Hochschule für Angewandte
Wissenschaften Hamburg
Hamburg University of Applied Sciences

Moderne Präsentationen:

Erarbeitung eines Präsentationsprogramms unter Verwendung aktueller Browser-Technologien

Bachelorthesis

HAW Hamburg
Fakultät Design, Medien und Information
Studiendepartment Medientechnik
Studiengang Media Systems

Felix Janus
2116462

Erstprüfer: Prof. Dr. Edmund Weitz
Zweitprüfer: Prof. Ralf Hebecker

Abgabe: 7. Dezember 2015

Felix Janus

Thema der Arbeit

Moderne Präsentationen: Erarbeitung eines Präsentationsprogramms unter Verwendung aktueller Browser-Technologien

Stichworte

Präsentation, Webanwendung, JavaScript, Webtechnologien, HTML5, CSS3, Three.js, WebGL

Kurzzusammenfassung

Ziel dieser Bachelorarbeit ist es, ein Programm zu entwickeln, das dazu genutzt werden kann, Präsentationen zu erstellen. Dabei wird der Stand derzeitiger Präsentationsprogramme analysiert und besonders auf das Bedienparadigma zur Erstellung einer Präsentation untersucht. Aufbauend auf dieser Grundlage wird ein eigenes Programm als Webanwendung programmiert und eine passende Bedienoberfläche entwickelt.

Felix Janus

Title of the paper

Modern Presentations: Development of a presentation program using current browser technologies

Keywords

Presentations, Web application, JavaScript, Web technologies, HTML5, CSS3, Three.js, WebGL

Abstract

The objective of this bachelor thesis is to develop a program, which can be used to create presentations. Therefor current presentation programs will be analysed, especially the user interface paradigm which is used to create a presentation. Based on these insights, a new program and user interface paradigm will be developed as a web application.

INHALTSVERZEICHNIS

1 Einleitung 1

2 Analyse 2

2.1 Analyse bestehender Präsentationsprogramme 2

2.1.1 Anforderungen an ein Präsentationsprogramm 2

2.1.2 Analyse des Bedienparadigmas zur Erstellung einer Präsentation 4

2.1.3 Analyse der grafischen Benutzeroberfläche 6

2.2 Ideen für ein neues Präsentationsprogramm 8

2.2.1 Konzept der Bedienung 8

2.2.2 Zielgruppe 10

3 Planung der Umsetzung 11

3.1 Verwendete Technologien 11

3.2 Verwendete JavaScript-Bibliotheken 12

3.2.1 Three.js 12

3.2.2 JSZip 13

3.2.3 FileSaver.js 13

3.2.4 Spectrum.js 13

3.2.5 Easing.js 13

3.2.6 FontAwesome 14

3.2.7 Carota.js 14

3.2.8 Material Design Lite 15

4 Umsetzung der Benutzeroberfläche 16

4.1 Bedienkonzept der Benutzeroberfläche 16

4.1.1 Präsentationsbereich 16

4.1.2 Hauptmenü 17

4.1.3 Zeitleiste 18

4.1.4 Aufgabenmenü 19

4.1.5 Zusammenspiel der einzelnen Bereiche 19
4.2 Darstellung der Präsentation 20
4.3 Präsentationsobjekte 21
4.3.1 Texte und Schriftarten 21
4.3.2 Symbole 22
4.3.3 Geometrien 22
4.4 Prinzip der Speicherfunktion 23
5 Umsetzung der Programmierung 24
5.1 Framework und 3D-Engine 25
5.2 Aufgaben-Typen 30
5.3 Animationen 36
5.4 Easing von Animationen 41
5.5 TaskObjects 42
5.6 Speicherfunktion 43
6 Erweiterbarkeit 45
6.1 Weitere allgemeine Funktionen 45
6.2 Multi-User und Second Screen 45
6.3 Interaktive Elemente 46
6.4 Gemeinsames Arbeiten 46
7 Fazit 48
Abbildungsverzeichnis 50
Literaturverzeichnis 52
Versicherung über Selbstständigkeit 53

1 EINLEITUNG

Bekannte Programme zum Erstellen von Präsentationen wie Microsoft Power Point[1], Keynote für Mac[2] oder Libre Office[3] unterscheiden sich in ihrer Funktionsweise durch verschiedenen Versionen hindurch nur geringfügig. Die Bedienparadigmen ähneln sich ebenso oft wie die produzierten Ergebnisse. Diese Arbeit soll sich mit den gebräuchlichen Schritten zur Erstellung einer typischen Präsentation auseinandersetzen. Aufbauend darauf soll ein eigenes zeitgemäßes Präsentationsprogramm entworfen und programmiert werden, welches ein neues Bedienkonzept zur Erstellung von Präsentationen vorstellt.

Dieses neue Programm soll dabei nicht nur klassische, einfache Präsentationen ermöglichen, sondern auch aufwendige Szenen und Objekte darstellen können. Nützlich soll dies vor allem aus Sicht eines Medieninformatikers sein, um technische Vorgänge, 3D-Szenen und Game-Elemente einfach veranschaulichen zu können. Die Idee entspringt dabei der von Steven Wittens[4] programmierten JavaScript Bibliothek MathBox.js[5], die es ermöglicht mathematische Vorgänge in einer dreidimensionalen Umgebung darzustellen. Wittens selbst nutzte diese Bibliothek auf einigen Konferenzen[6], um technische Vorgänge der Informatik, wie z.B. Antialiasing, visuell zu erklären. Dies gab den Anstoß zur Idee, ein reduziertes, benutzerfreundliches Programm zu erstellen, welches eine simple Benutzeroberfläche bietet. Die angesprochene Zielgruppe sind dabei trotzdem Benutzer mit Programmierfähigkeiten.

Die Umsetzung des Programms teilt sich in die zwei Bereiche, der Logik des Programms und der Benutzeroberfläche auf. Die Logik des Programms sorgt dabei für den programmtechnischen Ablauf, der nötig ist, um eine Präsentation zu erstellen und später anzuzeigen. Dazu zählen unter anderem das Erzeugen von Texten, Grafiken, das Berechnen von Animationen und die Speicherung der erstellten Präsentation.

Außerdem soll eine Benutzeroberfläche entworfen werden, die es auch Benutzern ohne Programmierfähigkeiten ermöglichen soll, simple Präsentationen zu erstellen.

---

[1] https://products.office.com/de-de/powerpoint | 5.12.2015
[2] https://www.apple.com/de/mac/keynote/ | 5.12.2015
[3] https://de.libreoffice.org | 5.12.2015
[4] http://acko.net | 5.12.2015
[5] https://github.com/unconed/MathBox.js/ | 5.12.2015
[6] https://www.youtube.com/watch?v=GNO_CYUjMK8 (JSConfUS 2013) | 5.12.2015

2 ANALYSE

Im Folgenden werden einige Elemente heutiger Präsentationsprogramme exemplarisch am Programm PowerPoint 2013 von Microsoft vorgestellt. Die hier beschriebenen Elemente machen nur einen geringen Teil des Umfangs von PowerPoint aus. Die Auswahl an Elementen wurde so getroffen, dass später beschrieben werden kann, wie die Elemente in dem neu entworfenen Präsentationsprogramm auf eigene Art reproduziert werden.

Im zweiten Teil dieses Kapitels werden die Ideen und das Konzept für das selbsterstellte Präsentationsprogramm näher beschrieben und dabei mit denen klassischer Präsentationsprogramme verglichen. Dabei wird beschrieben was die dadurch entstehenden Vorteile sind.

