multimedialen und plattformübergreifenden Auswertung, welche durch die Zusammenarbeit mit angebundenen Produzenten und Kooperationspartnern ständig weiterentwickelt wird.

Die Tester

Samuel Stephan - Stenarts Berlin

Samuel ist ausgebildeter Kommunikationsdesigner und erstellt freiberuflich Illustrationen und Layout/Satzarbeiten. Bei Stenarts arbeite er an der Planung und Umsetzung der interaktiven Comics für Ria-Digital. Durch sein Studium und eigenes Interesse besitzt er ein grundlegendendes Verständnis für Softwareerstellungsprozesse, hat aber selbst keine Programmierfähigkeiten. Zu der Software aus Samuels täglichem Arbeitsumfeld gehören Grafik- und Animationsprogramme von *„Adobe“* wie:
„Photoshop“, „InDesign“, „Aftereffects“

Kevin Blank - Happy Tuesday Berlin

Kevin ist an der Games Academy Berlin ausgebileter Game Designer. Bei Happy Tuesday ist er unter anderem für die Konzeption der Tutorials für das ›Tower Defense‹[24] Spiel *„Haunted Island“*

---

[22] http://happy-tuesday.de/

[23] http://www.stenarts.com/index.php?home

[24] Spielgenre bei dem ein Ziel mit Hilfe von Türmen vor anstürmenden Gegner beschützt werden muss. Vergl. http://de.wikipedia.org/wiki/Tower_Defense | abgerufen am 26.01.2014

zuständig. Zu der Software aus Kevins täglichem Arbeitsumfeld gehören Game-Editoren, Flowchart- und Tabellekalkulationsprogramme wie:
„Unity3d“, „UDK“, „Gamemaker“, „Powerpoint“, „Excel“ und *„yEd“*.

Tino Englert - Happy Tuesday Berlin

Tino ist Game Design Praktikant bei Happy Tuesday, er hat Game Design und -Art an der Games Academy Berlin mit Zertifikat abgeschlossen. Seit 2010 arbeitet er im Bereich Game Design. Seine Aufgaben umfassen unter anderem: Entwerfen und Erstellen von ›Tutorials‹, ›Balancing‹[25] und das ›Finetuning‹ von Spielsystemen. Zu der Software aus Tinos täglichem Arbeitsumfeld gehören Game-Editoren, Flowchart- und Tabellekalkulationsprogramme wie:
„MS Excel“, „MS Word“, „Unity3d“, „Powerpoint“, „Visio“

Pauline Altmann - Freie Grafikdesignerin

Pauline hat an der FH-Potsdam Kommunikations-Design studiert und 2012 mit B.A abgeschlossen. Seit 6 Jahren arbeitet sie als freie Grafikdesignerin im Bereich Grafik- und Motion-Design mit einem Fokus auf feine Typografie und Buchgestaltung. Zu Ihren Kunden zählen unter anderem Verlage, Kulturinstitutionen und Filmschaffende. Zu der Software aus Paulines täglichem Arbeitsumfeld gehören Grafik- und Animationsprogramme von *„Adobe“* wie:
„Photoshop“, „InDesign“, „Aftereffects“

Begründung der Auswahl

Kevin, Tino und Samuel arbeiten in Arbeitsbereichen, in denen sie täglich mit der Erstellung interaktiver Inhalte für Software betraut sind.

Bei dem Tutorial, das bei Happy Tuesday von Kevin und Tino erstellt wird, handelt es sich um ein ständiges Wechselspiel aus kurzen Erklärungen und Scriptsequenzen. In Letzteren werden die geforderten Interaktion und Storyelemente abgespielt. Anschließend folgen Sequenzen, in denen der Nutzer schrittweise die Handlungen nachahmt. Das System muss dabei stetig auf die Eingaben des Nutzers reagieren und, je nach Situation, Hinweise wiederholen, überspringen oder bestätigen.

Die interaktiven Comics die Samuel bei Stenarts erstellt, bestehen aus festgelegten Animationen und Szenen die in üblichen Comic Panels dargestellt werden.
Der Leser kann durch ›Tap‹ und ›Swipen‹ die Art und Geschwindigkeit der Animation beeinflussen. Durch Einsammeln von Gegenständen oder der Wahl von verschiedenen Wegen, kann der Nutzer den Verlauf der Geschichte lenken.

Im Rahmen der Zieldefinition für diese Arbeit wurde angenommen, dass die Konzeption und Umsetzung interaktiver Inhalte ein hohes Maß an Kommunikation zwischen verschiedenen Abteilungen vorraussetzt, und der kreative Arbeitsprozess dadurch erschwert wird.

[25] Festlegen der Spielrelevanten Parameter

Die genannten Tester arbeiten in größeren Teams mit weiteren Grafikern und Programmierern. Es gehört zu ihrem täglichem Arbeitsablauf die erarbeiteten Konzepte an andere Abteilung zu kommunizieren. Im Rahmen der aktuellen Aufgabenstellung der Teams können Verbesserungen durch die Software direkt angewandt und anschließend beurteilt werden.
Im Hinblick auf die definierten Ziele des Editors bildet die Testgruppe die Ausgangslage, sowie die Bedürfnisse der Zielgruppe repräsentativ ab.

Pauline hat aufgrund ihrer Arbeit im Bereich ›Motion-Design‹ Erfahrung mit animiert präsentierten Inhalten, repräsentiert mit ihrem Schwerpunkt jedoch eine Berufsgruppe die für die „klassischen“ Medien produziert. Wie in der Einleitung beschrieben, liegt die Produktion von Inhalten bei den „klassischen“ Medienproduzenten. Deswegen wird es notwendig die Zielgruppe zukünftig auf diese Berufsgruppen auszuweiten.

„Gibt es das nicht schon?“ - Die Softwaretests

Software für Vergleichstests

Der Markt für Software die der Vision von EFI entspricht ist recht jung, aber nicht neu. Das Erstellen interaktiver Inhalte ohne Programmierkenntnisse wird von einigen existierenden Softwarepakete versprochen. Für den Nutzertests werden Programme benutzt, die dieses Versprechen geben.

Eine Auswahl möglicher Softwarepakete:

- Adobe Edge Animate[26] | Engine für Interaktive Websiten
- Baker Framework[27] | Engine für interaktive E-Books
- Playmaker [28] | Prototypen Framework für *„Unity3d“*
- Scirra [29] | 2D Game Engine
- Purple Motion[30] | Engine zum erstellen von interaktiven Apps

Das *„Baker Framework“* bietet die Möglichkeit HTML-Seiten zum „*e-Pub*“ Format zu exportieren. Die Erstellung von HTML Seiten erfordert allerdings Programmierung. Damit scheidet das *„Baker Framework“* für den Nutzertest aus.
„Scirra“ ist eine 2d-Gameengine mit einer Vielzahl an Exportmöglichkeiten. *„Scirra“* besitzt einen umfangreichen Editor mit vielen vordefinierten Aktionen. Diese können ohne weitere Programmierung direkt eingesetzt werden. Zugrunde liegt ein zustandsbasiertes System. Aufbau und die Arbeitsweise sind dem Programm *„Playmaker“* ähnlich. *„Playmaker“* ist in den Spiele Editor ➸Unity3d integriert. Dadurch bietet es eine bekannte Umgebung für zwei der

[26] http://html.adobe.com/edge/animate/
[27] http://bakerframework.com/books
[28] http://www.hutonggames.com/
[29] https://www.scirra.com/
[30] http://sprylab.com/de/digital-publishing

Tester. Da die Programme einem ähnlichen Konzept folgen, wird für den Softwaretest „*Playmaker*“ benutzt.

»Purple Motion stellt die Purple Publishing Suite, eine Lösung für digitales Publizieren von qualitativ hochwertigen Apps mit außergewöhnlichen Animationen und Interaktionen.« [31] „*Purple Motion*“ bietet einen zeitleistenbasierten Editor an. In dem 2d-Elemente und Sound auf der ➸›Stage view‹ bearbeitet und über die Zeit verändert werden können. Das Konzept ist dem dem von „*Adobe Edge Animate*“ ähnlich. Produkte der „*Adobe*“ Familie sind im Allgemeinen in ihrer Bedienung stark vergleichbar. „*Edge Animate*“ bietet ein vertrautes Umfeld für die Tester und wurde deswegen für den Softwaretest ausgewählt.

Im Folgenden werden die beiden ausgewählten Softwarepakete genauer beschrieben.

Adobe Edge Animate CC

Abbildung 06: Screenshot „Adobe Edge Animate“ mit ›Stage view‹ oben und ›Timeline‹ unten

„*Adobe Edge Animate*“ ist ein Programm zum »Erstellen interaktiver Inhalte für das moderne Web«[32] Laut dem offiziellen Einführungstutorial von Thomas Kretschmar auf der Website von „*Adobe Edge*“ ist es ein: »Werkzeug zum Erstellen von Animationen sowie

[31] Vergl. http://sprylab.com/de/digital-publishing Abgerufen am 15.12.2013

[32] Vergl. Subtitle von Adobe Edge Proukt Seite: http://html.adobe.com/edge/animate/ abgerufen am 05.01.2014

interaktiver Werke mithilfe von Webstandards wie HTML, CSS und Javascript, allerdings auf eine sehr kreative Art und Weise, ohne selbst Code schreiben zu müssen.« [33]

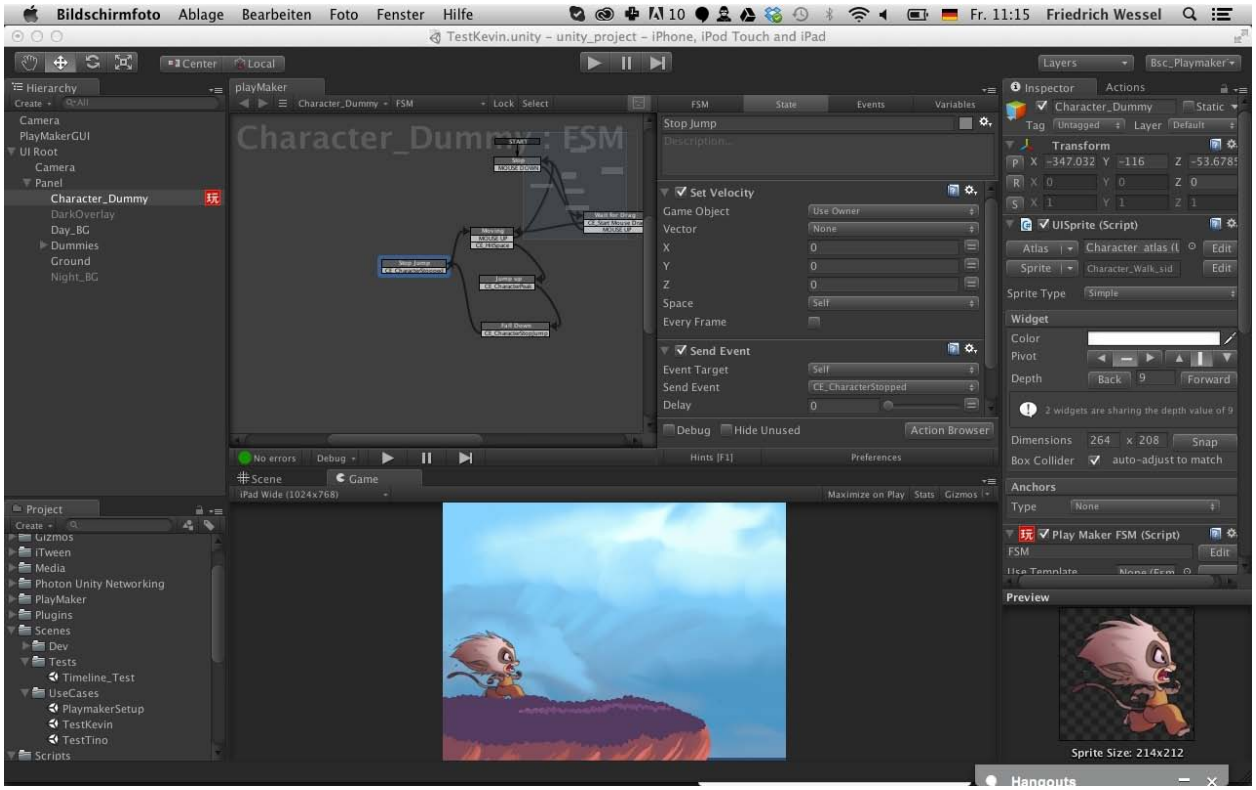
Der Aufbau von „*Edge Animate*“ ist vergleichbar zu anderen „*Adobe“* Produkten. Zum Beispiel dem Filmanimations- und Compositing[34]- Programm „*Aftereffects“*. „*Edge Animate“* besteht grundlegend aus einer ›Stage view‹ (Bühnenansicht) und einer ›Timeline‹ (Zeitleiste). Auf dieser werden einzelne Elemente angeordnet und animiert. Zusätzlich existiert ein Eigenschaftenfenster, in dem bestimmte Werte für die Elemente angepasst werden. Das definierte Ausgabeformat ist eine Website mit HTML, CSS und Javascript. Die Benennung der Eigenschaften richtet sich ausschließlich nach den entsprechenden CSS-Style Eigenschaften. Die Werte finden deswegen häufig keine Entsprechung in anderen Grafikprogrammen wie „*Photoshop“* oder auch „*Aftereffects“*.
Der Workflow besteht darin Elemente auf der ›Stage view‹ zu positionieren. Anschließend werden Transformations- oder Farbwerte mithilfe der ›Timeline‹ animiert. Im Gegensatz zu „*Aftereffects“* bietet „*Edge Animate“* die Möglichkeit diese Animationen sowohl in Abhängigkeit der Zeit, als auch durch feste Bedingungen zu starten. Diese Bedingung können über kleine Scripte definiert werden. Um den Ansatz „ohne Programmierkenntnisse“ zu verwirklichen, bietet der Scripteditor eine große Anzahl vordefinierter ›Code-Snippets‹. Die ›Snippets‹ werden aus einer Bibliothek in das Script eingefügt. Mit vorhandenen Programmierkenntnissen kann man auf diese Art und Weise sehr schnell die Syntax und API lernen. Ob dieser Ansatz auch ohne Programmierkenntnisse ausreichende Möglichkeiten bietet, wird im Nutzertest überprüft.

---

[33] *Trainings Video Adobe Edge Animiate, Thomas Kretschmar: https://helpx.adobe.com/de/creative-cloud/learn/start/animate.html Abgerufen am 11.12.2013*

[34] **Compositing** ist ein Begriff aus der Filmtechnik und findet in der Postproduktion eines Filmes als visueller Effekt Anwendung. Im Compositing werden zwei oder mehr voneinander getrennt aufgenommene oder erstellte Elemente zu einem Bild zusammengeführt. - Wikipedia: http://de.wikipedia.org/wiki/Compositing abgerufen am 05.01.2014

Hutongames Playmaker



Screenshot „Playmaker“ mit ›Flowgraph-Editor‹ oben und ›Stage view‹ unten - Abbildung 07

„Playmaker“ ist eine Editorerweiterung für ➸Unity3d. Es handelt sich um ein nodebasiertes System um digitale Spiele, KI- und Effektverhalten zu definieren. Die Basis ist der Aufbau von endlichen Automaten deren Übergänge über vordefinierte Signale wie zum Beispiel ›OnClick‹ ausgelöst werden. Die einzelnen Zustände der Automaten können mit vordefinierten Aktionen belegt werden. Während der Zustand aktiv ist, werden diese Aktionen nacheinander ausgeführt. Die Palette der mitgelieferten Aktionen reicht von mathematischen Operationen, über Transformations- und Farbveränderungen von Objekten, bis dahin Signale zum Statuswechsel auszulösen.
Die Umsetzung von interaktiven Inhalten der Stufe 3 oder 4 wird nur durch die mitgelieferten Aktionen bestimmt, da endliche Automaten sich in ihren Möglichkeiten nicht von Turingmaschinen unterscheiden.