2.1 Analyse bestehender Präsentationsprogramme

2.1.1 Anforderungen an ein Präsentationsprogramm

Die grundlegende Aufgabe eines Präsentationsprogramms ist es, einem Redner die Möglichkeit zu geben, sein Publikum auch auf einer visuellen Ebene zu erreichen. Dabei soll der Inhalt durch Texte, Bilder und Grafiken visuell unterstützt werden. Dies ist vor allem deshalb nützlich, da so der Informationsgewinn für den Zuhörer maximiert wird. So können Zuhörer 30 Prozent mehr an Informationen behalten, wenn sie Informationen nicht nur hören, sondern auch sehen[7]. Dies zeigt, wie wichtig es ist, eine Präsentation so zu gestalten, dass sie Interesse beim Zuhörer weckt.

Abbildung 1 Tageslichtprojektor

Eine erstellte Präsentation wird dann auf einem Monitor oder einer Leinwand mit Beamer gezeigt, wo die Zuhörer sie verfolgen können.

Oft sind Präsentationen in Folien unterteilt, was auf der früheren Benutzung von Dia- und Tageslichtprojektoren beruht. Um mit einem Tageslichtprojektor zu präsentieren, wurden Klarsichtfolien mit Markern beschrieben, die dann mithilfe des Projektors an die Wand gestrahlt

[7] Seifert, J. (2006). Visualisieren, Präsentieren, Moderieren (23. Aufl. ed.): S. 11. Offenbach: Gabal.

wurden. Dieses Paradigma hat sich bis in die heutigen, digitalen Präsentationsprogramme gehalten.

Das Präsentationsprogramm ist für den Benutzer dabei ein Werkzeug, welches ihm mit möglichst wenig Aufwand ermöglicht, diese Folien zu erstellen und zu gestallten. Oft gibt es dabei einen vorgefertigten Satz an Bearbeitungsmöglichkeiten, die dem Benutzer eine Grundlage schaffen, seinen Vortrag visuell umzusetzen. Dies sind z.B. Textfelder, Diagramme, Geometrien oder auch Clip-Arts. Häufig werden diese Bestandteile einer Präsentation in vorgefertigten Layouts angeboten, in die ein Benutzer lediglich seine Daten und Texte eintragen muss. Darüber hinaus hat der Benutzer einige Möglichkeiten, die Gestaltung der Präsentation zu beeinflussen, indem z.B. Schriftarten, Farben und Foliendesign geändert werden können. PowerPoint bietet zudem vorgefertigte Farbpaletten und Foliendesigns, die es dem Benutzer einfacher machen sollen, eine gut abgestimmte Präsentation zu erstellen.

Während diese Form der Präsentation heute den Standard bildet, gibt es auch einige Ausnahmen wie z.B. das Präsentationstool Prezi[8]. Dieses setzt nicht auf eine Folienaufteilung, sondern stellt die komplette Präsentation als Diagramm dar, was mehr einem Präsentationsplakat gleichkommt, auf dem alle Informationen der Präsentation als Schaubild dargestellt werden. Das von Prezi erstellte Schaubild wird dann präsentiert, indem einzelne Elemente herangezoomt werden, die die vorgetragene Information enthalten.

Abbildung 2 Logo der Software Prezi

Für eine gute Präsentation kann es nützlich sein, sich Notizen zu machen, die den Redner unterstützen. Moderne Programme bieten hierfür eine Notizansicht an. Diese ermöglicht dem Präsentierenden, Notizen zu seinen Folien zu machen, die ihm wie auf einer Art Teleprompter, auf seinem Bildschirm gezeigt werden. Für die Zuhörer sind diese Notizen während der Präsentation nicht zu sehen.

[8] https://prezi.com | 5.12.2015

2.1.2 Analyse des Bedienparadigmas zur Erstellung einer Präsentation

Im Folgenden wird der grobe Ablauf beschrieben, der nötig ist, um eine Präsentation zu erstellen. Dabei wird sich auf die Erstellung einer Präsentation in Microsoft PowerPoint 2013 bezogen, da der dort zu findende Ablauf ähnlich zu anderen populären Programmen ist.

Um die Erstellung einer Präsentation möglichst einfach und übersichtlich zu gestalten, werden einzelne Abschnitte des Vortrags in Folien unterteilt. Diese Folien bestehen meist aus einer Überschrift und einem Textkörper, welche sich formatieren lassen. PowerPoint bietet zur Vereinfachung der Formatierung und Gestaltung bereits eine kleine Auswahl an Designvorschlägen an.

Zusätzlich kann der Benutzer Grafiken, Bilder o.ä. in die Folie einbinden, um diese visuell zu gestallten. Hier stellt PowerPoint eine Sammlung aus verschiedenen Geometrien und Diagrammen bereit, die dabei helfen sollen Informationen interessant zu gestallten.

Abbildung 3 PowerPoint - Auswahl an Geometrien

Um die verschiedenen Elemente möglichst einheitlich und passend auf den Folien zu platzieren, kann jede Folie ihr eigenes Layout und Aussehen haben. PowerPoint bietet dafür eine Auswahl an vorhandenen Layouts an, die vom Benutzer ausgewählt und angepasst werden können.

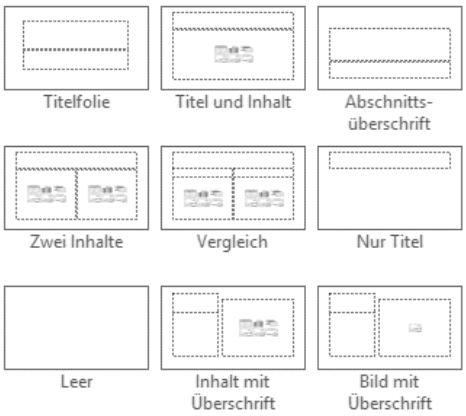

Abbildung 4 PowerPoint - Layout Auswahlmenü

Die einzelnen Elemente auf einer Folie lassen sich außerdem mit Animationen versehen. PowerPoint unterteilt diese Animationen dabei in vier verschiedene Kategorien: Eingangseffekte, Ausgangseffekte, Hervorhebungseffekte und Animationspfade. Diese Effekte ermöglichen es z.B., Texte oder Ähnliches nacheinander anzuzeigen, ohne die Folie zu wechseln. Dies ist vor allem dann sehr nützlich, wenn die Aufmerksamkeit der Zuhörer auf ein bestimmtes Element gelenkt werden soll.

Abbildung 5 PowerPoint - Verschiedene Animationstypen

Nachdem der Benutzer alle Folien erstellt, sie mit dem gewünschten Inhalt gefüllt und formatiert hat, kann er über die Sortierung der Folien festlegen, in welcher Reihenfolge diese später bei der Präsentation gezeigt werden. Während der eigentlichen Präsentation kann der Benutzer dann mithilfe der Tastatur oder der Maus durch die Präsentation navigieren.

2.1.3 Analyse der grafischen Benutzeroberfläche

Die Benutzeroberfläche lässt sich grob in drei Bereiche unterteilen: die Folienübersicht, die aktuell bearbeitete Folie und die Menüleiste.

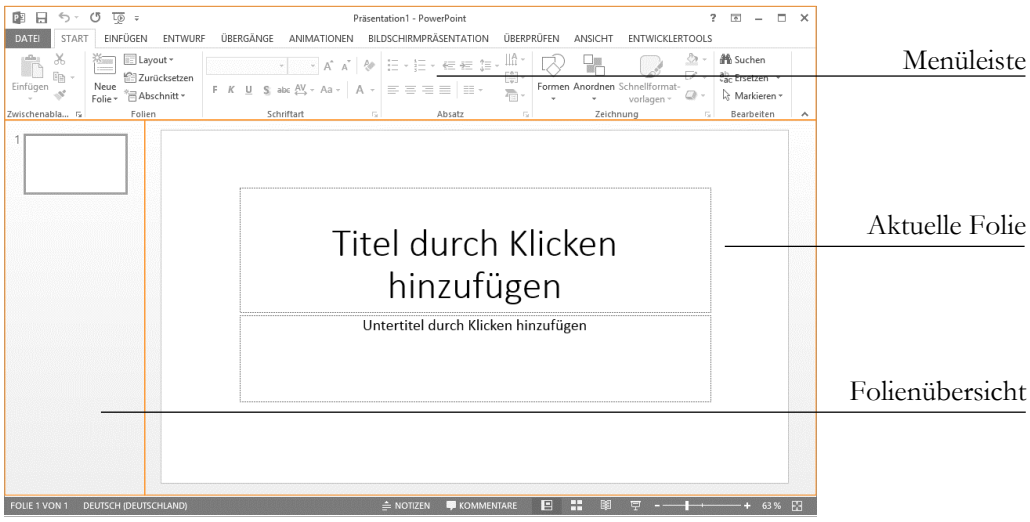

Abbildung 6 PowerPoint - Unterteilung des Fensters

Die Folienübersicht listet alle hinzugefügten Folien in chronologischer Reihenfolge auf und bietet eine kleine Vorschau darauf, was sich auf der jeweiligen Folie befindet. Hier kann der Benutzer auch die Reihenfolge bestimmen, indem er die Folien per „Drag&Drop“ sortiert. Bei einem Klick auf eine Folie kann diese in der Hauptansicht bearbeitet werden.

Die Menüleiste wird als für Microsoftprodukte typisches Ribbon[9] dargestellt. Beim Ribbon handelt es sich um eine von Microsoft entworfene Idee zur übersichtlichen und einheitlichen Darstellung von Menüs.

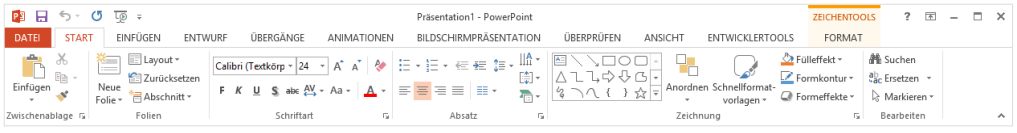

Abbildung 7 Microsoft Ribbon

Das Ribbon befindet sich im oberen Abschnitt des Fensters und nimmt die volle Breite ein. Es bildet das Hauptmenü, über das der Benutzer Objekte auf den Folien bearbeiten und Einstellungen an der Präsentation vornehmen kann.

[9] Seite „Ribbon“. In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 20. Juli 2015, 19:46 UTC. URL: https://de.wikipedia.org/w/index.php?title=Ribbon&oldid=144213820

Zur verbesserten Übersichtlichkeit ist das Ribbon in verschiedene Tabs unterteilt. Neben allgemeinen Einstellungen werden auch spezialisierte Einstellungen angezeigt, die sich auf ein ausgewähltes Element beziehen. Wenn der Benutzer beispielsweise einen Textblock auf einer Folie auswählt, wird im Ribbon-Menü der zusätzliche Eintrag „Zeichentools“ angezeigt. In diesem Untermenü finden sich dann Funktionen zur Konfiguration des Textfelds, die es z.B. ermöglichen, eine Textfarbe oder einen Rahmen auszuwählen.

Das Ribbon zeigt dabei die Elemente immer möglichst groß als Vorschau an, sodass für den Benutzer schnell ersichtlich ist, welche Änderung er vornimmt oder vornehmen kann. Diese Elemente sind in den einzelnen Tabs zusätzlich in Gruppen unterteilt. Diese Gruppierung wird durch vertikale Trennlinien zwischen den einzelnen Gruppen verdeutlicht, außerdem hat jede Gruppe ein beschreibendes Schlagwort am unteren Rand.

Da PowerPoint ein sehr umfangreiches Programm ist, ist auch das Ribbon-Menü entsprechend komplex. Oft kann es vorkommen, dass es einige Zeit dauert, bis der Benutzer einen bestimmten Menüpunkt oder eine Einstellung in den verschiedenen Tabs und Gruppen gefunden hat. Dies liegt auch an den verschiedenen Stilen, in denen die Menüpunkte gezeigt werden: einige Menüpunkte bestehen nur aus einem Symbol, andere haben zusätzlich noch einen Beschreibungstext und wieder andere werden direkt als Vorschau angezeigt.

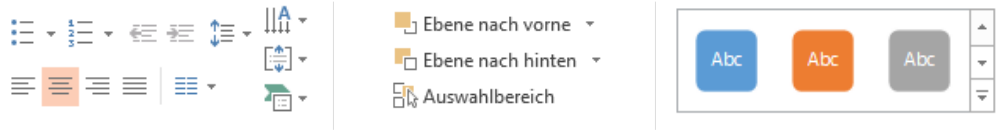

Abbildung 8 PowerPoint - Ribbon-Menüpunkte

Dieses System soll dafür sorgen, dass der Benutzer häufig gebrauchte Menüpunkte schnell wiederfinden kann. Allerdings führt es im Umkehrschluss dazu, dass einige selten gebrauchte Einstellungen schwerer zu finden sind.

2.2 Ideen für ein neues Präsentationsprogramm

2.2.1 Konzept der Bedienung

Teil dieser Arbeit ist es eine etwas andere Herangehensweise an das Bedienparadigma zur Erstellung einer Präsentation umzusetzen. Dabei sollen einige der typischen Elemente einer Präsentation überdacht und neu erarbeitet werden. Dies soll basierend auf modernen Design- und Bedienstandards geschehen. Ziel ist es dabei ein Programm zu entwickeln, das im Front-End leicht zu bedienen ist, aber im Back-End trotzdem viele Anpassungen durch den Benutzer ermöglicht.

Eine der größten Änderungen ist, dass anstatt auf flachen Folien, die komplette Darstellung der Präsentation in einer dreidimensionalen Umgebung erfolgt. Dort können Texte, Bilder und Grafiken ebenso frei angeordnet werden, wie 3D Modelle. Damit soll es dem Benutzer ermöglicht werden, auch technisch komplexe Probleme einfach visuell darzustellen.

Außerdem soll als zentraler Punkt ein Zeitstrahl eingeführt werden, auf dem der Ablauf der Präsentation dargestellt wird. Dieser Zeitstrahl soll den bisherigen Standard, Präsentationen in Folien zu unterteilen, ersetzen. Die Funktionsweise des Zeitstrahls soll dabei ähnlich sein, wie sie bereits von Software zum Filmschnitt bekannt ist. Das heißt, Elemente aus der Präsentation werden in chronologischer Reihenfolge von links nach rechts auf dem Zeitstrahl angeordnet. Um diesen Vorgang möglichst anschaulich und einfach darzustellen, wird der Zeitstrahl in einzelne Abschnitte unterteilt, die sogenannten Schritte. Während der Präsentation hat der Benutzer die Möglichkeit, zum nächsten, vorherigen oder einem bestimmten Schritt zu springen. Die Auswahl des Schrittes geschieht dabei über eine Vor- bzw. Zurückschaltfläche oder durch einen Klick auf den gewünschten Schritt.