Bei *„Edge Animate“* ist die Fragestellung im Nutzertest, ob das System alle Bedingungen erfüllen kann. Im Gegensatz dazu ist bei *„Playmaker“* die Fragestellung ob die Modellierung des Inhaltes über Automaten für Kreative intuitiv und übersichtlich genug ist.

Softwaretests

Die Softwaretests verfolgen zwei Ziele. Erstens wird evaluiert welchen ➸Interaktivitätsgrad die erstellten Inhalte erreichen können. Zweitens werden Informationen über die vom Nutzer gewünschten Anwendungsgebiete gesammelt.
Die Tester kennen die Software vorher nicht. Sie sind aber mit dem jeweiligen Grundkonzept vertraut. Die Nutzer erhalten eine 10 minütige Einführung. In dieser werden Bedienung und sichtbare Fenster erklärt. Danach erhalten sie ein vorbereitetes Grafikset. Das Set enthält verschiedenen Charaktere, Gegenstände und Hintergrundelemente (➸Bilder). Die Aufgabe lautet mit diesem Material eine kleine interaktive Applikation zu erstellen. Der Zeitrahmen beträgt 60 Minuten. Die Tester werden gebeten nach dem ›Thinking-Aloud‹[35] Prinzip zu arbeiten.
Die Grafiken sind für die Software entsprechend aufbereitet und bereitgestellt. Es ist noch kein Material importiert.

1. Nutzertest ➸Samuel Stephan, ➸*„Adobe Edge Animate“*

Abbildung 08: Screenshot Nutzertest mit Samuel - „Adobe Edge Animate“

Samuel möchte in seiner Applikation dem Nutzer die Möglichkeiten geben die Brücke mithilfe eines ›Klicks‹ zu zerstören. Bei einem ›Klick‹ auf die andere Seite der Brücke soll sich der Charakter anschließend über den Fluss bewegen. Wenn die Brücke kaputt ist, soll der Charakter

[35] Siehe: http://de.wikipedia.org/wiki/Thinking_Aloud_Test | Abgerufen am 21.01.2014

ins Wasser fallen, ansonsten auf der anderen Seite ankommen. Einige Hintergrundelemente, wie zum Beispiel das Wasser, möchte Samuel in einer Endlosschleife animiert haben.

Die Endlosanimation ist für Samuel schwierig zu erstellen. Die Funktion ist nur über Scripte zu realisieren. Durch ausprobieren aller möglichen ›Abspiel-Code-Snippets‹ gelingt es die Animation wie gewünscht anzulegen. Die interne Vorschau von „*Edge Animate*“ funktioniert nur für Animation. Durch Scripte gesteuerte Elemente werden nur in der exportierten HTML-Seite angezeigt. Dadurch wird das Ausprobieren der ›Snippets‹ erschwert.
Mit Hilfe eines weiteren ›Snippets‹ wird die Grafik der Brücke ›OnClick‹ ausgetauscht. Die Grafik wird allerdings bei jedem ›Klick‹ ausgetauscht. Ein Verhalten das nicht gewünscht ist. Das Problem wird auch mit weiterem Ausprobieren nicht gelöst. Die Brücke kann also mehrfach zerstört werden.
Die Bewegung des Charakters über die Brücke wird durch unterschiedliche Grafiken dargestellt. Diese werden gemeinsam animiert und blenden ineinander über. Dafür ist eine Gruppierungsfunktion gewünscht. Wie schon erwähnt, sind alle Objekte nach ihrem entsprechenden HTML Gegenstücken benannt. Gruppen heißen demnach „DIV“. Diese Bezeichnung ist Samuel nicht geläufig. Erst das Probieren aller Optionen im Kontextmenü bringt die Lösung.
Alle verwendeten Grafiken müssen immer wieder aus dem Explorer importiert werden. In der zunehmend komplexeren Datei vermisst Samuel eine Bibliothek innerhalb des Programms.
Die Animation des Charakters soll abhängig vom Brückenzustand gestartet werden. Dazu müssen in dem Script Zustände gespeichert und abgefragt werden. Es gelingt Samuel nicht die ›Snippets‹ entsprechend anzupassen. Stattdessen ändert Samuel seine Ziele und verwendet die bisher bekannten Elemente. Er sucht sich aus dem Hintergrund die Sonne aus, und versteckt sie in dem Fass neben der Höhle. Auf Klick steigt die Sonne zum Himmel auf. Mit dem Animieren der Elemente in den verschiedenen ›Timelines‹ kommt Samuel ohne Probleme zurecht. Es fällt auf, dass „*Adobe Edge*“ hier sehr ähnlich und intuitiv wie „*Aftereffects*“ funktioniert.
In einem kurzen Gespräch nach dem Test beschreibt Samuel das Arbeiten in der ›Timeline‹ als bekannt und angenehm. Das Scriptfenster ist benutzbar, aber die vordefinierten Snippets bieten zu wenige Möglichkeiten. Das der Scriptcode direkt sichtbar ist, hilft nicht weiter.
Das Resultat des Tests befindet sich als Video und HTML-Seite auf der CD.
(↘05_Resultat_Nutzertest_Samuel/index.html) und (↘05_Resultat_Nutzertest_Samuel.mp4)

Abbildung 09: Screenshot aus dem Resultat von Samuels Nutzertest

2. Nutzertest: ➸Kevin Blank, ➸*„Playmaker“* für Unity

Kevin setzt sich als Ziel den Charakter mit Hilfe von ›Swipe‹ Bewegungen nach links und rechts durch die Szene zu bewegen. Anschließend möchte er den Charakter mit der Leertaste über den Abgrund springen lassen.

Das Aufbauen des ersten Automaten verläuft zügig. Das Nodegraph-Fenster ist effizient bedienbar und übersichtlich. Die benötigten Aktionen werden in der Bibliothek mithilfe der Suchfunktion schnell gefunden. Die einblendbaren Informationen zu den einzelnen Aktion erweisen sich bei der Bewertung der Aktionen als hilfreich.
Die Namen der Aktionen folgen üblichen Fachbegriffen aus der Informatik. Dadurch entstehen vermehrt Unklarheiten. Als stellvertretendes Beispiel steht hier: ›Event‹.
Ein Zustandsübergang wird durch ›Events‹ wie ›OnClick‹, oder ›OnMouseMove‹ ausgelöst. Diese beiden Funktionen existieren bereits vorgefertigt auf jedem Zustand. Kevin benutzt sie ohne Probleme.
Wenn eine bestimmte Bewegungsdistanz mit der Maus zurückgelegt wurde, möchte Kevin den Zustand des Charakters wechseln. Dazu wird ein eigener Zustandsübergang benötigt. Kevin sucht nach Aktionen wie: „Statechange“, „Condition“ oder „Transition“. Die Aktion „Start Custom Event“ wird daher nicht gefunden.
Danach zeigen sich Probleme beim Verständnis des endlichen Automaten. Kevin geht davon aus, dass die einzelnen Zustände des Automaten nacheinander durchlaufen werden. Im Automaten werden allerdings nur Aktionen des aktiven Zustandes evaluiert. *„Playmaker“* markiert während der Vorschau die aktiven Zustände und Aktionen grün. Dadurch erkennt Kevin den Fehler schnell und stellte den Automaten um.

„Playmaker“ bietet die Möglichkeit verschiedenen Ergebnisse der Aktionen in Variablen zu speichern. Diese Variablen können frei benannt werden. Kevin benutzt die Funktion ohne Probleme um Zustände wie „Character is Moving“ oder „Character Idle“ zu speichern.

Zum Abschluss justiert Kevin die Werte für seine Steuerung mithilfe der Debug-Funktion von *„Playmaker“* die es erlaubt den Automaten beim Ausführen von Aktionen anzuhalten, und sich die aktuellen Werte anzuschauen.
Der Aufbau der Steuerung dauert ca. 50 Minuten. Dabei fällt auf, dass relativ wenig in dem gut strukturierten Flowgraph gearbeitet wird. Es wird fast ausschließlich das listenförmige Aktionsfenster benutzt. Dieses Fenster ist zum Ende des Tests unübersichtlich.
Der Sprung des Charakters gelingt nur in vertikaler Richtung. Es gelingt Kevin nicht beide Automaten gleichzeitig auszuführen.
Im anschließenden Gespräch beschreibt Kevin das Verstehen des endlichen Automaten als schwierig. Er empfindet es aber als »mächtiges Werkzeug«. Der Test hat ihm allerdings auch gezeigt, das schon sehr einfache Aktionen aufwändige Automaten erzeugen.
Das Ergebniss befindet sich als Windows ›exe‹ und Mac ›app‹, sowie als Video auf der CD. (↘06_Resultat_Nutzertest_Kevin/) und (↘06_Resultat_Nutzertest_Kevin.mp4)

Abbildung 10: Screenshot aus dem Resultat von Kevins Nutzertest

Auswertung der Nutzertests

In der Vision wird ein „kreativ experimentelles Arbeiten“ gefordert. Die beiden Tests zeigen exemplarisch, dass erste Schritte in diese Richtung gegangen wurden. Beide Nutzer konnten Teile ihrer selbstgesteckten Ziele verwirklichen. Der kreative Arbeitsprozess wird allerdings an einigen Stellen behindert. Samuel konnte in der kurzen Zeit viele Elemente einbinden, und animieren. *„Adobe Edge“* unterstützt den künstlerischen Erstellungsprozess folglich sehr gut.

Allerding wurde der Inhalte angepasst, da die Software nicht die gewünschten Funktionen bereitstellt. Das Speichern und Laden von Zuständen ist nicht ohne Programmierung möglich. Deswegen lassen sich in „Adobe *Edge"* ohne Programmierkenntnisse nur interaktive Inhalte der Stufe 2 umsetzen. Die vorgefertigten Scripte ermöglichen ein sehr schnelles Einarbeiten von Interaktionspunkten. Dadurch ist mit geringem Aufwand eine hohe Frequenz und damit ein hoher Nebenindex zu erreichen.
Im Gegensatz dazu konnte Kevin ohne Probleme Zustände setzen und abfragen. „*Playmaker"* ermöglicht somit interaktive Inhalte der Stufe 3 bis Stufe 4. Der hohe Abstraktionsgrad der einzelnen Aktionen erfordert viel Feinarbeit für einzelne Interaktionspunkte. Eine hohe Frequenz ist somit nur mit hohem Aufwand zu erreichen. Der Abstraktionsgrad birgt noch einen weiteren Nachteil. Strukturelle Änderungen verursachen wie beim Programmieren viel Aufwand. Der experimentelle Arbeitsstil wird damit eingeschränkt. Zeitliche Verzögerungen müssen ebenfalls als Zustand modelliert werden. Das erfordert ein Verständnis des ›Renderloops‹ einer ›Gameengine‹. Wie eine kurze Umfrage in der Künstlerabteilung von ➸HappyTuesday ergeben hat, kann dieses Verständnis nicht vorausgesetzt werden.

Das zweite Ziel der Tests ist die Erhebung von Anforderungen zur Funktionalität und Bedienbarkeit. Die selbstgewählten Ziele der Nutzer zeigen, das unter interaktiven Inhalten intuitiv Stufe 3 verstanden wird. Beide Aufgaben fordern das Speichern und Abfragen von Zuständen.
In den Aufgabenstellungen zeigen sich viele weitere Aspekte. Die Darstellung von Grafik und Animation, sowie die Verarbeitung von Nutzerinput. Diese Funktionen finden sich größtenteils in den Nutzerinterviews im folgenden Kapitel wieder.

Folgende Funktionale Eigenschaften werden im Interview nicht genannt:

- (Automatisierte) Wiederholbarkeit von Effekten
- Gruppierungsfunktion

Im Punkt Bedienbarkeit gibt der Test erste Hinweise. Folgende Ideen werden für EFI übernommen:

- Einblendbare Information und Warnhinweise zu einzelnen Effekten
- Timelinebasiertes Arbeiten
- Benennung von Effekten und Eigenschaften wie in gängigen Grafikprogrammen
- Darstellung aller Effekte in der Vorschau
- Speichern von Zuständen mithilfe von Textbezeichnern
- Suchfunktion für Effekte

„Was muss der Editor können?" – Das Featureset

Nutzerinterviews

Als Basis für das Featureset von EFI werden typische Anwendungsfälle aus dem Arbeitsumfeld der Tester aufgenommen.

Zuerst wird mit jedem Tester ein kurzes Interview geführt. Die Nutzer werden gebeten interaktive Inhalte zu nennen, die sie gerne umsetzen würden. Die Umsetzbarkeit sollte nicht beachtet werden. Im weiteren Verlauf beschreiben sie die grundsätzlichen Abläufe innerhalb des gewählten Inhaltes. Zum Abschluss wird nach zwingend erforderlichen Bestandteilen der Software gefragt.

Die genannten Anwendungsbeispiele fußen erwartungsgemäß auf den aktuellen Aufgabenstellungen der jeweiligen Tester. Vereinzelt werden auch zusätzliche Anwendungen beschrieben. Aus den Interviews ergeben sich die im Folgenden diskutierten Kernpunkte. Aufgrund der geringen Anzahl der Ergebnisse ist ein Rückschluss von der Anzahl der Nennungen auf die Wichtigkeit des Features nicht zielführend. Mehrfachnennungen sind aus diesem Grund nicht gesondert gekennzeichnet.

Ergebnis des 1. Interviews

Mögliche Anwendungsgebiete:

- Comicreader
- Interaktives Portfolio
- E-Books mit grafischen Inhalten
- Animierte Shorts [anm. d. Verf. kurze Filmsequenzen ca. 30s bis 5 min.]
- Mini Games
- Infotainment
- Inhalte für digitale Spiele wie Tutorials oder Cut Scenes
- Erstellen der Funktionalität von Interfaces
- Definieren von ›Achievements‹ (Erfolge) eines Spiels
- Erstellen von Mini Games innerhalb eines Spiels
- Wireframes mit „klickbaren“ Interface Prototypen

Aus den oben genannten Anwendungsgebieten lassen sich verschiedene Kernfeatures extrahieren.
Alle Anwendungen erfordern die Darstellung von wechselndem Text oder Bildinhalten. Da die Befragten auch Bewegtbild erstellen, fordert das interaktive Portfolio hier auch die Darstellung von Videodaten.
Eine Außnahme bilden die ›Achievements‹. Auf diesen Sonderfall wird später eingegangen.
Animierte Kurzfilme und ›Cutscenes‹ (Zwischensequenzen) fügen die Darstellung von Animation hinzu, die Verknüpfung zum Film erfordert implizit die Wiedergabe von Ton.

Die Unterscheidung zwischen der geforderten Darstellung von Bewegtbild und der Darstellung von Animation liegt in der Quelle ihrer jeweiligen Erstellung. Bei Bewegtbild handelt es sich um die Reproduktion von Animation welche mittels Drittsoftware erstellt wurde. Beispielhaft sei hier die Wiedergabe von Filmen genannt. Die Darstellung von Animation fordert von der Software, diese selbst erstellen zu können. Dies ermöglicht folglich den Zugriff auf einzelne Elemente der Animation, sodass verschiedene Komponenten zu unterschiedlichen Zeitpunkten abgespielt werden können.

›Tutorials‹ und Interface Prototypen erfordern den Umgang mit Nutzereingaben.
Für Interface Prototypen ist darüber hinaus noch ein Statusspeicher gefordert. Dieser ermöglicht die abhängige Navigation zwischen einzelnen Menüs. Diese Anforderung verdeutlicht sich am Beispiel eines Optionsmenüs. Dieses soll von zwei unterschiedlichen Stellen im ›Interface‹ erreicht werden, und über einen „Zurück“-Button an den jeweiligen Einstiegspunkt zurückführen.

›Achivements‹ erfordern die Möglichkeit auf Ausführung bestimmter Aktionsketten reagieren zu können. Wie schon durch den Interface Prototypen gefordert, benötigen wir die Möglichkeit Zustände zu speichern, und schlussendlich auf Basis logischer „Wenn-Dann“-Entscheidung auszulösen. Abschließend müssen die Aktionen verknüpft werden.