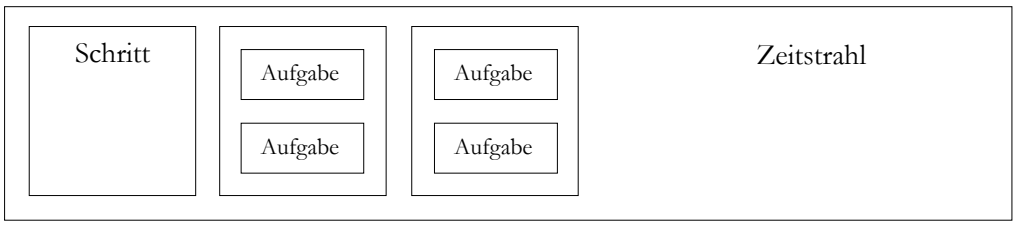

Abbildung 9 Aufbau des Zeitstrahls

Diese Schritte wiederrum werden in Aufgaben unterteilt, wobei sich jeder Schritt aus mehreren Aufgaben zusammensetzen kann, die dann gleichzeitig abgearbeitet werden. Jede dieser Aufgaben repräsentiert einen bestimmten Vorgang. Ein solcher Vorgang kann z.B. sein ein Textfeld, eine Grafik o.ä. der Präsentation hinzuzufügen, zu entfernen oder zu animieren. Dabei kann unabhängig bestimmt werden wann ein solches Element erscheint und wann es wieder verschwindet. So kann z.B. ein Element drei und ein anderes fünf Schritte lang auf der Präsentation zu sehen sein.

Des Weiteren hat jede hinzugefügte Aufgabe eigene Attribute und Eigenschaften. Bei Texten können diese Eigenschaften beispielsweise Textgröße, Textfarbe und Schriftart sein, bei Animationen können es Eigenschaften wie z.B. Dauer der Animation sein.

Diese Eigenschaften sollen in einem Menü vom Benutzer geändert werden können. Zum Anzeigen dieses Menüs muss der Nutzer dafür auf die entsprechende Aufgabe in der Zeitleiste klicken. Jeder Typ von Aufgabe hat dabei ein eigenes Menü mit den nötigen Menüpunkten. Dieser Mechanismus soll dafür sorgen, dass immer nur die passenden Eigenschaften in einem Menü angezeigt werden, um dieses möglichst übersichtlich zu halten.

Der Benutzer kann Schritte und Aufgaben beliebig auf dem Zeitstrahl anordnen und so den Ablauf seiner Präsentation koordinieren. Dieses System soll es später möglichst einfach machen, Animationen und Effekte in die Präsentation einzubauen. Außerdem werden dadurch komplexere Vorgänge, wie Animationen von 3D-Modellen, ermöglicht.

Elemente, die der Benutzer zur Präsentation hinzufügt, können direkt auf der Präsentationsfläche angeordnet werden, indem der Benutzer sie mit der Maus verschiebt. Zudem soll es die Möglichkeit geben, diese Elemente auch rotieren und skalieren zu können. Da die Darstellung in einer dreidimensionalen Umgebung erfolgt, sollen Elemente auch in der Tiefe verschiebbar sein. Dies soll für eine natürliche Verdeckung und damit zugleich auch eine Anordnung einzelner Elemente sorgen. So soll es direkt ersichtlich sein, welches Element sich im Vordergrund und welches sich im Hintergrund befindet.

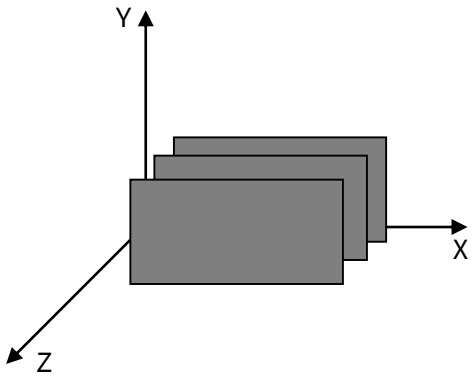

Abbildung 10 Anordnung der Elemente auf der Tiefenachse

2.2.2 Zielgruppe

Während sich der Hauptteil der Handhabung des Programms an Normalverbraucher richtet, sollen erfahrene Benutzer trotzdem die Möglichkeit haben, eigene Funktionen in die Präsentation einzubauen. Um dies zu ermöglichen, sollen Bestandteile des Programms modular aufgebaut werden. Mit Hilfe einer simplen API hat der Benutzer dann die Möglichkeit, selbst Module zu programmieren. Ein Modul kann z.B. ein 3D-Objekt sein oder auch der Ablauf einer Animation. Dieses System soll auch der einfachen Erweiterung, durch externe Bibliotheken, dienen.

Für Benutzer ohne Programmierkenntnisse soll es einige Standardmodule geben. Dies sind bereits programmierte Module, wie z.B. Textfelder, Bilder oder einfache Animationen.

3 PLANUNG DER UMSETZUNG

3.1 Verwendete Technologien

Die Umsetzung des Präsentationsprogramms teilt sich in zwei Bereiche auf: einerseits die Umsetzung der eigentlichen Darstellung der Präsentation und andererseits die Umsetzung des Interfaces. Die Darstellung der Präsentation soll in einer dreidimensionalen Umgebung erfolgen. Das heißt, im Gegensatz zu klassischen Präsentationsprogrammen kann der Benutzer Elemente der Präsentation nicht nur auf zwei, sondern auf drei Achsen verschieben. Dies eröffnet außerdem die Möglichkeit, 3D-Objekte wie Würfel, Kugel oder andere 3D-Modelle mit in die Präsentation einzubauen. Nützlich ist dies auch wieder für den Aspekt, dass das Programm Themen aus der Medieninformatik visualisieren soll, also besonders Vorgänge aus der Mathematik sowie der Informatik. Auch für das Einbinden von Animationen ist die Verwendung der dreidimensionalen Umgebung eine Neuerung. Da so eine neue Bewegungsachse gewonnen wird, können Animationen interessanter gestaltet werden.

Die Benutzeroberfläche soll aus klassischen zweidimensionalen Menüs bestehen, welche um den Präsentationsbereich angeordnet werden. Sie soll so simpel gestaltet sein, dass die Bedienung mit einem Finger über einen Touchscreen theoretisch möglich ist. Vorerst erfolgt die Umsetzung allerdings nur für die Maus als Eingabegerät.

Umgesetzt wird das Programm als Webanwendung. Der Benutzer muss also nur eine Webseite aufrufen und kann direkt seine Präsentation erstellen und bearbeiten. Diese Umsetzung als Webseite bringt einige Vorteile mit sich, die bei Verwendung einer nativen Programmiersprache nicht gegeben wären.

Einer dieser Vorteile ist es, dass das Programm plattformunabhängig funktioniert. Dies gilt nicht nur für bekannte Computer-Betriebssysteme wie Windows, Mac oder Linux, sondern auch für mobile Geräte die einen Webbrowser haben. Dadurch kann eine breite Abdeckung von unterstützten Plattformen erreicht werden. Der Fokus der Entwicklung liegt dabei auf Desktop-Systemen und dem Webbrowser Google Chrome[10].

Als Programmiersprache wird für die Umsetzung JavaScript benutzt, was sich aus mehreren Gründen anbietet. Zum einen ermöglicht JavaScript eine plattformunabhängige Programmierung, da es von allen modernen Browsern interpretiert werden kann. Außerdem ist mit der Einführung von HTML5 der Funktionsumfang von JavaScript deutlich ge-

[10] https://www.google.de/chrome/browser/desktop/ | 5.12.2015

wachsen. So ist es z.B. möglich Videos, Sounds und Animationen ohne Plug-Ins in Webseiten einzubinden. Diese Elemente kann man gut in einem Präsentationsprogramm gebrauchen und somit auch einfach umsetzen. Außerdem bietet sich JavaScript in Kombination mit HTML und CSS besonders an, um Benutzeroberflächen zu gestalten.

Im Allgemeinen gibt es viele Überschneidungen zwischen Webtechnik und Präsentationstechnik. Viele Webseiten müssen sich gut präsentieren und benutzerfreundlich sein, weshalb Webprogrammierung auf genau diese Aspekte ausgelegt ist, was dem Präsentationsprogramm zugutekommen kann.