Zusammengefasst ergeben sich folgende Kernfeatures:

- Darstellung von Text und Bild
- Darstellung von Bewegtbild
- Wiedergabe von Ton und Musik
- Erstellung und Wiedergabe von Animation
- Speichern und Abfragen von Zuständen
- Ausführen von Aktionen als Reaktion auf Nutzereingaben
- Ausführung von Aktionen in Abhängigkeit von logischen Operationen

Die Interviewergebnisse auf die Frage nach den Hauptelementen der zu erstellenden Inhalte spiegeln die oben diskutierten Kernpunkte zu weiten Teilen wieder:

Hauptelement des genannten Inhalts:

- Gute Darstellung von 2D Inhalten
- Timeline basierte Animationen
- Tablet Interaktionen
- Interface Gestaltung
- Benutzer Input
- Simulation von Benutzereingaben
- Erklärungen visuell wie auch textlich.
- Erstellen von Interface Designs
- Wireframes
- Screenshots

„Tablet Interaktion“ definiert den Nutzerinput genauer, und weißt darauf hin das auch bekannte Gesten wie ›Swipe‹ oder ›Pinch‹ unterstützt werden sollten.
„Interface Gestaltung“ und „Interface Design“ lassen vermuten, dass die Nutzer direkt in der Software gestalten möchten. Dieser Bereich erfordert eine Menge zusätzlicher Funktionen. Der Editor soll seinen Schwerpunkt auf der Interaktionserstellung haben. Gestalterische Aufgaben werden weiterhin in dafür spezialisierter Software wie *„Adobe Indesign“* oder *„Photoshop“* durchgeführt. Die Resultate können als *Assets* importiert werden.

Als Anforderung ergibt sich daraus eine offene Schnittstelle für unterstützte Importformate. So dass Erweiterung von neuen oder proprietären Formate unterstützt werden.
Die Forderung nach simulierten Benutzereingaben fügt eine neue Funktion hinzu. Nutzereingaben abhängige Aktionen müssen auch ohne Nutzereingabe ausgeführt werden können.
Ein möglicher Anwendungsfall ist im Rahmen des ›Tutorials‹ zu finden. Der Spieler wird aufgefordert einen bestimmten Button zu drücken. Geschieht dies aber nicht, so kann die Aktion durch die Software ausgelöst werden, um anschließend das ›Tutorial‹ weiter zu führen.

Zusätzlich ergeben sich folgende Features:

- Benutzereingaben: ›Swipe‹ und ›Pinch‹
- Simulierten Benutzerinput
- Offene Importschnittstelle

Beispielhafte Abläufe:

Auf Basis der beispielhaften Abläufe, die von den Nutzern skizziert wurden lassen sich noch einige Verfeinerungen herauslesen.

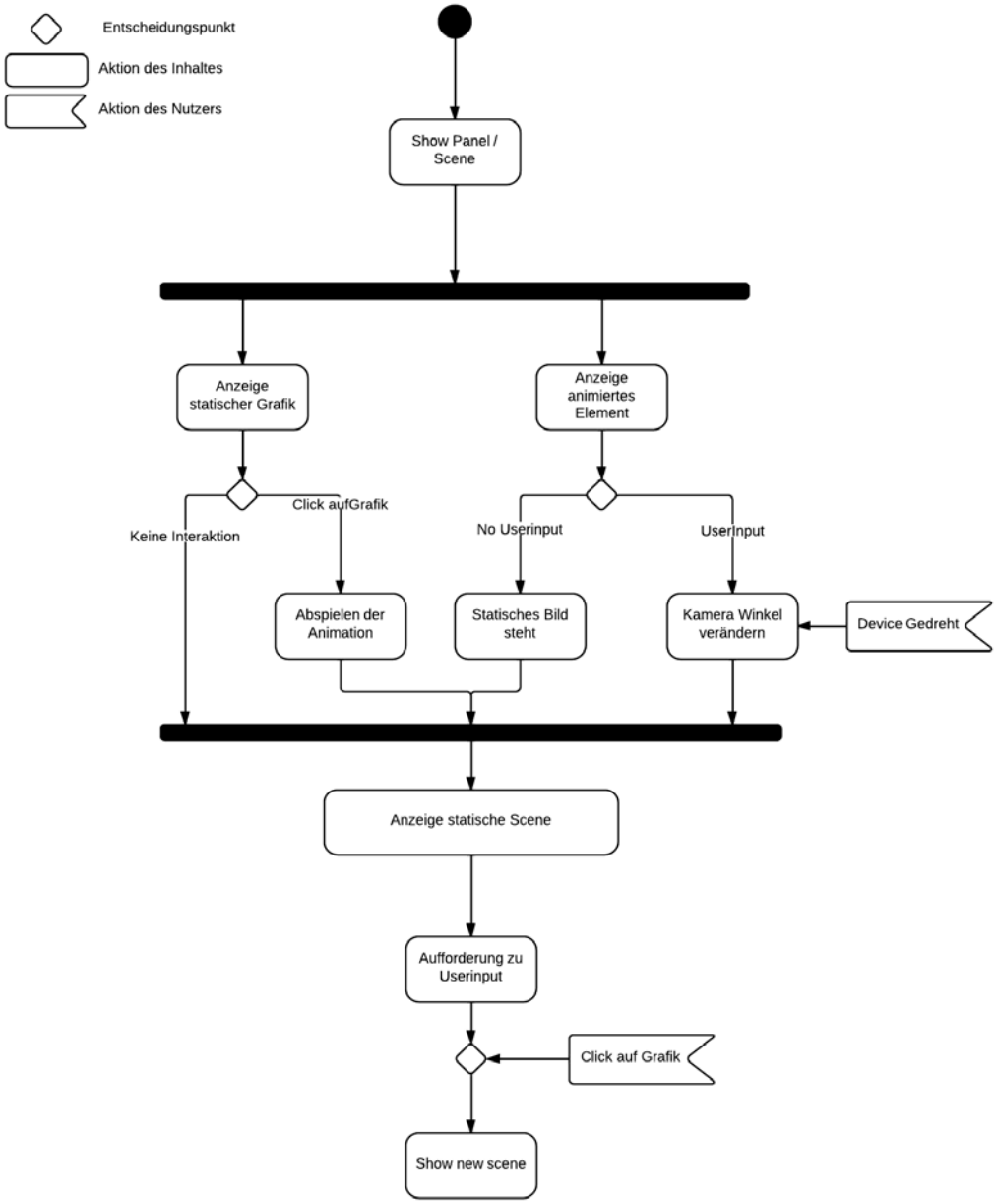


Abbildung 11 : Beispielhafter Ablauf Samuel

In dem ersten Ablauf von Samuel (Abbildung 11) lässt sich erkennen, dass zwischen den abgespielten Animationen keine Unterbrechung durch den Nutzer gewünscht ist.

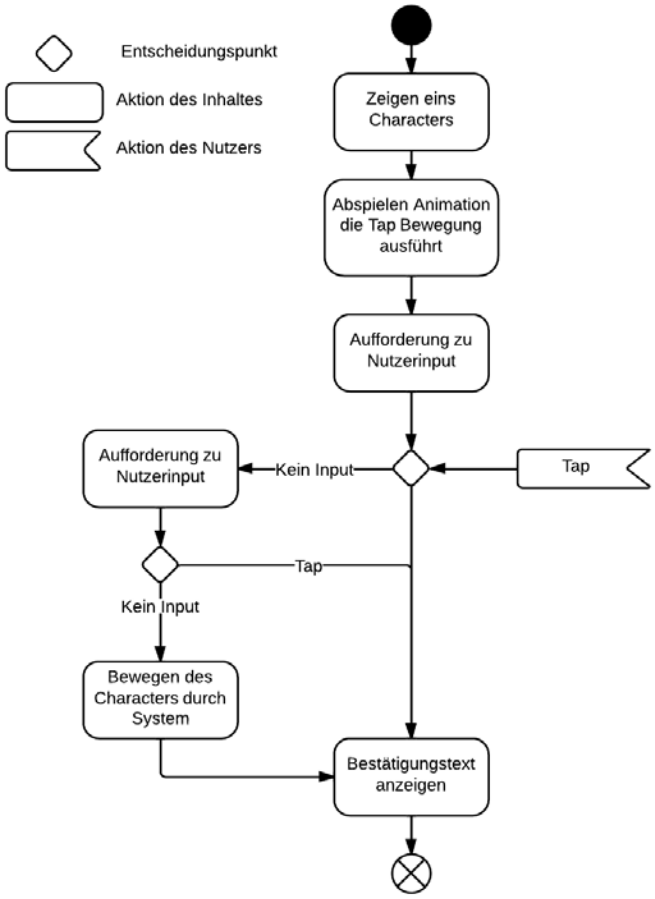


Abbildung 12 : Beispielhafter Ablauf Kevin

Der Ablauf von Kevin (Abbildung 12) zeigt, dass Aktionen pausieren, bis ein Nutzerinput erfolgt.

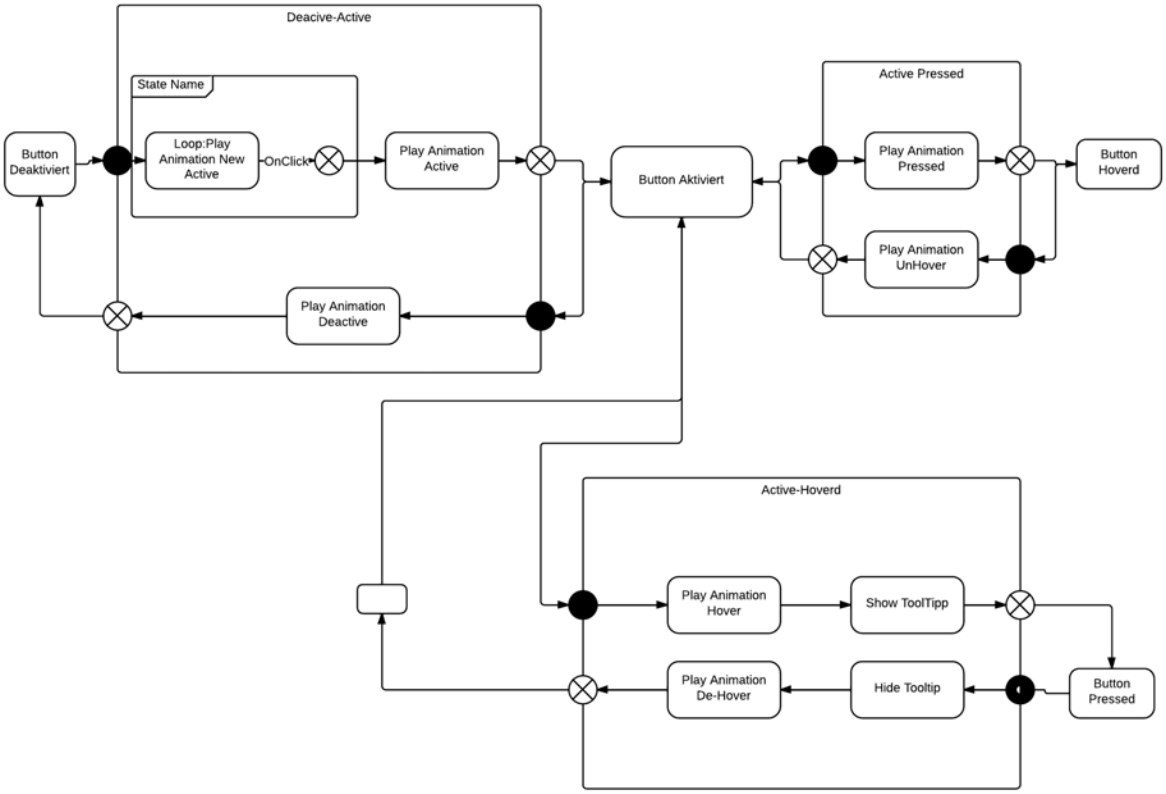


Abbildung 13 : Beispielhafter Ablauf Tino

Gleiches lässt sich in dem Ablauf von Tino (Abbildung 13) erkennen. Die grafischen Änderungen des Buttonstatus resultieren aus dem Nutzerinput. Der Effekt, wenn der Button wieder aktiv wird, erfolgt durch logische ›Events‹ aus anderen Aktionsketten.

Ich füge darraus folgend folgende Funktion hinzu:

- Abspielen von Effekten (Äquivalent zu Aktionen in EFI) in Abhängigkeit von Zeit oder Nutzerinput

Benutzbarkeit

Alle bisher erarbeiten Möglichkeiten sind funktional. Im Folgenden liegt der Schwerpunkt auf der Benutzbarkeit. Im Interview wird nach den zwingend notwendigen Funktionen gefragt. Die Antworten beziehen sich größtenteil auf Benutzbarkeitsanforderungen.

Zwingend erforderliche Funktionen:

- zuverlässige ›Previews‹ (Vorschau des Resultates)
- übliche Animationskurven [Anm. d. V. Hier sind Bezierkurven gemeint]

- gutes Sprite-/Textrendering
- Visuelle Hilfestellungen im Interface (Colorcodes, Abhängigkeiten von Aktionen anzeigen, verständliche Warnmeldungen, Hinweise etc.)
- Batch/Bulk Objektbearbeitungen.
- Einfache Bedienung, nur nötigste Funktionen vorhanden (KISS[36])

Um mögliche Probleme in der Benutzbarkeit und Funktionalität der Software aufzudecken ist ein zusätzlicher Workshop durchgeführt wurden.

Workshops - „Präventiv Obduktion“ [37]

Ziel des Workshops

Dieser Workshop wird im Idealfall kurz nach einem Projektstart ausgeführt. Bei der „Präventiv Obduktion“ werden die Teilnehmer vor die fiktive Tatsache gestellt, das Projekt ist beendet und katastrophal gescheitert. Ziel des Workshops ist es nun, die möglichen Gründe zu benennen.

Bei vielen Projekten werden die Lehren am Ende in einem ›Retrospektivmeeting‹ gezogen. Hierbei werden die Ursachen von Problemen herausgearbeitet.Häufig verfügen Projektmitglieder schon über Erfahrungswerte, welche Probleme innerhalb von Projekten auftreten können. Diese Arbeitsmethode soll diesen Erfahrungsschatz auf das aktuelle Projekt anwenden um typische Fehlerquellen aufzudecken. Für meine Anforderungsanalyse möchte ich mir die gleiche Grundannahme zu Nutze machen. Die Anwender haben demnach verschiedene Erfahrungen mit Problemen die bei einer Software auftreten können. Ziel ist es anhand der Fehlerbeschreibungen typische Arbeitsprozesse der Anwender zu erkennen, und nicht bedachte Fehlerquellen aufzudecken.

Ablauf

Die Teilnehmer bekommen die fiktive fertige Software EFI vorgestellt. Dazu die Tatsache das sie in den letzten 3 Monaten mit dieser Software ein 2D-Puzzle-Spiel (➳HappyTuesday) bzw. ein interaktives Comic (➳Stenarts) erstellt haben. Das fiktive Ergebnis ist ein Produkt mit vielen Grafik und Logikfehlern.

Die Teilnehmer schreiben innerhalb von 15 Minuten in Stichpunkte auf, was in dem Projekt nicht funktioniert hat.Die Stichpunkte werden auf ›Post-Its‹ geschrieben. Ein Stichpunkt pro Zettel. Anschließend stellen die Teilnehmer ihre Stichpunkte vor und pinnen sie an die Wand. Dabei wird sortiert nach Problemen mit der Software und anderen Rahmenbedingungen wie zum Beispiel: »schlechte Kommunikation im Team«. Nach der Einordnung gibt das Team gemeinsam zu jedem Stichpunkt ein Beispiel ab.

[36] Keep-it-simple-stupid Vergl. http://de.wikipedia.org/wiki/KISS-Prinzip | abgerufen am 26.01.2014
[37] Quelle: Game Storming von Daniel H. Pink O’Reilly 2010 S.123

Benannte Probleme

- Bedienung langsam → Häufig benutze Aktionen befinden sich in Untermenüs
- Fehlende Logikfunktionen verursachen umständliche ›work-arounds‹
- „Bugs“ [Anm. d. V. gemeint sind Versagen] in den Effekten sorgen für lange Fehlersuche im eigenen Setup - obwohl das Setup richtig ist
- Fehlende Debugmöglichkeiten machen genannte Fehlersuche langwierig
- Fehlende Strukturierungsmöglichkeiten (Gruppieren von Elementen, Kommentarfunktion, Farbcodierung) sorgen für schlechte Übersicht
- Elemente können nicht modular wiederverwendet werden, dadurch dauern grundsätzliche Änderungen auch wenn sie nur sehr klein sind sehr lange
- Bedienung ist nicht ausreichend dokumentiert
- Benennung der einzelnen Funktionen ist nicht eindeutig bzw. konsistent
- Performance des Werkzeuges sinkt schon bei mittlerer Komplexität

Nicht-softwarebezogene Gründe

- Dokumentation des gewünschten Produktes war nicht eindeutig dadurch kommt es zu vielen Änderungen am Produkt

Die Anwender sind es gewöhnt die Software nicht intuitiv zu verstehen, sondern zu erlernen. Sie fordern dafür die nötigen Informationen. Außerdem erwarten sie Dateien mit vielen Daten. Um die Übersicht nicht zu verlieren, benötigen sie verschiedene Möglichkeiten zum Gruppieren. Der Wunsch nach wiederverwendbaren Modulen entspricht dem schon geplanten Feature der ➸Prefab-Assets. Wie schon in den Softwaretests herausgestellt, sollten die Benennungen der Effekte aus dem Sprachgebrauch der Nutzer stammen.

Zusammenfassende Featureliste

- Darstellung von Text und Bild
- Darstellung von Bewegtbild
- Wiedergabe von Ton
- Erstellen und Wiedergabe von Animation
- Speichern und Abfragen von Zuständen
- Ausführen von Aktionen in Abhängigkeit von Zeit oder Nutzereingaben
- Ausführung genannter Aktionen in Abhängigkeit von logischen Operationen
- Annehmen von Benutzer Input
    - Tablet Interaktionen : Benutzerinput: ›Swipe‹ und ›Pinch‹
    - Simulation von Benutzereingaben

Anforderungen an den Editor

- Zuverlässige Vorschau im Editor
- Übliche Animationskurven
- ›Timeline‹ basierte Animationen

- Gutes Sprite/Textrendering
- Visuelle Hilfestellungen im Interface
    - Kurzdokumentation der Effekte direkt im Editor
    - Warnhinweise bei bekannten Fehlerquellen.
    - Farbgruppierungen
    - Darstellung und Hinweise auf Abhängigkeiten
- Batch/Bulk Objektbearbeitungen.
- Einfache Bedienung
    - nur nötigste Funktionen vorhanden (KISS)
- Offene Import-Schnittstelle
- Offene Export-Schnittstelle
- Fehlerfreie Effekte [um Versagen vorzubeugen]
- Debugmöglichkeiten
- Gute Performance
- Häufig benutzte Funktionen direkt erreichbar

„Wie entwickelt man den Editor?“ – Das Softwaredesign

In der Planungsphase können Benutzer nicht alle Anforderungen erfassen. Das bis zu diesem Zeitpunkt erarbeitet Featureset lässt höchstwahrscheinlich viele essentielle Elemente aus. Die Bewertung eines lauffähigen Programms durch den Benutzer ist aussagekräftiger. Dabei kann das Programm rudimentär gestaltet sein. Außerdem sind Benutzer auf dieser Basis schneller in der Lage wichtige fehlende Funktionen zu benennen. Deswegen wird für EFI im ersten Schritt ein Prototyp mit minimalem Featureset entwickelt. Auf Basis dieses Prototypen können zukünftig weitere Nutzertests durchgeführt werden.
In der herkömmlichen Softwareentwicklung versteht man unter einem Prototypen häufig ein Wegwerfprodukt. Der Zweck besteht in der Evaluation weniger, bestimmter Aspekte des zukünftigen Projektes. Erweiterbarkeit und Wartbarkeit spielen dabei eine untergeordnete Rolle. Im Gegensatz dazu, muss aufgrund der langfristigen Zielsetzung die mit EFI verfolgt wird, der Wartbarkeit eine hohe Priorität zugewiesen werden.
Agile Softwareentwicklung hat in den letzten zehn Jahren den Sprung zum ›Mainstream‹ vollzogen. Dabei ist der Grundgedanke, Software nicht mehr ›Up-Front‹ (von Beginn an) vollständig zu Planen, sondern nur die nächsten kleinen ›Iterationen‹ (Schritte) zu evaluieren. Die regelmäßige Rückmeldung aus Nutzertests wird dabei in die nächsten Planungsschritte eingearbeitet. Der Prozess ist darauf optimiert, mit stetig geänderten Anforderungen zu arbeiten. Der Prototyp in dieser Arbeit wird als erster Iterationszyklus eines agilen Softwareprojektes verstanden.
Bei einer nutzernahen Anwendung wie EFI, mit starkem Fokus auf Bedienbarkeit bietet sich die iterative Arbeitsweise an. Um trotzt wechselnder Anforderungen ein solides Softwaredesign zu

erhalten, werden für die Entwicklung Prozesse aus der bewährten agilen Methode „extreme Programming“[38] eingesetzt.
Einige der Prozesse sind auf Teamarbeiten ausgelegt, die für diese Arbeit nicht übernommen werden. Die Prozesse die für die Entwicklung übernommen werden sind:

- Test-Driven-Development
- Continuoues Integration
- On-Site-Customer
- Refactoring
- Itteratives Design

Auf Basis von Unittests werden nur funktionale Bestandteile der Software abgedeckt. Das grafische Interface wird, aufgrund der unzureichenden Testmöglichkeiten, nicht automatisiert getestet. Die Kooperation mit Happy Tuesday und Stenarts ermöglicht es über den gesamten Entwicklungszeitraum Feedback von den Nutzern zu erhalten. Die Notwendigkeit zum Refactoring ergibt sich aus dem iterativen Design. Nach den ersten Nutzertests ist mit Umstellungen im Softwaredesign zu rechnen. Einige absehbare Änderungen werden direkt im Kapitel ➙Systemarchitektur beschrieben. Die Abdeckung der funktionalen Bestandteile durch automatisierte Tests hält das Risiko für diese Umstrukturierungen gering. Durch die kontinuierliche Integration sind auch entstehende Probleme im grafischen Interface schnell zu entdecken.
Als Versionskontrollsystem wird „*Git*“[39] eingesetzt. Aufgrund der einfachen Möglichkeit ›Branches‹ zu erstellen, unterstützt es den iterativen, featuregetriebenen Arbeitsprozess.

Der Prototyp

Um im begrenzten Zeitrahmen dieser Arbeit, einen Prototypen mit den benötigten Features zu erstellen, wird ein leistungsfähiges Framework benötigt. Im Idealfall übernimmt das Framework alle der folgenden Aufgaben:

- Darstellung eines GUI
- Import und Wiedergabe von Grafiken und Animationen
- Import von Wiedergabe von Ton und Musik
- Export zu einer lauffähigen Applikation
- API- oder Script- Schnittstelle für Funktionen und GUI
- Verarbeitung von Tastatur und Mauseingaben
- *Asset* ähnliche Verwaltung der importierten Daten
- Abspeichern und Laden von *Asset*-Anordnungen
- Modifikation von *Asset*-Anordnungen
- Erstellen von Animationen

[38] Vergl. http://en.wikipedia.org/wiki/Extreme_Programming
[39] http://de.wikipedia.org/wiki/Git

Unterstützt das Framework alle oben genannten Punkte, kann die Programmierung vollständig auf die relevanten Funktionen der Effekte und Effekterstellung konzentriert werden. Das gewünschte Featureset legt eine Betrachtung von Spieleeditoren nahe.

Unity3D

Unity3d[40] ist ein interaktiver Editor zum Erstellen von digitalen Spielen, für Mobile-, Desktop- und Konsolenanwendungen. Mit Hilfe eines eigenen Browerser-Plug-Ins ist auch der Export für den Webbrowser möglich. *„Unity3d“* ist in der Basisausführung frei verfügbar.

Der hauptsächliche Workflow in „*Unity3d“* besteht aus dem Import von Grafikdaten, zumeist aus dritt Software wie „*Photoshop“* oder „*3ds-Max“*. Auf diesem Weg importierte Elemente werden in einer ›Scene view‹ (entspricht der ›Stage view‹ in EFI) angeordnet und animiert. Jedes Element kann Scripte erhalten, die über eine Anbindung an „*MonoDevelop“* programmiert werden können. Diese Scripte können Verhalten und Darstellung der Elemente verändern. „*Unity3d“* bietet ein Vorschaufenster, in dem der erstellte Inhalt getestet und mit klassischen Debugmethoden, wie Logs oder Breakpoints, Scripte korrigiert werden können. Die erstellte Komposition der Elemente kann als sogenannte ›Szene‹ abgespeichert und geladen werden. Abschließend bietet „*Unity3d“* den automatisierten ›build‹ zu einer eigenständigen Anwendung. „*Unity3d“* folgt der selben Assetbereitstellungsstrategie wie sie in der Definition von EFI beschrieben ist. Ebenso ist die Funktionalität von ➸Prefabs (vordefinierte Referenzobjekte) vorhanden und wird für die Umsetzung von EFI genutzt. „*Unity3d“* bietet umfangreiche Möglichkeiten den Editor durch eigene Menüs und Fenster zu erweitern. Diese Möglichkeit wird genutzt um das grafische Interface des Prototypen zu realisieren.

Jedes Element innerhalb einer ›Szene‹ wird bei „*Unity3d“* als *GameObject* bezeichnet. Diese Objekte können mit beliebig vielen Komponenten erweitert werden. Die Komponenten bestimmen die Funktion der Objekte. Ein *GameObject* mit der Komponente *AudioSource* (Audioquelle) kann zum Beispiel Sounddateien wiedergeben. Ein *GameObject* mit der Komponente Animation kann die Werte aller anderen Komponenten über die Zeit verändern usw. Die Scripte werden in C# programmiert. Jede Klasse kann von der Basiskomponente *MonoBehaviour* erben und dann als Komponente auf *GameObjects* benutzt werden. Diese Funktion wird in EFI benutzt um die Funktionalität der Effekte umzusetzen.
Wie für eine Gameengine typisch, arbeitet „*Unity3d“* intern mit einem ›Renderloop‹. Dieser wird so oft wie möglich pro Sekunde aufgerufen. Alle Komponenten können sich in diesen ›Renderloop‹ einhängen, und somit mehrfach pro Sekunde eine ›Update‹ erhalten. Diese Funktion wird in EFI benutzt um die Zeitleiste zu implementieren. „*Unity3d“* bietet alle vom Framework geforderten Funktionen. Die bereits vorhandene Vorschaumöglichkeit stellt eine zusätzliche Vereinfachung dar.

Übersicht der von Unity bereitgestellten, benötigten Funktionen:

- Laden und Speichern von Szenen
- Import und Wiedergabe von 2d-, 3d- Grafiken und Sound

[40] http://unity3d.com/

- Interne Vorschaufunktion
- Erstellen und Wiedergabe von Animationen
- Raycast und Kollisionserkennung
- Bearbeiten von Benutzereingaben über : Tastatur, Gamepad, Touchdevices, Mouse
- Debug-Möglichkeiten
- Schnittstelle für Editorerweiterungen (GUI)
- Bereitstellung des ›Renderloops‹
- Export als Android App, iOS App, Windows Exe, OSX App oder Webanwendung für den Unity eigenen Webplayer

In einem modernen Softwareumfeld ist der ›Community‹ ein hoher Stellenwert bei der Technologieauswahl einzuräumen. Der klassische Support via Email oder Telefon wird auch von *„Unity“* noch angeboten. Ein gut besuchtes Forum ist im Problemfall meist deutlich schneller und umfangreicher. In einer aktiven ›Community‹ entstehen darüber hinaus eine Vielzahl an Erweiterungen, kleinen Workflowverbesserungen oder auch Diskussionen die zur besseren Dokumentation bestimmter ›Features‹ beitragen.
„Unity“ verfügt über eine sehr aktive ›Community‹: Mit einem eigenen *Stackoverflow* Ableger als Forum: *UnityAnswers*[41]. In einem eigenen Wiki werden Erweiterungen und Tutorials gesammelt: *UnifyWiki*[42]. Unity bietet einen, direkt in den Editor integrierten, Assetstore. Hier werden von Nutzern erstellte Inhalte, Scripte und Editorerweiterungen (je nach Model: gegen einmalige Zahlung oder frei) zum Import in das eigene Projekt bereitgestellt.

Auf Basis der Evaluation besagter Vorteile fußt meine Entscheidung den Prototypen als Editorerweiterung für *„Unity3d“* umzusetzen. Verwendet wird die derzeit aktuelle Version 4.2 .

Anmerkung zu Lizenzen

Im Rahmen der Umsetzung des Prototypen werden Erweiterungen aus dem Assetstore benutzt. Hierbei handelt es sich um platzgebundene Lizenzen, die im entsprechende Ordner im Projekt eingesehen werden können. Die benannten Module dürfen nicht außerhalb des Prototypen verwendet oder vervielfältigt werden.

Module aus dem Assetstore (nicht freie Software)
- NGUI (zur verbesserten Darstellung von 2D Grafiken)
- Playmaker (eigenständige Editorerweiterung - eingesetzt als Vergleichssoftware)
- Autopilot (Erweiterung zum Autobuilden und Upload von iOS Apps eingesetzt für Continuoues Integration Builds)

Folgende Module besitzen eine OpenSource Lizenz und können entsprechend ihrer Lizenzen verwendet werden:

[41] http://answers.unity3d.com/index.html
[42] http://wiki.unity3d.com/index.php/Main_Page

- LitJSON (JSON Bibliothek)
- SharpUnit (Unittest in Unity für C#)
- NBuilder (Belegen von Klassen Membern mit zufälligen Werten)

Alle nicht selbsterstellten Scripte befinden sich im Ordner: ⤡Assets/Frameworks/External/

Einschränkungen für den Prototypen

Zwischen einigen Anforderungen besteht der Hauptunterschied, in der Form des benötigten Exportformates (Beispiel interaktives Portfolio zu E-Book).
Dieser Bereich der Anforderungen wird im Rahmen des Prototypen nicht weiter bearbeiten. Diese Beschränkung ist auch dahingehend sinnvoll, um mögliche Beschränkungen durch das Ausgabeformat nicht zu Beschränkungen der Software werden zu lassen. Gleiches gilt für die offene Import Schnittstelle. Im Rahmen des Prototypen werden nur Ein- und Ausgabeformate unterstützt, die „*Unity3d*“ zur Verfügung stellt.

Alle grafischen Darstellungsmöglichkeiten können sowohl im zweidimensionalen, als auch im dreidimensionalen Raum betrachtet werden. Durch die Erweiterung auf drei Dimensionen steigt die Komplexität der Steuerung und Darstellung. Die Erweiterung bietet keine neuen Interaktionsmöglichkeiten im Rahmen der Definition. Der Prototyp wird deswegen auf 2d Inhalte beschränkt. Ähnlich verhält es sich mit der Wiedergabe von Bewegtbild. „*Unity3d*“ kann nur mit deutlich höherem Aufwand vorgerenderte Filme wiedergeben. Die Einschränkung keine vorgerenderten Filme abspielen zu können, hat ebenfalls keinen Einfluss auf die Interaktionsmöglichkeiten.

Systemarchitektur

Für die Aufgabe bieten sich zwei Herangehensweisen an. Die Erste ist wie in ➸Playmaker eine Flowgraph-Architektur. Hierbei werden Effekte als Zustände verstanden und miteinander verkettet. Die zweite Variante ist der Aufbau über eine Zeitleiste, wie bei ➸Adobe Edge. Beide Möglichkeiten werden im Folgenden kurz abgewogen.