Ein zusätzlicher Vorteil der Verwendung von JavaScript ist die einfache Erweiterbarkeit des Programms: Benutzer können eigenen Programmcode einbinden, der direkt auf der Webseite ausgeführt werden kann, was besonders das Schreiben eigener Animationen ermöglichen soll. Zudem gibt es viele, frei verfügbare JavaScript-Bibliotheken, die verwendet werden können, um schnell neue Funktionen einzubauen.

3.2 Verwendete JavaScript-Bibliotheken

Zur Umsetzung des Präsentationsprogramms wird eine Auswahl an verschiedenen JavaScript Bibliotheken verwendet, die einige grundlegende Funktionen bereitstellen. Die folgenden aufgeführten Bibliotheken werden auf die jeweils beschriebene Art verwendet:

3.2.1 Three.js

Mithilfe der JavaScript Bibliothek Three.js[11] wird eine 3D-Umgebung erzeugt, in der später die Präsentationsobjekte gezeigt werden. Three.js wird dazu genutzt, einen WebGL Kontext auf einer HTML-Seite zu erzeugt.

WebGL ermöglicht es dem Webbrowser direkt mit der Grafikkarte des Computers zu kommunizieren. So können aufwendige 3D-Szenen effizient im Browser dargestellt werden. Three.js bildet dabei eine Schnittstelle, die den komplizierten WebGL-Code in eine benutzerorientierte API umsetzt. Zudem stellt Three.js bereits viele vorgefertigte Objekte, Materialien und Methoden bereit, die die Arbeit mit WebGL deutlich vereinfachen.

Die Bibliothek wird im Zusammenhang mit dem entwickelten Programm in der Version „r73“ verwendet. Three.js ist eine frei zugängliche Bibliothek, die unter der MIT-Lizenz verwendet werden kann.

[11] https://github.com/mrdoob/three.js/ | 5.12.2015

3.2.2 JSZip

Mithilfe der Bibliothek JSZip[12] lassen sich in JavaScript ZIP-Container erstellen, die mit Dateien gefüllt werden können. Dabei übernimmt JSZip das Erstellen von Dateien und Ordnern sowie die anschließende Archivierung als ZIP-Datei. Ein großer Vorteil dieser Bibliothek ist, dass sie vollkommen Clientseitig und auf reinem JavaScript läuft. Das bedeutet, dass keine Verbindung zu einem Server hergestellt werden muss.

In Hinblick auf das Präsentationsprogramm wird diese Bibliothek benötigt, um erstellte Präsentationen als eine Datei zu speichern. Die Bibliothek kann sowohl unter der MIT-Lizenz, als auch der GPLv3-Lizenz verwendet werden. Für die Verwendung in dem zu erstellendem Programm wird die MIT-Lizenz gewählt.

3.2.3 FileSaver.js

Die Bibliothek FileSaver.js[13] bietet Funktionen an, um in der JavaScript-Umgebung erstellte Dateien als Download zu generieren.

Benötigt wird diese Bibliothek für den Speichervorgang von Präsentationen, wo sie dafür sorgt, dass die von JSZip.js generierten ZIP-Dateien dem Benutzer automatisch als Download angeboten werden. Sie wird mit einer Lizenz angeboten, die inhaltlich genau der MIT-Lizenz entspricht.

3.2.4 Spectrum.js

Spectrum.js[14] bietet einen JavaScript Color Picker an, der sich als HTML-Element auf eine Webseite einbinden lässt. Die Programmierung erfolgt in jQuery.

Dieser Color Picker bietet viele Konfigurationsmöglichkeiten, wie der Benutzer Farben auswählen kann. Zudem gibt es auch die Möglichkeit, eine vordefinierte Farbpalette zu bestimmen, die dem Benutzer angezeigt wird.

Verwendet wird diese Bibliothek, um es dem Benutzer zu ermöglichen, Präsentationsobjekte einzufärben. Sie wird unter einer Lizenz angeboten, die inhaltlich genau der MIT-Lizenz entspricht.

3.2.5 Easing.js

Die JavaScript-Datei easing.js[15] ist eine Sammlung von verschiedenen mathematischen Funktionen, die einen Easing-Wert zwischen eins und null berechnen. Dies wird später

[12] https://github.com/Stuk/jszip | 5.12.2015
[13] https://github.com/eligrey/FileSaver.js | 5.12.2015
[14] https://github.com/bgrins/spectrum | 5.12.2015
[15] https://gist.github.com/gre/1650294 | 5.12.2015

gebraucht, um den Übergang von Animationen zu berechnen. Der verwendete Code wurde von GitHub Gist entnommen, wo er ohne spezielle Lizenz frei zugänglich ist.

3.2.6 FontAwesome

Die Schriftart FontAwesome[16] ist eine Sammlung von Symbolen, die sich als einfache Textzeichen einbinden lassen. Die Zeichen werden dabei sowohl als Schriftartdatei, wie auch als Vektordatei bereitgestellt. Die von FontAwesome angezeigten Symbole sind dabei immer einfarbige Textzeichen, die sich in verschieden Kategorien einteilen lassen. Unter anderem werden auch Logos von bekannten Firmen wie Twitter oder Amazon angeboten.

In der programmierten Software finden diese Symbole Verwendung, um verschiedene Geometrien und Zeichen für die Präsentation bereitzustellen. Verfügbar ist die Schriftart unter der „SIL OFL 1.1"-Lizenz, welches eine Lizenz speziell für Open-Source-Schriftarten ist.

3.2.7 Carota.js

Mithilfe von Carota.js[17] lässt sich ein Texteditor erstellen, der auf einem Canvas-Element angezeigt wird. Der Texteditor besitzt alle nötigen Grundfunktionen, um Text zu formatieren und anzupassen.

Verwendet wurde diese Bibliothek um Textfelder für die Präsentation zu erstellen. Dazu wurde das von Carota.js erstellte Canvas-Element als Textur in die Three.js-Umgebung eingebunden. Diese Methodik brachte allerdings einige Probleme mit sich, die in Abschnitt 4.3.1 „Texte und Schriftarten" näher erläutert werden. In der aktuellen Version der Präsentationssoftware findet Carota.js keine Verwendung mehr. Die Bibliothek kann unter der MIT-Lizenz verwendet werden.

[16] https://fortawesome.github.io/Font-Awesome/ | 5.12.2015
[17] https://github.com/danielearwicker/carota | 5.12.2015

3.2.8 Material Design Lite

Um die Benutzeroberfläche anschaulich zu gestalten, wird Material Design Lite[18] verwendet. Dieses Paket bietet ein Bündel aus einer CSS- und einer JavaScript-Datei an, welches dazu verwendet werden kann, Webseiten zu gestalten. Für dieses Projekt werden lediglich einige der Grundfunktionen von Material Design Lite verwendet. Dazu müssen die entsprechenden HTML-Elemente nur mir den von Material Design Lite vorgegeben Klassen versehen werden.

Material Design Lite basiert auf dem „Material Design" von Google. Es stellt eine einfache, schnelle Lösung dar, um „Material Design" auf eine Webseite einzubauen. Die Verwendung wird unter der Apache-2-Lizenz ermöglicht.

[18] http://www.getmdl.io | 5.12.2015

4 UMSETZUNG DER BENUTZEROBERFLÄCHE

4.1 Bedienkonzept der Benutzeroberfläche

Die Darstellung der Benutzeroberfläche erfolgt über herkömmliche HTML und CSS-Elemente. Sie dienen zur Erstellung und Bearbeitung einer Präsentation und lassen sich in vier Bereiche einteilen: das Hauptmenü, die Präsentationsansicht, die Zeitleiste und das Aufgabenmenü.