Timeline-Architektur

Das System mithilfe einer globalen ›Timeline‹ (Zeitleiste) aufzubauen ist naheliegend. Dadurch erhält man effiziente Kontrolle über den globalen Zeitablauf. Die Darstellung ist für den Nutzer gewohnt. Dies zeigt sich auch in der direkten Forderung der Zeitleiste als Feature: ➸›Timelinebasierte Animation‹. Ähnlich zu „*Adobe Edge*“ werden die Effekte in der Zeitleiste angeordnet und modifiziert. Die Effekte starten, wenn die globale Zeit den festgelegten Startzeitpunkt erreicht. Mithilfe von Effekten die die Wiedergabegeschwindigkeit und den momentanen Zeitpunkt modifizieren können, werden Stopp- und Sprungpunkte umgesetzt. Der Nachteil an diesem System: Für logische Abhängigkeiten existieren keine Modellierungsmöglichkeiten.

Flowgraph-Architektur

Als zweite Variante ist die Modellierung als endlicher Automat zu überlegen. Logische Abhängigkeiten können in dieser Struktur übersichtlich präsentiert werden. Diese Struktur wird von „*Playmaker*“ eingesetzt. In „*Playmaker*“ ist es nicht möglich zwei Aktionsketten parallel ablaufen zu lassen. In dem nichtdeterministische Automaten benutzt werden, kann diese Einschränkung aufgehoben werden.
Der Nachteil dieser Struktur besteht in der fehlenden Möglichkeit Zeitverläufe auszudrücken. Der erste Softwaretest mit Samuel hat gezeigt, dass häufig kurze Animationen zeitlich aufeinander abgestimmt werden. Spezielle „Warte-Zustände“ können eingerichtet werden, um diesen Nachteil abzuschwächen. Eine globale Abstimmung der Abläufe, lässt sich damit jedoch nur mit hohem Aufwand realisieren.

Beide Architekturen bilden isoliert keine tragfähige Grundlage für die Software. Eine vorteilhafte Struktur kann mit der Kombination beider Ansätze erreicht werden. Ausgangspunkt ist die Timeline-Architektur. Diese wird um Eigenschaften der Flowgraph-Architektur erweitert. Dazu erhalten die auf der Zeitleiste angeordneten Effekte zusätzlich einen ›Trigger‹ (Auslöser).
In der reinen Timeline-Architektur ist dieser ›Trigger‹ immer die Zeit (›OnTime‹). Als weiterer Auslöser wird ›OnExternal‹ hinzugefügt. Dadurch startet der Effekt erst, wenn er von einem anderen Effekt aufgerufen wird. Die Effekte können somit wie in der Flowgraph-Architektur verkettet werden. Die Kontrolle über den Zeitablauf bleibt wie bei der ›Timeline‹ erhalten.
Der Nachteil dieser Variante besteht in der schwierigen Darstellung. Die Effekte werden weiterhin auf einer Zeitleiste angeordnet. Dem Nutzer wird dadurch suggeriert das der Effekt zu einem bestimmten Zeitpunkt abgespielt wird. Wird der Effekt aber extern gestartet, so entspricht der tatsächliche Ausführungszeitpunkt nicht der Position auf der Zeitleiste.

Behaviour-Tree-Architektur

Behaviour-Trees können in ihrer Struktur bereits als Kombination der genannten Varianten angesehen werden. Behaviour-Trees bieten grundsätzlich die gleichen Möglichkeiten wie ein nichtdeterministischer endlicher Automat. Das zusätzliche hierarchische Update aller ›Nodes‹ im Graphen ermöglichen ein globales ›Timing‹. Dazu muss die Wurzel des Graphen als Zeitleiste verstanden werden. Diese aktualisiert den Graphen in regelmäßigen Abständen.
Der Nachteil an dieser Variante ist die höhere Abstraktion des Aufbaus. Es existieren keine direkten Abhängigkeitsketten mehr. Zum Beispiel „Wenn die Brücke kaputt ist → dann fällt der Charakter ins Wasser“, kann nicht mehr einfach als Verkettung dargestellt werden, sondern wird in zwei parallele Stränge des Graphen eingeordnet(➙Abbildung 14). Dieser Abstraktionsweg kann von einem Benutzer aus dem kreativen Arbeitsumfeld nicht verlangt werden. Daher müssten sich die interne Systemmodellierung und die visuelle Repräsentation stark unterscheiden. Das würde einen sehr komplexen Controller (MVC vorausgesetzt) benötigen.

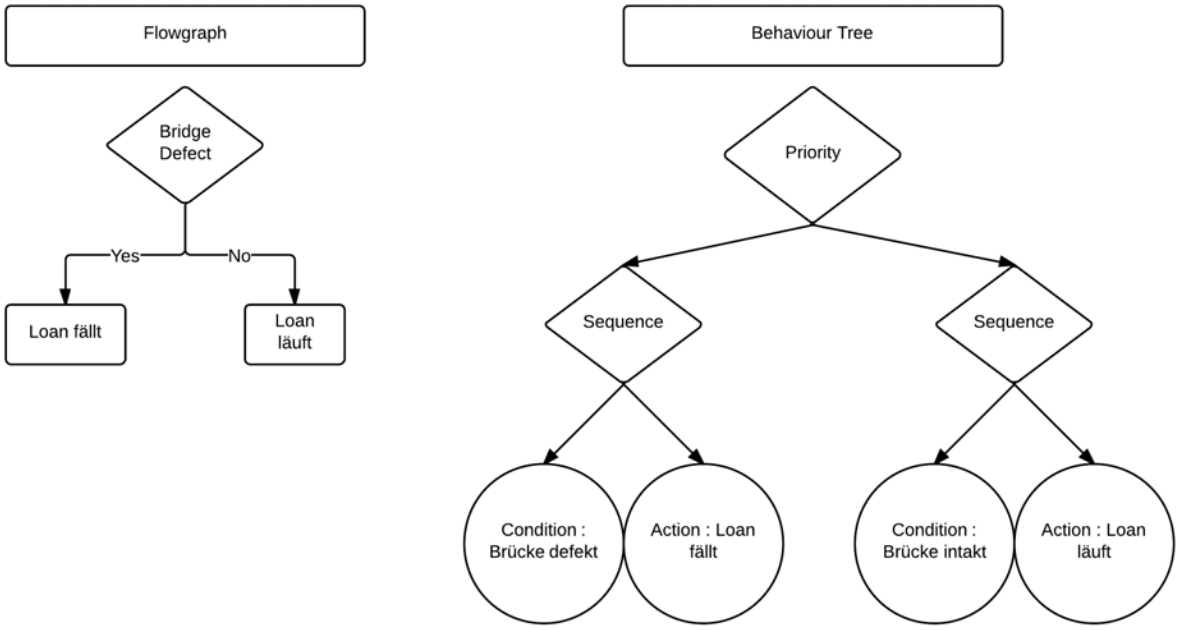


Abbildung 14: Vergleich „Wenn-Dann“-Abfrage zwischen Flowgraph und Behaviour-Tree

Alle genannten Systeme besitzen verschiedene Vor- und Nachteile. Welches der drei Systeme am geeignetsten ist, hängt von den zukünftig geforderten Features und Anwendungsgebieten ab. Derzeit beschränken sich die interaktiven Inhalte auf wenige zustandsabhängige Interaktion und arbeiten verstärkt mit dem Zeitablauf des Inhalts. Wie in den Softwaretests gezeigt, unterstützt aktuelle Software zustandsbasiertes Arbeiten nur eingeschränkt. Aufgrund der neuen Möglichkeiten die EFI bietet, könnten Inhalte in Zukunft deutlich komplexer und zustandsorientierter werden. Dadurch wäre eine Flowgraph-Architektur zu rechtfertigen. Sollten hingegen die eingebauten Möglichkeiten für die Timeline-Architektur ausreichen, so ist diese aufgrund des direkten Nutzerwunsches vorzuziehen. Halten sich beide Elemente die Waage, so wäre der erhebliche Mehraufwand für die Behaviour-Tree-Architektur gerechtfertigt.

Die gewählten Softwareprozesse geben in diesem Fall eine klare Richtlinie vor. Iteratives Design ist eine der eingesetzten Methoden. Das erste Design basiert deswegen auf der Kombination aus ›Flowgraph‹ und ›Timeline‹. Dieser Ansatz entspricht dem direkten Nutzerwunsch. Bei dem gewählten Grundaufbau lässt sich diese Architektur bei Bedarf zu einem Flowgraphsystem umbauen. Der erste Ansatz gestaltet sich wie folgt:

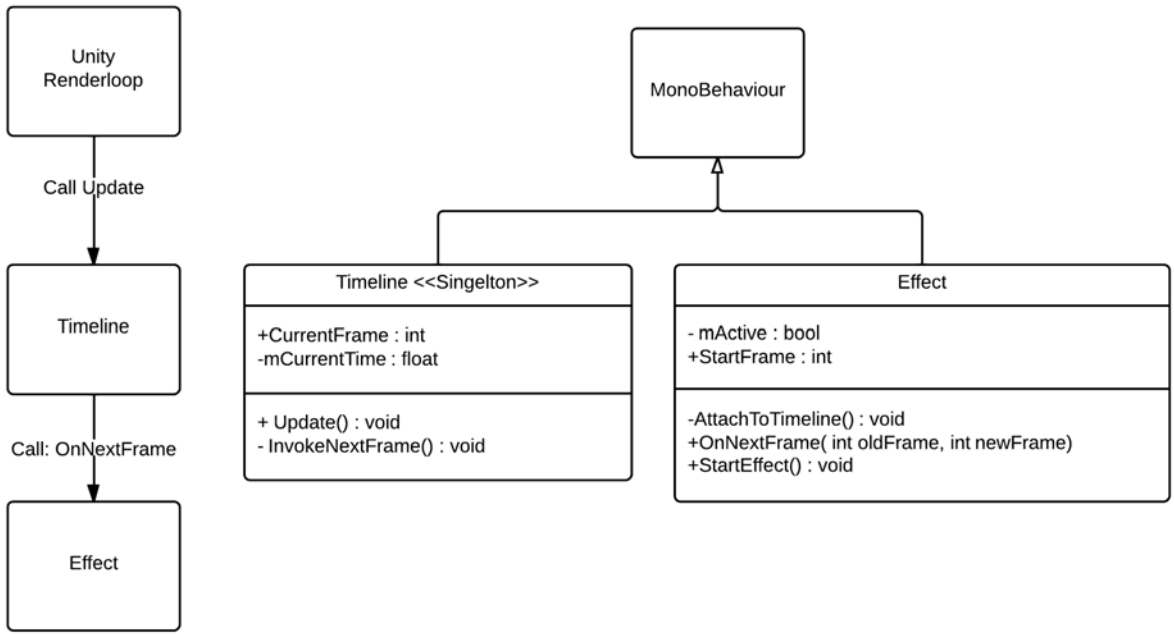


Abbildung 15: Grundlegende Klassenstruktur: Timeline-Architektur

Wie schon erwähnt ruft der ›Unity-Renderloop‹ die ›Update‹ Funktionen auf allen Komponenten auf. Dies geschieht im Normallfall zwischen 30 und 200 Mal pro Sekunde. Innerhalb dieser ›Update-Routine‹ besteht Zugriff auf die Zeitdifferenz zwischen den Aufrufen. Das ganze System wird, wie im Film- und Anmationsbereich üblich, framebasiert sein. Ein ›Frame‹ entspricht 1/30 Sekunde. Die ›Timeline‹ wird innerhalb der ›Update-Methode‹ die Zeitdifferenzen aufsummieren, bis 1/30 Sekunde überschritten ist. Dann wird auf allen angehängten Effekten die Funkion *OnNextFrame* aufgerufen. Innerhalb dieser Routine wird die Funktionalität der Effekte implementiert. Zum Beispiel die Darstellung einer Animation. Zusätzlich erhalten die Effekte einen Status *Active*. In der ersten Implementierung wird dieser immer dann gesetzt, wenn das aktuelle ›Frame‹ der ›Timeline‹ größer als das Startframe des Effektes ist.
Durch die Abkopplung der Effekte vom ›Unity-Update‹ bleibt die Möglichkeit offen die Architektur weiter zu einem Behaviour-Tree auszubauen. In diesem Falle würde *OnNextFrame* durch die Updatemethode des Graphen ersetzt werden. Die Effekte würden als Aktionen des Graphen ausgeführt werden. Dazwischen können die Ablaufkontrollen des Behaviour-Trees geschaltet werden.
Die *StartEffekt* Funktion erlaubt es die Effekte wie Zustände in einem Automaten zu behandeln. In diesem Fall würde der *Active* Status nicht mehr abhängig von der Zeit geändert, sondern durch einen weiteren Effekt.

Die Features aus EFIs Anforderungsliste werden in diesem System als Effekte umgesetzt. Im ersten Entwurf ergibt sich dadurch folgender Klassenaufbau.

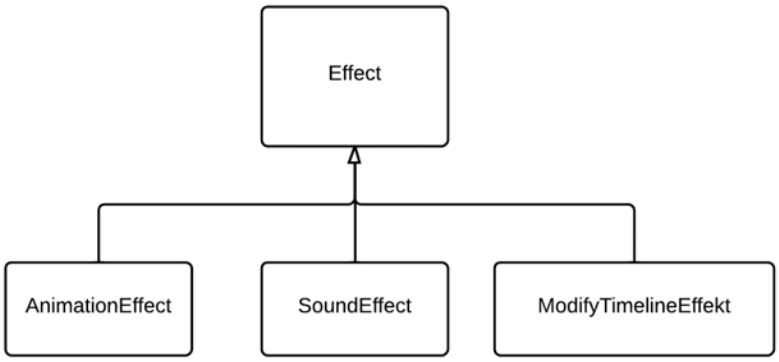


Abbildung 16: Klassenstruktur: Effekte

Mit diesem einfachen Design lassen sich bereits interaktive Inhalte der Stufe 2 erstellen. Damit ist die Grundstruktur für die Funktionalität aufgestellt. Es fehlt noch die visuelle Repräsentation sowie das Datenmodel. Aufgrund dieser Struktur bietet sich das Model-View-Controler Pattern an. ›Controler‹ ist in diesem Falle der Effekt wie bisher beschrieben. Als ›View‹ wird eine ›Unity-Editor-Fenster-Klasse‹ erstellt. Als Model wird eine Datenklasse ausgelagert. Die Struktur sieht dann wie folgt aus:

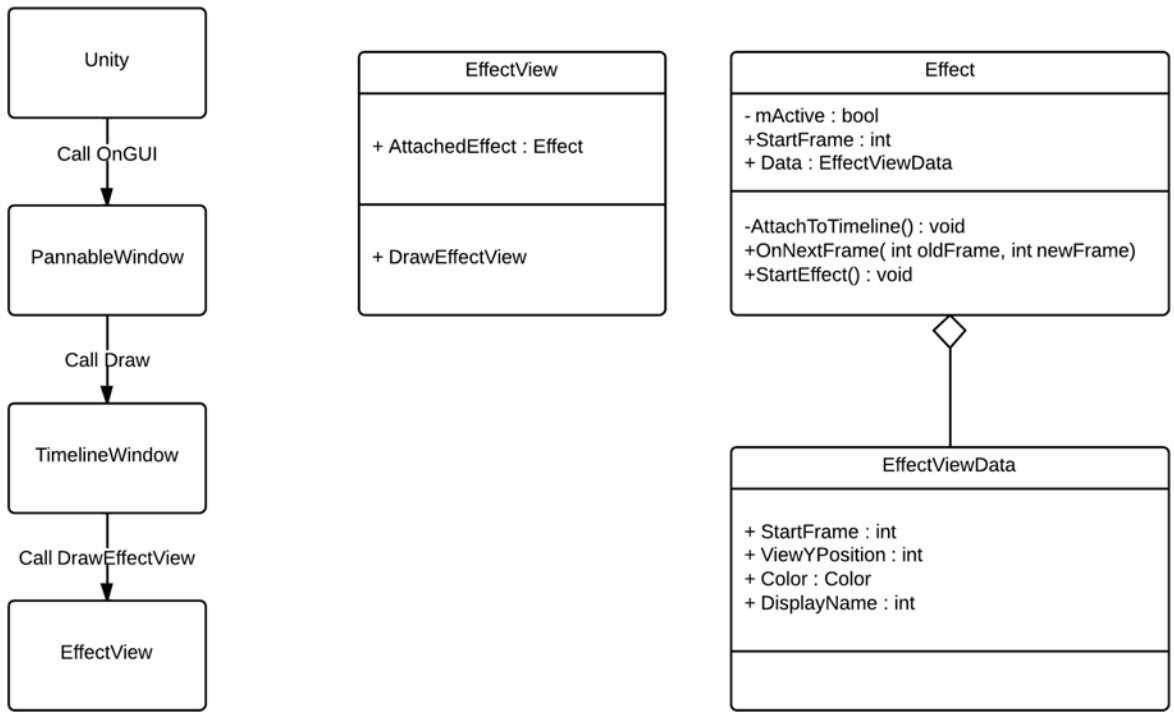


Abbildung 17: Klassenstruktur Effekt ›View‹

Wie schon erwähnt bietet *„Unity“* die Möglichkeit eigene Editorfenster zu erstellen. Diese besitzen eine *OnGUI* Funktion, die ähnlich zu der ›Update‹ Funktion der Komponenten

funktioniert. Mit jedem Aufruf wird das Fenster neu gezeichnet. Standardmäßig besitzen „Unity-Editorfenster“ keine Zoom- oder Verschieben-Funktion. Um eine endlose Arbeitsfläche wie in Grafikprogrammen üblich zu erstellen, werden diese Funktionen benötigt. Deswegen besitzt das *Timelinewindow* eine Basisklasse, in der alle entsprechenden Funktionen implementiert sind. *„Unity“* speichert alle als *serialisierbar* gekennzeichneten Daten von Komponenten automatisch zusammen mit der ›Szene‹ ab. Der ›Controler‹ ist eine Komponente. Deswegen bietet es sich an, das Datenmodel in den ›Controler‹ zu integrieren. Der Editor sieht nach dieser ersten Iteration wie folgt aus:

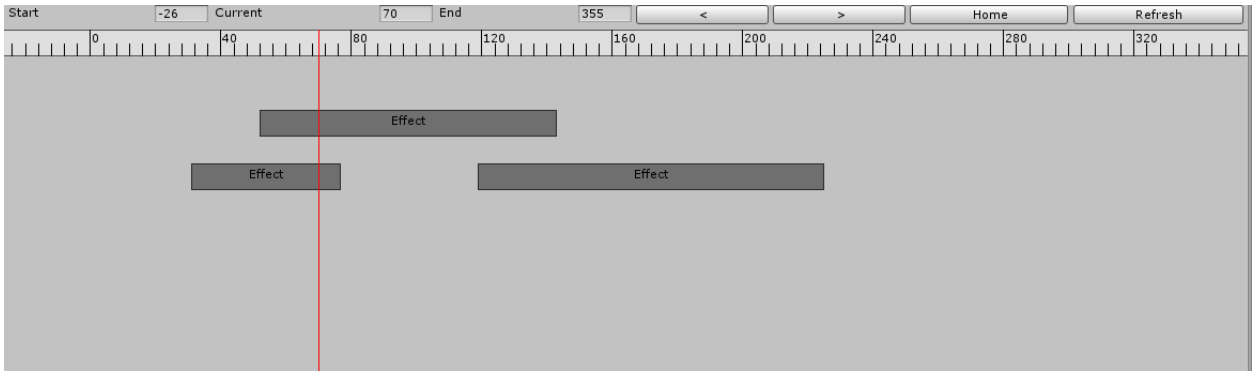


Abbildung 18: Screenshot von EFI nach der ersten Iteration

Die Effekte werden als einfache Rechtecke dargestellt. Beginnend am Startframe. Die Breite der Box wird durch die Länge des Effektes bestimmt. Die rote Line gehört zur ›Timeline‹ und markiert den aktuellen ›Frame‹. Gleichzeitig kann diese Line bewegt werden um den aktuellen ›Frame‹ zu verändern. Im Vorschaumodus wird diese Line durch den ›Unity-Renderloop‹ automatisch bewegt.
Diese Ansicht bietet eine schnelle Übersicht über die Animationen und den Ablauf der Effekte. Allerdings lassen sich noch keine weiteren Eigenschaften wie zum Beispiel der Name oder die Farbe einstellen. In der folgenden Iteration wird deswegen ein eigenes Fenster eingerichtet, das alle gemeinsamen Werte der momentan selektierten Effekte anzeigen und modifizieren kann. Zusätzlich werden verschiedenen Komfortfunktionen hinzugefügt. Die Effekte können danach über ein Kontextmenü erstellt werden. Sie können mit einem Rahmen selektiert, sowie mit Tastenkürzeln dupliziert und gelöscht werden.

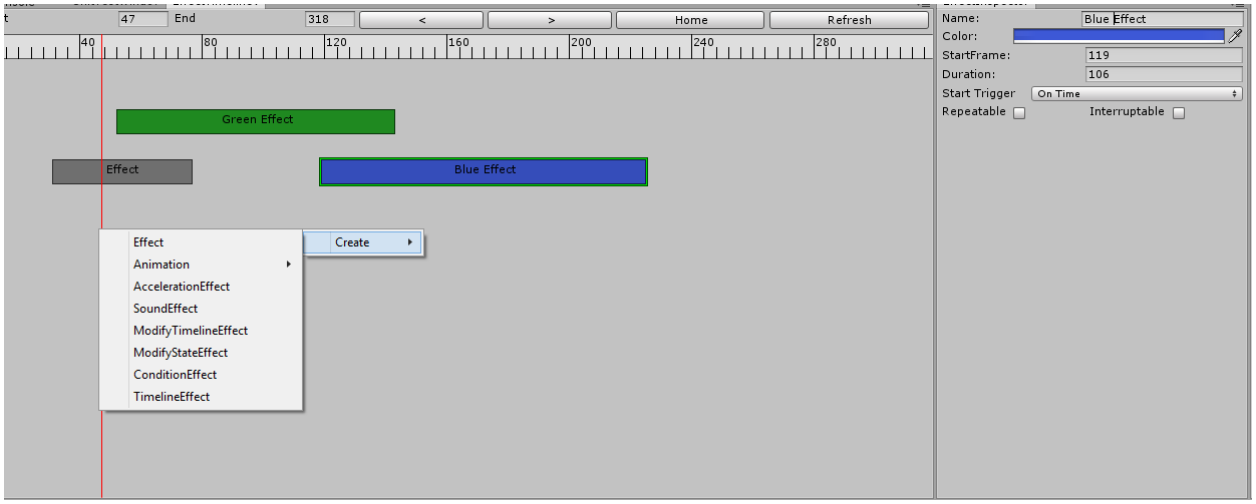


Abbildung 19: Screenshot EFI nach der zweiten Iteration

Nach der Implementierung des Basiseffektes sowie den Effekten für Animation und Sound, kann der Prototyp für Nutzertests eingesetzt werden. Diese ersten Tests zeigen, dass die Bedienung der ›Timeline‹ dem gewohnten Muster entspricht. Farben und Benennung werden intensiv benutzt. Bei hoher Zoomstufe (weit herraus gezoomt) ist das genaue navigieren des aktuellen ›Frames‹ schwierig. Als Resultat werden deswegen die beiden Buttons: „Frame vor (<)“ und „Frame zurück (>)“, die in dem Screenshot über der Zeitleiste zu sehen sind, eingebaut.
Über die Architekturfrage kann anhand von diesem Stand noch nicht entschieden werden. Die dafür relevanten Funktionen: Speichern und Laden von Bedingungen sind noch nicht implementiert. Diese folgen in der nächsten Iteration. Wie schon erwähnt wird dazu das reine Timelinesystem aufgebrochen. Effekte erhalten die Möglichkeit andere Effekte zu starten. Wie im Folgenden skizziert:

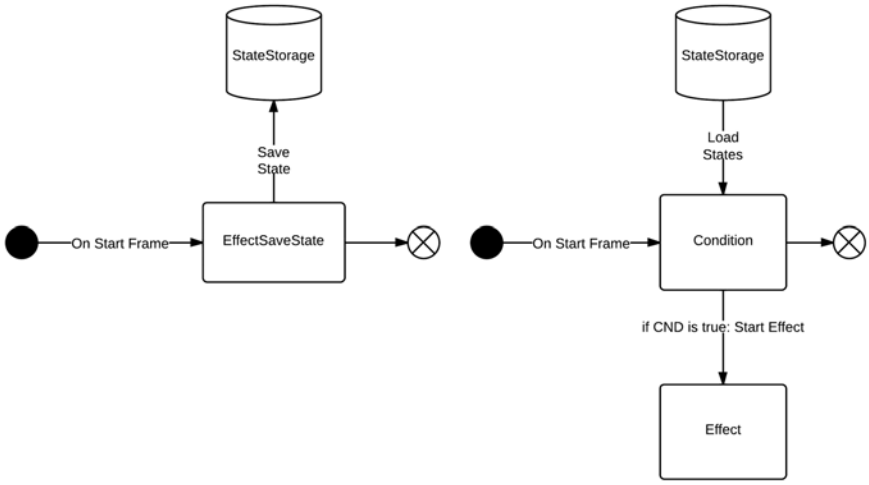


Abbildung 20: Struktur zum Laden und Speichern von Zuständen in EFI

Zusätzlich wird ein Speicher für die Zustände benötigt. Dieser sollte sowohl persistent als auch temporär agieren können. Aufgrund der gutlesbaren und leichtgewichtigen Struktur wird bei dem permanenten Speicher eine JSON formatierte Textdatei eingesetzt.
Die Zustände werden als einfache Textbezeichner gespeichert. Es stellt sich aber die Frage, wie die Bedingungsverknüpfung für die Anwender verständlich dargestellt werden kann. Im Gegensatz zu dem ➙Abspeichern der Informationen bieten die bisherigen Nutzertests dazu keine Informationen.

Deswegen wird ein weiterer Nutzertest durchgeführt. Die Nutzer werden dafür an einen Papierprototypen gesetzt. Zur Verfügung steht verschiedenes Material um logische Operationen auszudrücken, wie auf der nächsten Abbildung zu sehen ist:

Abbildung 21: Papierprototyp für Nutzertest: „Logische Operationen“

Zusätzlich bekommen die Nutzer nacheinander folgende Sätze vorgegeben. Diese sollen mit dem gegebenen Material ausgedrückt werden.

»Wenn die Brücke kaputt ist oder Loan springt, dann starte die Animation *Loan_fall.animation*.«

»Wenn es regnet und Loan einen Regenschirm hat, oder die Sonne scheint und die Brücke nicht kaputt ist, dann zeige die Grafik *Loan_happy.image* an.«

»Wenn entweder die Brücke kaputt ist, oder es regnet und Loan keinen Schirm besitzt, dann zeige die Grafik *Loan_sad.image* an.«

»Wenn es geregnet hat und Loan keinen Regenschirm hatte, und seitdem die Sonne nicht mehr geschienen hat, dann zeige die Grafik *Loan_wet.image* an. «

Die Resultate gestalten sich ähnlich wie auf dem folgenden Bild zu sehen.

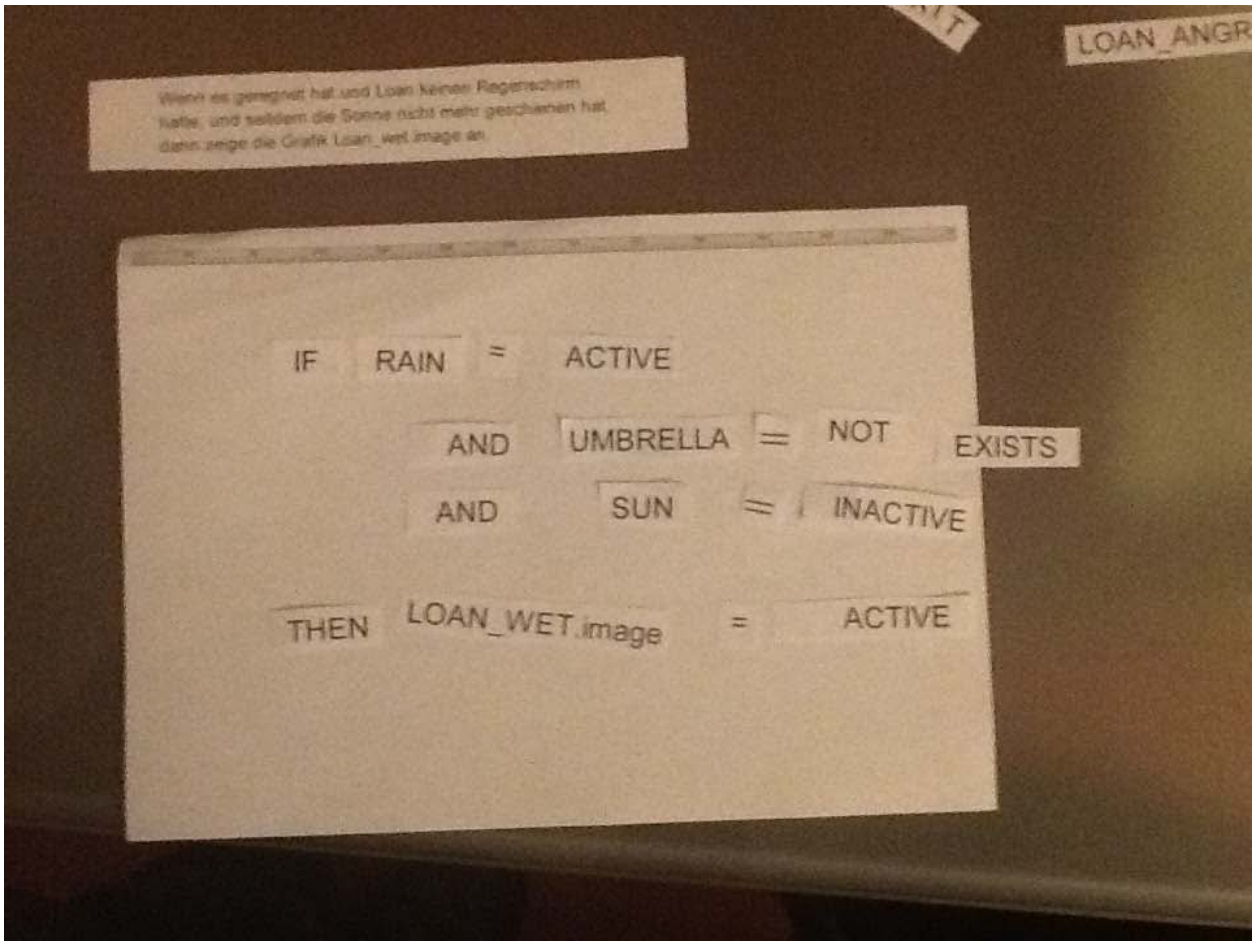


Abbildung 22: Beispielhafter Einsatz des Prototypen im Nutzertest

Der Test wurde in der Art-Abteilung von ➳Happy Tuesday mit insgesamt fünf Personen durchgeführt. Die Nutzer haben keine Probleme, einfache logische Verknüpfung mit dem gegebenen Material auszudrücken. Die „entweder-oder“ Aussage wurde durchgehen übergangen und als „oder“ dargestellt. Am beliebtesten ist der Aufbau als Satzstruktur in Verbindung mit „Active“ und „Inactive“ für die Statusabfragen.

Daraus folgend wird der *Conditioneffect* die logischen Operationen in Satzstrukturen anbieten. Die Operatoren werden dabei aus ›Dropdown-Menüs‹ ausgewählt. Die auszuführenden Effekte werden über ›Drag&Drop‹ mit einer Linie mit dem *Conditioneffekt* verbunden.
Der zeitliche Ablauf in der letzten Aussage bereitet allen Nutzern Probleme. Trotz bereitgestellter ›Timeline‹ nutzt nur ein Tester die Möglichkeit die Zeit über den Raum von links nach rechts darzustellen. Diese Vorgehensweiser wird in EFI zwingend notwendig. Der Punkt wird deswegen ausführlich erklärt werden müssen.

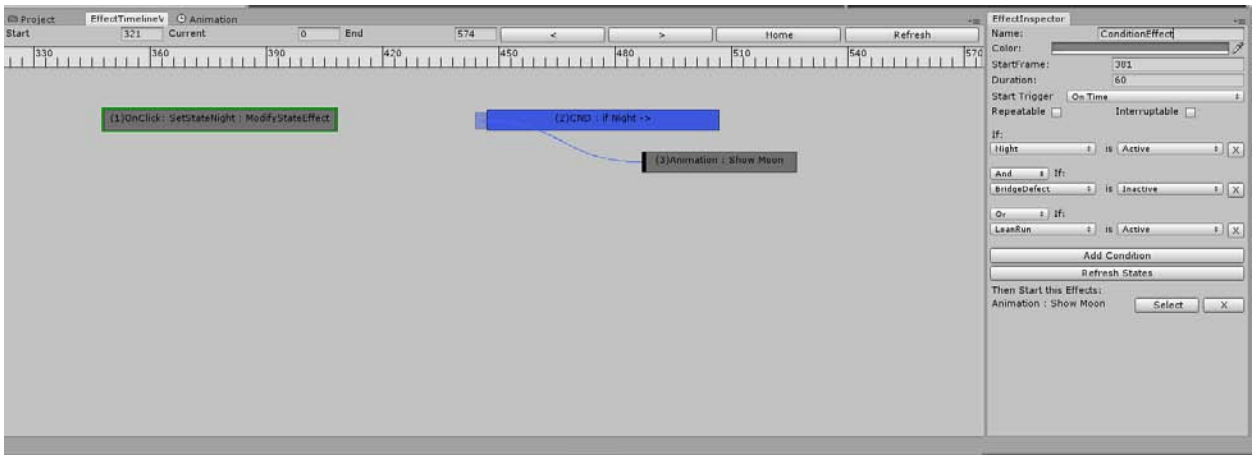


Abbildung 23: Screenshot des *ConditionEffects* in EFI nach der dritten Iteration

Alle Effekte erhalten in dieser Iteration den schon besprochenen ›Trigger‹. Zur Auswahl stehen:

- OnTime
- OnClick
- OnSwipe (Left & Right)
- OnExternal

Die Start Logik der Effekte ändert sich dahingehend, dass ein Effekt wenn sein Startzeitpunkt erreicht ist aktiv wird. Das bedeutet sein ›Trigger‹ kann ausgelöst werden. Die Funktion des Effektes wird aber erst ausgeführt, wenn zusätzlich der ›Trigger‹ ausgelöst wurde. Bei *OnTime* wird der ›Trigger‹ direkt ausgelöst. Bei *OnClick* und *OnSwipe* sind die entsprechenden Nutzereingaben der Auslöser. *OnExternal* wird durch andere Effekte gestartet. Nach diesen Änderungen ist die Iteration abgeschlossen, und EFI bereit zum nächsten Nutzertest.

„Funktioniert das auch?“ – Der Nutzertest

Die Fertigstellung des Prototypen stellt den Abschluss des ersten Entwicklungszyklus dar. Der Release dieses Zyklus passiert in Form von Nutzertests. Um die Ergebnisse bewerten zu können, wird ein definierter Testfall benötigt. Dieser stellt sicher, dass alle Features eingesetzt und verstanden werden. Die Kriterien für den Testfalls setzen sich aus mehren Teilen zusammen. Der erste Teil lässt sich aus der erarbeiteten Definition für interaktive Software

ableiten. Der zweite Teil ergibt sich aus dem Featureset des Prototypen. Der Testfall wird als Aufgabe an die Nutzer im folgenden Test verwendet.

Beschreibung des Testfalls

Für den Testfall wurden von ➳Stenarts Grafiken zur Verfügung gestellt. Der vollständige Ablauf ist auf folgendem ›Storyboard‹ zu sehen:

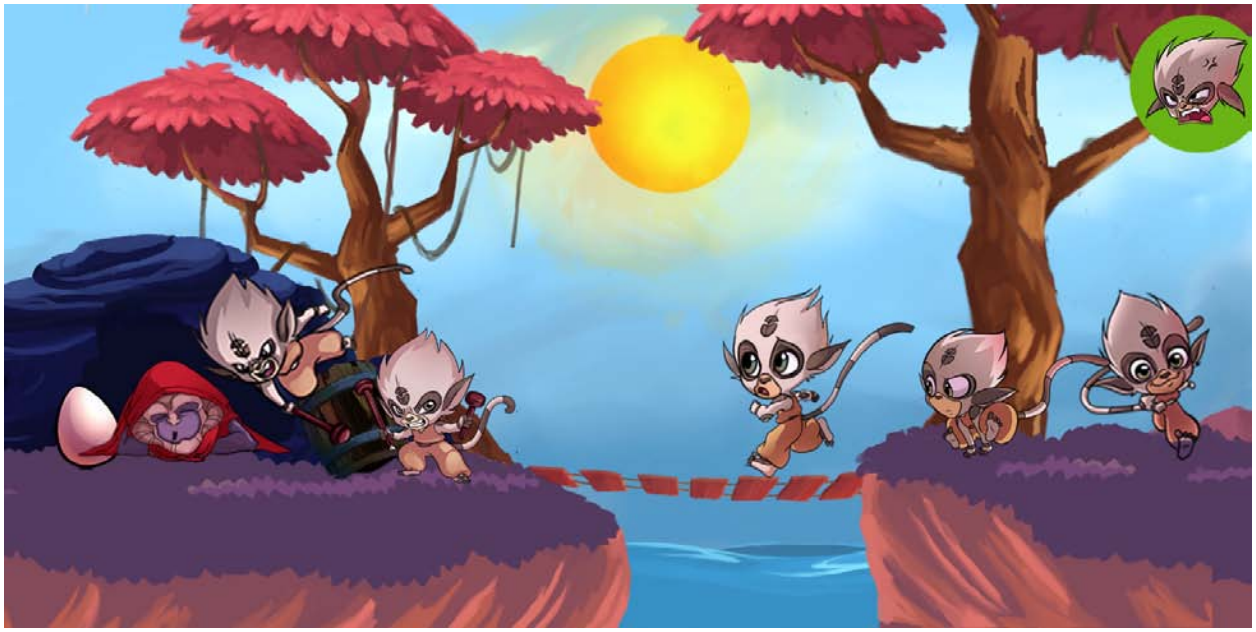


Abbildung 24 : Storyboard Testfall 01

Der Protagonist *Loan* (auf der rechten Seite startend, mehrfach im Bild) möchte an das Ei auf der linken Seite des Bildes herankommen. Allerdings schläft sein Gegenspieler *Shaden* in der Höhle vor dem Ei.

Abbildung 25: Storyboard Testfall 02

Das Aufwecken des Gegenspielers sorgt dafür, das *Loan* vertrieben wird.

Abbildung 26: Storyboard Testfall 03

Durch ›Klick‹ auf die Sonne bzw. den Mond lässt sich die Tageszeit zwischen Tag und Nacht umschalten.

Abbildung 27: Storyboard Testfall 04

Nachts wacht allerdings ein kleiner Kraken über die Brücke, so dass *Loan* diese nicht überqueren kann.

Abbildung 28: Storyboard Testfall 05

Durch ›Klick‹ auf das Fass, rechts neben der Höhle, kann der Nutzer nachts den Geist aus dem Fass aufscheuchen. Der *Shaden* fürchtet den Geist und flieht. Am Tag ist jetzt der Weg zum Ei für *Loan* frei.
Um bei der Steuerung zwischen den Bewegungen von *Loan* (Laufen über die Brücke) und den Aktionen des Nutzers (Tageszeit wechseln, Geist befreien) zu unterscheiden gibt es in der

oberen rechten Ecke ein kleines Interfacesymbol. Mit ›Klick‹ auf den Kopf wechselt man zwischen *Loans* Aktionen (grünes Icon) und Benutzeraktionen (graues Icon).
Wenn *Loan* aktiv ist, bewirkt ein Klick auf eine Stelle des Bildschirms, dass die Grafik von *Loan* sich animiert zu diesem Punkt bewegt. Die Aktionen des Nutzers sollen mit Audio-Feedback versehen werden.

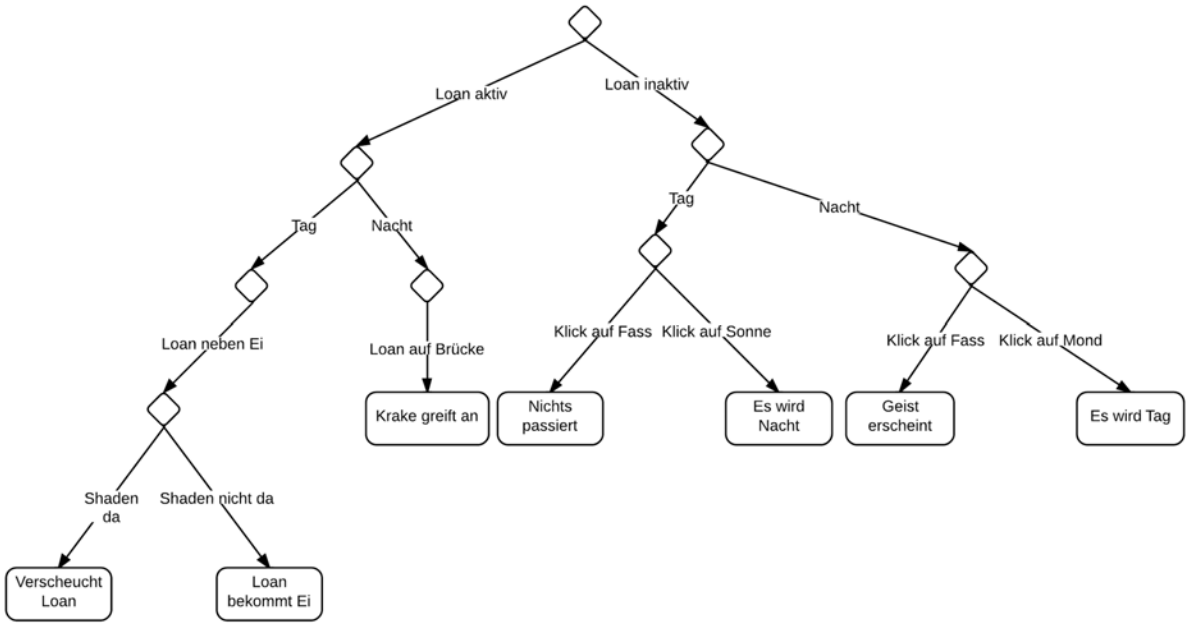


Abbildung 29: Verzweigung der Interaktionspunkte im Tesfall

Der Anwendungsfall bietet verzweigte Interaktionsbäume. Damit erfüllt er Stufe 3 der Interaktivitätsskala. Die Interaktionen:

1. Krake nicht zu sehen bei Tag
2. Krake erscheint bei Nacht und greift Charakter an

führen dazu das der Nutzer die Brücke bei Tag überquert. Dies ist eine minimal Rückwirkung auf den Nutzer, um die Möglichkeiten für Stufe 4 aufzuzeigen.
Die Frequenz und Bandbreite des Testfalles sind bewusst klein, um den Zeitaufwand gering zu halten.
Die aufgelisteten Kernfeatures werden durch den Testfall abgedeckt:

- Darstellung von Text und Bild
  ⇒ alle Grafiken in der Szene
- Wiedergabe von Ton
  ⇒ Audio-Feedback bei Nutzerinteraktion
- Erstellen und Wiedergabe von Animation
  ⇒ animierte Bewegung von *Loan*
- Speichern und Abfragen von Zuständen
  ⇒ Tag und Nacht

    ⇒ *Shaden* vor der Höhle / geflohen
- Ausführen von Aktionen in Abhängigkeit von Zeit oder Nutzerinput
    ⇒ Zeit: Bewegung von Hintergrundelementen
    ⇒ Alle anderen Aktionen
- Ausführung genannter Aktionen in Abhängigkeit von logischen Operationen
- Annehmen von Benutzer Eingaben

Nutzertest: Samuel Stephan – EFI

Wie schon bei den Softwaretests ist der Zeitrahmen für den Test auf 60 Minuten beschränkt. Zielsetzung ist die Erarbeitung der im Testfall beschriebenen Anwendung.

Wie bereits am Papierprototypen evaluiert, gelingt es Samuel nicht ohne Erklärung die Bedingung in zeitlicher Abhängigkeit darzustellen. Nach einer kurzen Einweisung kann er die einfachen Aufgabestellungen, wie das Erscheinen der Krake bei Nacht, erstellen.
Der vermutete Schwachpunkt in der Darstellung ›extern ausgelöster Effekte‹ zeigt sich im Verlauf der Nutzertests ausführlich. So ist Samuel nicht immer klar, welcher Effekt zu welchem Zeitpunkt aktiv ist.
Zusätzlich weist Samuel auf die ersten vermissten Features hin: Animationen starten und enden immer an festen Punkten. Soll der Charakter also von A nach B laufen, und anschließend zu C, so muss für beide Strecken eine eigene Animation erstellt werden. Selbst dann, wenn die Bewegung die gleiche ist. Gefordert ist eine Möglichkeit Animationen in Abhängigkeit von relativen Punkten zu erstellen.
Mit dem *ModifyTimelineEffekt* kann die Hauptzeitleiste angehalten werden um zum Beispiel auf Nutzereingaben zu warten. In der momentanen Architektur werden alle Effekte durch die gleiche Zeitleiste synchronisiert. Deswegen stoppen im beschriebenen Fall auch alle Hintergrundanimationen wie zum Beispiel das Wasser. Eine ungewollte Eigenschaft, da die Anwendung dadurch „abgestürzt“ aussieht.
Bei der Bedienung des Editors fällt auf, das Samuel öfter die gleichen Fehler unterlaufen. Häufig handelt es sich dabei um vergessene Verbindungen zwischen den Elementen, oder nicht eingestellte Werte.
Der Testfall konnte aufgrund der genannten Probleme nicht ausreichend bearbeitet werden.

Als Ergebnis des Nutzertests werden in der nächsten Iteration folgende Änderungen vorgenommen:

1. Um ›extern gestartete Effekte‹ besser darstellen zu können, wird die Idee der übergeordneten ›Nodes‹ aus dem Behaviour-Tree übernommen und auf die ›Timeline‹ übertragen. Konkret bedeutet das: Die ›Timeline‹ wird in ›View‹ und ›Controler‹ unterteilt und als Effekt implementiert. Jeder Effekt ist Kind eines *Timeline-Effekt*. Als oberste Ebene bleibt eine *Maintimeline* bestehen, die weiterhin von überall erreichbar ist. Die *Timeline-Effekte* können wie normale Effekte gestartet und gestoppt werden. Wenn ein *Timeline-Effekt* aktiv ist, gibt er auf *OnNextFrame* das ›Update‹ an seine Kinder weiter.

Dadurch wird gleichzeitig auch das Problem der stoppenden Elemente behoben, da diese in eine eigene ›Timeline‹ ausgelagert werden können.

2. Es wird ein *Tween-Effekt* hinzugefügt. Bei den bisherigen Animationen sind Start und Endpunkt in der Animation gespeichert. Alle Translationen in der Animation sind absolut. Bei einem ›Tween‹ wird nur die Animationskurve gespeichert. Dadurch kann mit der gleichen Animation zwischen beliebigen Start und Endpunkten interpoliert werden.
3. Der *Editor-Inspektor* in dem die Werte der Effekte eingestellt werden bekommt eine Fehlerüberprüfung und entsprechende Warnmeldung für:
    a. fehlende Verbindungen
    b. fehlende oder falsche Einstellungen
4. Der *ModifyStateEffect* wird in *SetConditionEffect* umbenannt, sowie der *ConditionEffect* in *QueryConditionEffect.* Dies geschieht um die Zusammengehörigkeit und Reihenfolge hervorzuheben.