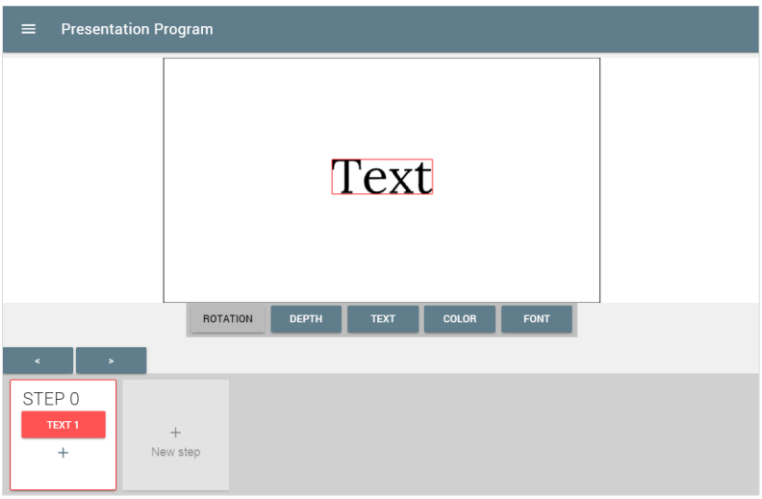

Abbildung 11 Benutzeroberfläche

Die Elemente sind in Streifen untereinander angeordnet und passen sich an verschiedene Auflösungen und Seitenverhältnisse an. In den folgenden Abschnitten werden die einzelnen Elemente etwas näher vorgestellt.

4.1.1 Präsentationsbereich

In diesem Bereich wird der aktuelle Stand der Präsentation angezeigt. Er dient dem Benutzer als Bearbeitungsfläche, auf der Präsentationsobjekte platziert und manipuliert werden können. Diese Ansicht ist dabei an die Zeitleiste gekoppelt und zeigt immer den aktuellen Schritt entsprechend der Zeitleiste an.

Dieser Bereich bildet das zentrale Element der Oberfläche und soll dem Benutzer einen möglichst guten Eindruck geben, wie die Präsentation später während des Vortrags, aussieht. Dazu können Rahmen in einem 4:3 oder 16:9 Verhältnis angezeigt werden, die widerspiegeln, was während der Präsentation zu sehen ist. Wahlweise können diese Rahmen ausgeblendet werden oder Rahmen mit einem benutzerdefinierten Seitenverhältnis einge-

bunden werden. Wichtig ist hier, dass die Rahmen immer die volle Höhe des Präsentationsbereichs einnehmen müssen und sie nur in der Breite variieren. Dadurch wird die Höhe als allgemeiner Maßstab genommen, der bei allen Seitenverhältnissen gleich ist. Dies ist wichtig, damit sich Präsentationsobjekte nicht in der Höhe verschieben und dadurch unten oder oben vom Bildschirmrand abgeschnitten werden.

Das Hinzufügen neuer Präsentationsobjekte erfolgt über die Zeitleiste, welche durch den Benutzer anschließend beliebig auf dem Präsentationsbereich platziert werden können. Dies kann durch ziehen mit der Maus oder dem Finger erfolgen.

4.1.2 Hauptmenü

Das Hauptmenü befindet sich am linken Rand des Fensters und wird nur eingeblendet, wenn es über die Hauptmenüschaltfläche aufgerufen wird. Es beinhaltet allgemeine Einstellungen und Aktionen.

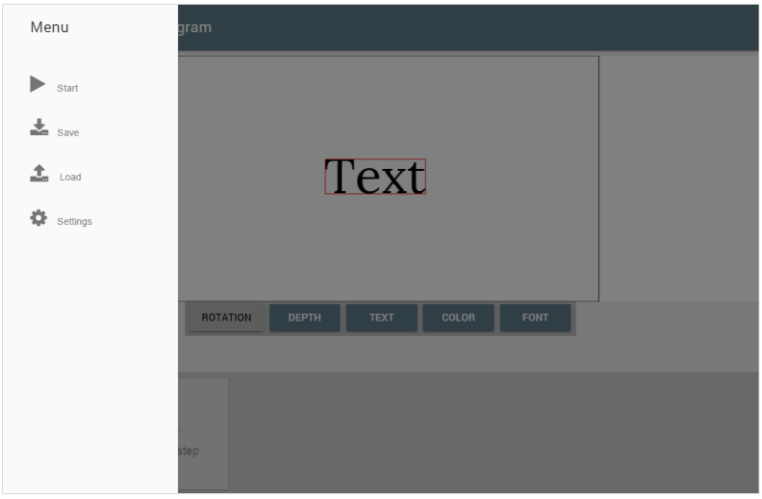

Abbildung 12 Funktion des Hauptmenüs

Unter anderem lässt sich die Präsentation hier starten. Dies hat zur Folge, dass ein zweites Browserfenster geöffnet wird, in dem lediglich der Präsentationsbereich angezeigt wird. Dieses Fenster kann dann auf einen zweiten Monitor oder Beamer geschoben werden. Durch einen Klick auf das neue Fenster wechselt dies in den Vollbildmodus. Der Benutzer hat nun zwei Fenster, eins für die Zuschauer und eins für sich. Auf beiden Fenstern wird immer live der gleiche Inhalt des Präsentationsbereiches angezeigt.

Weitere Punkte des Hauptmenüs sind das Speichern und Laden von erstellten Präsentationen, sowie ein Einstellungen-Dialog. In diesem lassen sich allgemeine Einstellungen der Präsentation konfigurieren, wie z.B. die Hintergrundfarbe. Nähere Informationen, wie

der Benutzer eine Präsentation speichern und laden kann, wird im Abschnitt 4.4 „Prinzip der Speicherfunktion“ erläutert.

4.1.3 Zeitleiste

Die Zeitleiste befindet sich am unteren Rand des Fensters und nimmt dort die volle Breite ein. Sie besteht aus mehreren Abschnitten. Im oberen Abschnitt finden sich Bedienelemente zur Navigation durch den Zeitstrahl, mit denen der Benutzer zum nächsten oder vorherigen Schritt springen kann. Diese sind als zwei Schaltflächen auf der linken Seite angeordnet.

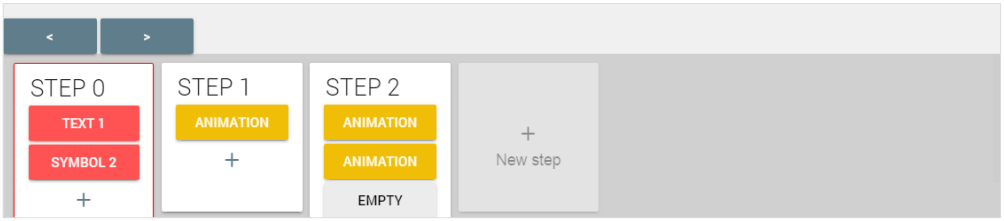

Abbildung 13 Aufbau der Zeitleiste

Unterhalb dieses Elements befindet sich der Hauptbereich der Zeitleiste. Entsprechend der Funktion der Zeitleiste findet eine grafische Unterteilung der Zeitleiste in Schritte und Aufgaben statt.

Schritte werden als kleine Karten angezeigt, die als Überschrift eine numerische Bezeichnung erhalten. Dementsprechend werden sie chronologisch von links nach rechts sortiert und bieten immer am rechten Ende eine Schaltfläche, über die ein neuer Schritt hinzufügt werden kann.

In jedem Schritt lassen sich beliebig viele Aufgaben hinzufügen. Jede Aufgabe wird durch eine kleine grafische Box repräsentiert. Die Aufgabenboxen werden in ihrem jeweiligen Schritt untereinander angeordnet, wobei die Reihenfolge der Boxen innerhalb eines Schrittes keine Bedeutung hat. Über Drag&Drop lassen sich die Aufgabenboxen sortieren und können so anderen Schritten zugeordnet werden.