Die umstrukturierte Architektur sieht dann wie folgt aus:

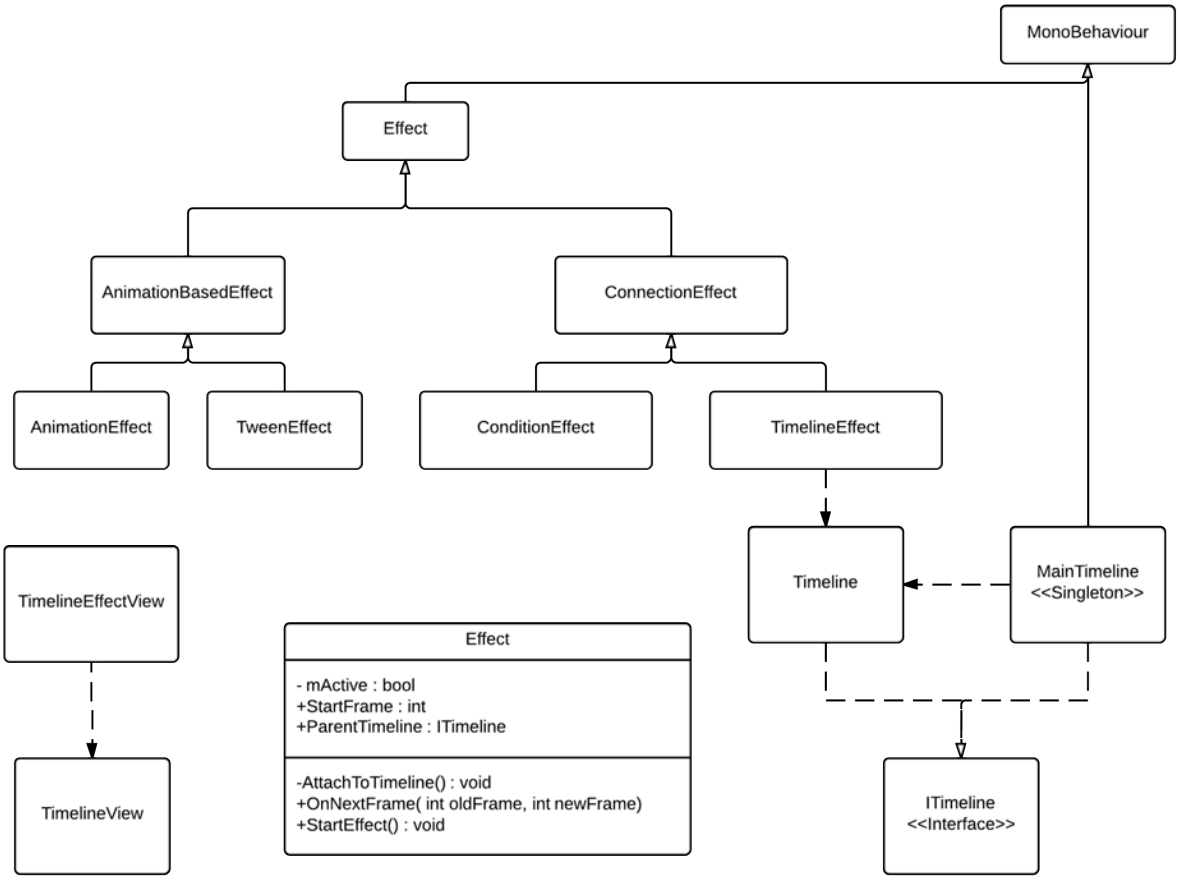


Abbildung 30 : Erweiterte Systemstruktur am Beginn der vierten Iteration

Für die animationsbasierten Effekte: ›Tween‹ und ›Animation‹ werden sämtliche Zeitberechnungsroutinen in eine Basisklasse ausgelagert. Der neue *Timeline-Effect* wird auf die

gleiche Art und Weise mit seinen Kinder verbunden wie der *Condition-Effect.* Deswegen wird auch hier eine neue Basisklasse ausgegliedert. Alle Effekte erhalten das neue Attribut *ParentTimeline*. Von dieser erhalten sie in Zukunft ihre ›Updates‹. Die Timeline wird als Interface abstrahiert, um einheitlichen Zugriff auf den *Timeline-Effect* und die *MainTimeline* zuzusichern.

Im Editor sieht die ›Timeline‹ wie folgt aus:

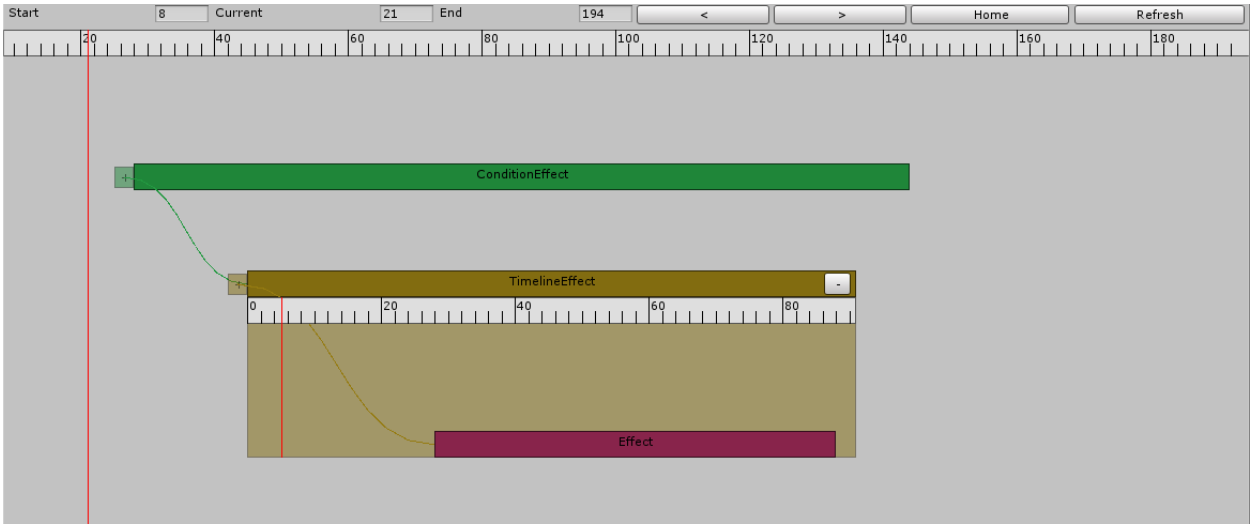


Abbildung 31: Timeline-Effekt in EFI nach der vierten Iteration

Der *Timeline-Effect* erhält eine eigene Anzeige für den aktuellen ›Frame‹. Zusätzlich kann er zu einem „normalen“ Effekt zusammengeklappt werden, und bietet somit auch eine Gruppierungsfunktion.

Die neuen Warnmitteilungen sind auf dem nächsten Screenshot zu sehen:

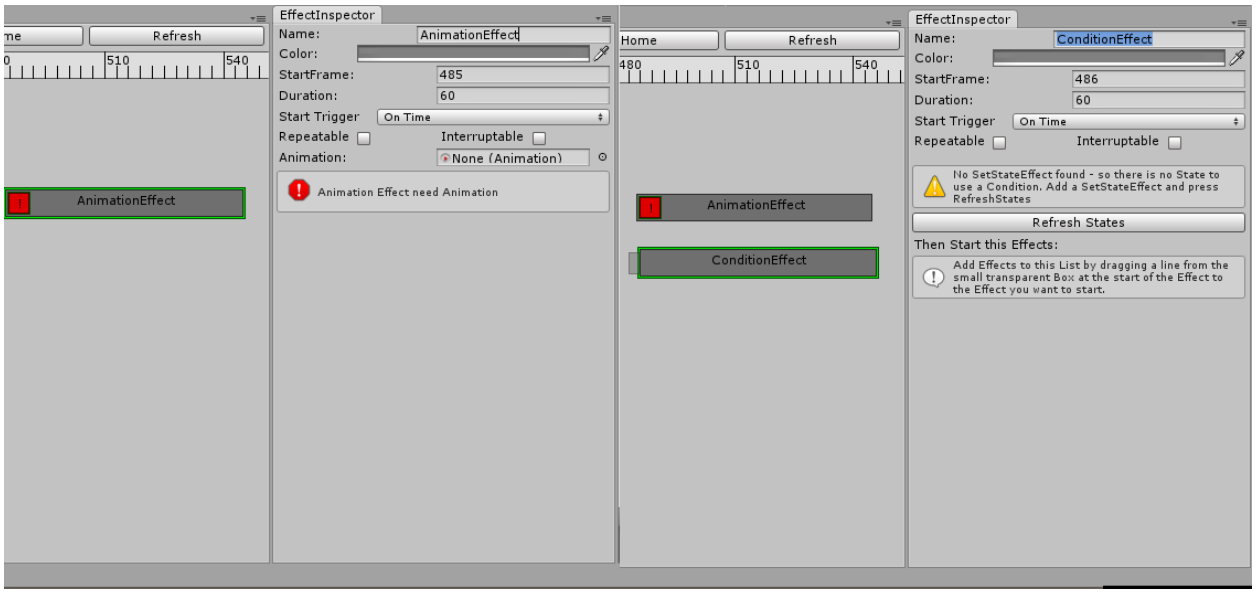


Abbildung 32: Warnmitteilungen in EFI nach der vierten Iteration

Nutzertest: Pauline Altmann – EFI

Als Resultat aus den vorangegangenen Nutzertests wird in der Softwareerklärung in diesem Test speziell auf den Aufbau und die Abfrage von Bedingungen eingegangen. Zusätzlich wird das Konzept des neuerstellten *Timeline-Effektes* erklärt. Die Bedingungen können darauf hin von Pauline nach einigem ausprobieren problemlos erstellt werden.
Um Effekte durch eine Bedingung zu starten, muss der ›Trigger‹ des Effektes auf *OnExternal* gestellt werden. Dabei fällt auf das die ›Trigger‹-Bezeichung *OnExternal* schwierig zu verstehen ist. Besser währe hier ein Bezeichner der den Zusammenhang deutlich macht, wie zum Beispiel *FromCondition*.
Alle zeitabhängigen Animation sortiert Pauline in eigene Zeitleisten mithilfe des *Timeline-Effekt*. Die angelegten Bedingungen, wie zum Beispiel das ändern der Tageszeit, verbleiben auf der *Maintimeline*. Diese Effekte sind jedoch inhaltlich unabhängig von der Zeit. Pauline findet die *Maintimeline* daher irritierend. Unterstützt wird dieser negative Eindruck durch die Darstellung der Effekte. Diese werden auf der *Maintimeline* weiterhin mit einem Startzeitpunkt und einer Dauer als Box angezeigt.
Obwohl Pauline in der aktuellen Variante des Prototypen auf keine großen Verständnisprobleme mehr stößt, ist es nicht möglich in der beschränkten Zeit den ganzen Testfall umzusetzen.
Die umgesetzten Features sind:

- Tag und Nacht Wechsel
- Bewegung von *Loan* über die Brücke bei Tag
- Angriff der Krake bei Nacht

Diese Umsetzung zeigt das interaktive Inhalte der Stufe 3, mit dem momentanen Prototypen ohne Programmierkenntnisse erstellt werden können.

Im Rahmen dieser Arbeit wird für den Prototyp keine weitere Iteration ausgeführt werden. Der letzte Nutzertest legt allerdings nahe, die Darstellung von zeitbasierten Effekten und Aktionsketten, die der Flowgraph-Architektur entstammen, zukünftig visuell stärker voneinander zu trennen. Denkbar wäre hierbei zum Beispiel ein Verzicht auf die *Maintimeline* und eine Darstellung der durchgehend gültigen Effekte als Kreise. Die Bezeichnung des *OnExternal* ›Triggers‹ zeigt, dass trotz starkem Fokus auf Namen aus der Anwenderdomäne, immer wieder unklare Begriffe auftauchen. Diese Unklarheiten können nur durch kontinuierliche Nutzertests aufgedeckt werden. (↘07_Resultat_Nutzertest_Pauline/) und (↘07_Resultat_Nutzertest_Pauline.mp4)

„Wie geht es weiter?“ – Das Fazit

In der Vision, die die Grundlage für diese Arbeit bietet, wurde eine Software beschrieben, mit der eine direkte Umsetzung von einer Idee zum Produkt möglich ist. Eine Software: »[...] mit deren Hilfe interaktive Inhalte ohne Programmierkenntnisse erstellt werden können«.
Auf Basis von verschiedenen Interaktionsdefinitionen, sowie einigen Anwendungsbeispielen, wurde das Ergebnis dieser Software im Rahmen der Arbeit definiert. Nachfolgend konnten mithilfe von Interviews und Softwaretests die gewünschten Kernfunktionen herausgearbeitet werden. Verschiedene Bedienkonzepte, wie die Darstellung über zustandsbasierte Automaten, oder zeitbasierte Effekte, wurden beschrieben und zusammen mit den erarbeiteten Funktionen in den Nutzertests auf praxistauglichkeit überprüft. Wie die abschließenden Nutzertest gezeigt haben, ist jedoch auch der entwickelte Prototyp noch weit von einem Werkzeug entfernt, das »Kreativen die Möglichkeit gibt, schnell und effizient Ihre Inhalte umzusetzen«.
Die momentan übersichtliche Menge der Funktionen wird erwartungsgemäß starken Zuwachs bekommen, sobald das Bedienkonzept verfeinert wurde, und der Nutzer die grundlegende Bedienung verinnerlicht hat. EFI verlangt, wie alle spezialsierten Softwareprogramme aus dieser Domäne, von den Nutzern einen größeren Lernaufwand. Dieses Lernverhalten konnte in den, auf 60 Minuten begrenzten, Nutzertests nicht abgebildet werden. Die Beurteilung kann erst durch den Einsatz in einem „realen“ Projekt abgegeben werden.

Der entwickelte Prototyp stellt eine Basis dar, um verschiedenen Konzepte nutzernah zu überprüfen. Der Einbau des *Timeline-Effekt* hat gezeigt, dass die zu Grunde liegende Softwarearchitektur robust genug ist verschiedene Umstellungen innerhalb des diskutierten Rahmens möglich zu machen. Die kontinuierliche Einarbeitung von Nutzerfeedback hat es ermöglicht schon während der ersten Iterationen einen benutzbaren Prototypen zu erstellen. Diese Entwicklungsmethode wird bei der Weiterentwicklung beibehalten werden, um EFI auch weiterhin effizient an die sich ändernden Bedürfnisse anzupassen.
Eine kurze Demonstration des Prototypen bietet das Video: (↘08_Beispiel_Workflow_EFI.mp4).
Benötigte Installationsdateien für den Prototypen, sowie eine Installationsanleitung befinden sich im Ordner: (↘Unity3d_Installation/).

Die Frage „Gibt es das nicht schon?“, bleibt auch nach der Entwicklung des Prototypen weiterhin mit „Nein, leider noch nicht!“ zu beantworten. Aber ich hoffe das EFI einen Beitrag dazu leistet, dass wir in weiteren zehn Jahren vielleicht auch die letzte, noch offene Vision aus dem Jahre 1993 als Alltäglich betrachten können, indem wir uns am Feierabend, unsere ganz persönliche Fassung von *„Terminator XXV“* anschauen können.

Danksagung

Für die Möglicheit der Kooperation, die die ausführlichen Nutzertests im Rahmen der Arbeit erst ermöglicht haben, bedanke ich mich bei Karsten Härle (Happy Tuesday) und Thorsten Kiecker (Stenarts Berlin). Vielen Dank an die Tester: Pauline Altmann, Kevin Blank, Tino Englert, Samuel Stephan. Sowie Nora Möllers, Karsten Härle, Alexander Surma und Pauline Altmann für die Fehlerkorrektur der Arbeit. Besonderer Dank gilt Prof. Dr. Lutz Prechelt und Julia Schenk für Betreuung der Arbeit.