4.1.4 Aufgabenmenü

Das Aufgabenmenü wird unterhalb des Präsentationsbereichs angezeigt und gibt dem Benutzer die Möglichkeit, Eigenschaften von Aufgaben und Präsentationsobjekten zu modifizieren.

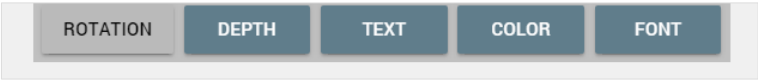

Abbildung 14 Darstellung des Aufgabenmenüs

Sobald eine Aufgabe oder ein Präsentationsobjekt ausgewählt wird, öffnet sich das Aufgabenmenü und zeigt die für die gewählte Aufgabe nötigen Einstellungen. Ist gerade weder ein Präsentationsobjekt noch eine Aufgabe ausgewählt, wird dieses Menü versteckt.

Dies soll dafür sorgen, dass dem Benutzer nur die nötigsten Menüs und Modifizierungsmöglichkeiten angezeigt werden, wodurch die Gesamterscheinung der Benutzeroberfläche aufgeräumter wirken soll. Dazu soll auch beitragen, dass im Aufgabenmenü immer nur die für die gewählte Aufgabe nötigen Menüpunkte angezeigt werden. Diese werden zuvor in den verschiedenen Aufgabentypen definiert und speziell auf diese Aufgabe angepasst.

4.1.5 Zusammenspiel der einzelnen Bereiche

Zur Verdeutlichung des Zusammenhangs der einzelnen Bereiche werden visuelle Markierungen angezeigt. Diese Markierungen erscheinen z.B., wenn ein Präsentationsobjekt im Präsentationsbereich angeklickt wird. Dieses Präsentationsobjekt gilt dann als ausgewählt und ein Rahmen wird angezeigt, der diese Auswahl verdeutlichen soll.

Ebenso wird ein Rahmen um ausgewählte Aufgaben angezeigt. Dabei wird unterschieden, ob ein Präsentationsobjekt im Präsentationsbereich ausgewählt wurde oder eine Aufgabe in der Zeitleiste angeklickt wurde. Bei ersterem werden alle Aufgabenboxen, die mit dem ausgewähltem Präsentationsobjekt in Verbindung stehen, mit einer visuellen Kennzeichnung versehen. Beim Klick auf eine Aufgabe in der Zeitleiste werden die gewählte Aufgabenbox und das entsprechende Präsentationsobjekt markiert.

In diesen Zusammenhang ist auch das Verhalten des Aufgabenmenüs wichtig. Ist das Aufgabenmenü geschlossen und ein Präsentationsobjekt wird angewählt, öffnet es sich und zeigt die Menüpunkte der Aufgabe an, durch die das Präsentationsobjekt hinzugefügt wurde. War das Menü bereits geöffnet und die gewählte Aufgabe steht mit dem neu gewählten Präsentationsobjekt in Zusammenhang, bleibt das Menü der gewählten Aufgabe geöffnet.

Klickt der Benutzer im Präsentationsbereich auf eine Stelle, an der sich kein Präsentationsobjekt befindet, werden alle Markierungen entfernt und das Aufgabenmenü geschlossen. In diesem Zustand sind alle Präsentationsobjekte abgewählt.

4.2 Darstellung der Präsentation

Die Darstellung der Präsentation erfolgt in einer 3D-Umgebung, die alle Objekte der Präsentation anzeigt. Eine von Three.js erzeugte 3D-Szene wird auf einem HTML-Canvas Element angezeigt, welches sich wie andere üblichen HTML-Elemente einbinden lässt. Dadurch ist es einfach, die Darstellung der Szene in die Benutzeroberfläche einzubinden. Außerdem kann sich dieses Element auch dynamisch an verschiedene Bildschirm- und Fenstergrößen anpassen.

Durch die Verwendung von WebGL und Three.js ergibt sich der Vorteil, dass die erstellte Präsentationsszene mehrfach gerendert werden kann. Das bedeutet, dass die Präsentation in verschiedenen Fenstern angezeigt werden kann, was besonders dann wichtig ist, wenn die Präsentation auf verschiedenen Monitoren oder einem Beamer angezeigt werden soll. Dann kann die erstellte WebGL-Szene mehrfach angezeigt werden, wobei jede Darstellung in Fenstergröße und Seitenverhältnis variieren kann. Durch diese Technik ist es möglich, auf einem Beamer die Präsentation für Zuschauer im Vollbild anzuzeigen, während der Vortragende auf seinem Bildschirm z.B. eine Notizansicht sieht, die eine kompaktere Darstellung der Präsentationsszene beinhaltet. Da beide Darstellungen die gleiche Szene anzeigen, ist es für den Vortragenden auch möglich, Änderungen an der Szene vorzunehmen, die in Echtzeit in der Darstellung für die Zuhörer geändert werden. Dies soll vor allem dazu genutzt werden, dynamische und interaktive Präsentationen zu erstellen.

4.3 Präsentationsobjekte

Zur Erstellung einer ansprechenden Präsentation wird eine Vorauswahl an Textfeldern, Symbolen und Geometrien angeboten, die im Präsentationsbereich platziert werden können. Diese Objekte werden im weiteren Verlauf Präsentationsobjekte genannt. Hinzugefügt werden neue Präsentationsobjekte, indem eine neue Aufgabe erstellt wird. Dies öffnet dann den Präsentationsobjekt-Auswahldialog, mit dem der gewünschte Typ von Präsentationsobjekt gewählt werden kann.

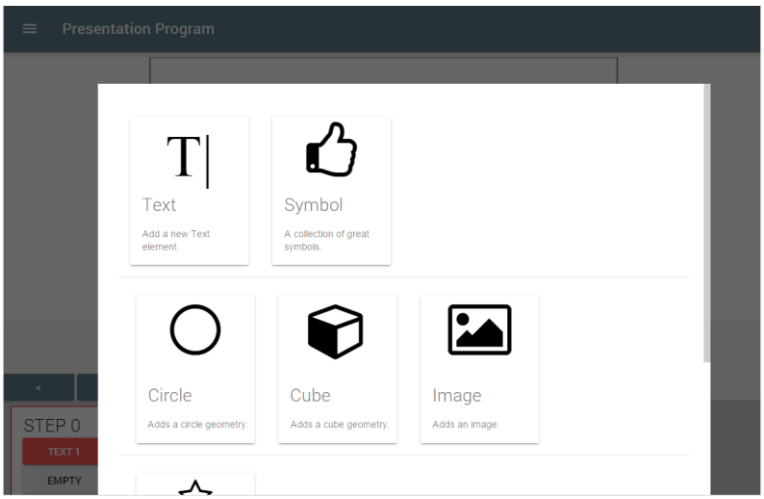

Abbildung 15 Präsentationsobjekt-Auswahldialog

Es folgt nun eine kurze Zusammenfassung der verschiedenen Kategorien von Präsentationsobjekten.

4.3.1 Texte und Schriftarten

Das Einbinden von Texten wurde testweise auf zwei verschiedene Arten umgesetzt. Dabei kamen jeweils unterschiedliche Techniken zum Einsatz, um den Text als 3D-Text oder als Textfeld zu benutzen.

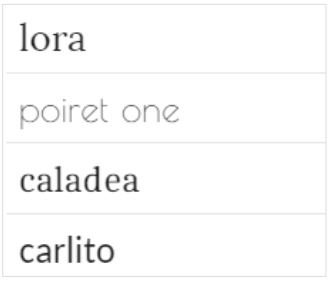

Abbildung 16 Auswahl an OpenSource-Schriftarten

Ein 3D-Text wird als dreidimensionales Objekt auf dem Präsentationsbereich angezeigt. Dabei kann ein 3D-Text auch eine Tiefe haben. Mithilfe von facetype.js[19] können normale Schriftartdateien in JavaScript-Dateien konvertiert werden, so dass alle üblichen Schriftarten auch mit

[19] https://github.com/gero3/facetype.js | 5.12.2015

Three.js eingesetzt werden können. Dies erfordert allerdings eine manuelle Konvertierung der Schriftarten. Es werden einige OpenSource-Schriftarten angeboten, die bereits bei der Programmierung konvertiert und standardmäßig in das Programm eingebunden wurden.

Die zweite Methode wurde genutzt, um Textfelder zu erzeugen. Dabei wurde der Text als eine Textur in Three.js eingebunden. Um die Textur zu generieren, wurde mithilfe der JavaScript-Bibliothek *carota.js* ein Canvas erzeugt, welcher vorerst als normales HTML-Element auf die Seite eingebunden wurde. Dieser Canvas konnte dann als Textur in Three.js gespiegelt werden.

Eins der größten Probleme dieser Methodik war es, dass der Text nicht pixelgenau dargestellt wurde, wodurch die Schrift unscharf wirkte. Dieses Problem rührt daher, dass sich die Größe des Textfelds auf dem Präsentationsbereich zu der Größe des von *carota.js* erstellten Canvas unterscheidet. Das führt dazu, dass die Textur gestreckt oder gestaucht werden muss, was zu dem unscharfen Ergebnis führt. Aufgrund dieser und einiger weiteren Probleme wurde dieses Verfahren nicht weiter umgesetzt.

4.3.2 Symbole

Für eine gute Präsentation sind Symbole und Grafiken ebenso wichtig wie Texte. Um eine kleines Kontingent von Symbolen zu bieten, wird die Schriftart FontAwesome verwendet. Diese speziell auf Symbole fokussierte Schriftart bietet viele simple Icons, die gut die Funktion von Pfeilen und Clip Arts in PowerPoint ersetzen können.

Abbildung 17 Auswahl an Symbolen der FontAwesome Bibliothek

Über eine Unicode-Tabelle können die Icons einfach eingebunden und für den Benutzer sichtbar aufgelistet werden.

4.3.3 Geometrien

Um eine Präsentation zu gestalten, wird noch ein kleiner Satz an simplen Geometrien wie z.B. Kreisen, Rechtecken oder Dreiecken angeboten. Diese Geometrien besitzen ähnliche Eigenschaften, die vom Benutzer gesetzt werden können: u.a. die Größe, Farbe und Rotation. Außerdem ist es möglich, mithilfe von Three.js-Code, weitere Geometrien einfach zu ergänzen.

4.4 Prinzip der Speicherfunktion

Natürlich ist es für ein solches Programm sehr wichtig, eine Funktion anzubieten, die die vom Benutzer erstellte Arbeit sichern kann. Dabei sollte die Speicherung einige Anforderungen erfüllen, die es dem Benutzer einfach machen mit dem Programm sicher und einfach zu arbeiten. In der implementierten Variante kann durch einen Menüpunkt die Präsentation als Datei lokal auf dem Rechner des Benutzers gespeichert werden. Diese Datei wird automatisch generiert und dem Benutzer zum Download angeboten und kann dann mit einem beliebigen Dateinamen an einem benutzerdefinierten Pfad gespeichert werden.

Der Vorgang zur Generierung dieser Datei geschieht dabei lokal auf dem Computer des Benutzers, es wird also keine Internetverbindung benötigt. Dies ist dann von Vorteil, wenn der Benutzer offline arbeiten möchte.

Über einen weiteren Menüpunkt kann die gespeicherte Datei wieder geladen werden. Dabei werden die zuvor gesetzten Aufgaben wieder erstellt und mit Präsentationsobjekten verknüpft.

Eine denkbare Erweiterung dieses Systems ist z.B. automatisches Speichern oder das Speichern auf einem online Server. Da die Onlinespeicherung einen Server und ein System für Benutzerkonten voraussetzt, würde die Implementierung eines solchen Verfahrens die Komplexität dieses Projektes überschreiten und wird deshalb nicht umgesetzt.

Ein System, welches in regelmäßigen Abständen automatische Speicherungen vornimmt, wurde aufgrund der schlechteren Handhabung nicht implementiert. Leider bietet JavaScript keine verlässliche Möglichkeit, Dateien auf dem Computer des Benutzers zu erstellen oder zu verwalten. Dies müsste über Cookies oder den sogenannten „HTML Local Storage“ erfolgen. Für den Benutzer ist bei diesen Methoden allerdings nicht ersichtlich wo sich eine gespeicherte Datei befindet, da sie lediglich im Browsercache abgelegt wird. Aus diesen Gründen ist es nicht sinnvoll eine automatische Speicherung zu implementieren, da dem Benutzer jedes Mal ein Downloaddialog angezeigt werden müsste.

Zudem kann aus Sicherheitsgründen die von JavaScript vorgegebenen sind, kein automatischer Download einer Datei gestartet werden, ohne dass der Benutzer eine Anweisung, wie z.B. den Klick auf einen Button, gegeben hat. Diese technischen Einschränkungen von JavaScript führen dazu, dass der Benutzer den Download einer generierten Datei jedes Mal manuell durchführen muss.

5 UMSETZUNG DER PROGRAMMIERUNG

Die Programmierung der Software erfolgt in der Programmiersprache JavaScript. Im Zusammenspiel mit HTML und CSS wird das Frontend entworfen und erhält durch JavaScript seine Funktion. Die Programmierung ist dabei nicht ausschließlich objektorientiert, sondern enthält auch Abschnitte in imperativer Programmierung. Diese Mischung aus rein imperativer Programmierung und Datenkapselung soll die Gestaltung des Codes möglichst simpel machen. Da sich der Code so an die einzelnen umgesetzten Bereiche gut anpassen kann, werden die Fähigkeiten von JavaScript optimal ausgereizt.

Der Aufbau des Programmcodes wird in die drei Zuständigkeitsbereiche **Framework**, **Aufgaben-Typen** und **Benutzeroberfläche** aufgeteilt. Der imperative Teil der Programmierung wird dabei Hauptsächlich für die Funktion der Benutzeroberfläche sowie den Ablauf der Hauptfunktionen des Programms verwendet.

Das **Framework** soll den grundsätzlichen Ablauf des Programms steuern und auf Benutzereingaben reagieren. Dabei lädt es Aufgaben-Typen und verwaltet auf dem Präsentationsbereich platzierte Präsentationsobjekte. Außerdem soll das Framework die von Three.js bereitgestellte 3D-Engine verwalten und die damit erzeugten Objekte entsprechend des Fortschritts der Präsentation anzeigen.

Der als „**Aufgaben-Typen**“ bezeichnete Teil der Programmierung spiegelt die verschiedenen Präsentationsobjekte wieder. Diese werden in verschiedene Typen, wie z.B. Text, Bild oder Animation unterteilt. Jeder Typ von Präsentationsobjekt wird als sogenanntes Modul erzeugt. Der Aufbau eines solchen Moduls folgt immer einer bestimmten Struktur, welche für alle verschiedenen Typen geeignet sein soll. Diese allgemeingehaltene Programmierung soll dafür sorgen, dass stätig neue Module ergänzt werden können und das Programm somit erweiterbar bleibt.

Der letzte Bereich der Programmierung stellt die Funktion und das Verhalten der **Benutzeroberfläche** da. Hier erhalten die auf der HTML-Seite platzierten Elemente ihre Funktion und kommunizieren dabei mit dem Framework. Dies beinhaltet auch die Funktionsweisen der verschiedenen Menüs.

Die folgenden Kapitel stellen einige Elemente der Programmierung vor. Diese werden anhand von Auszügen aus dem Programmcode und einigen Beispielen erklärt.

5.1 Framework und 3D-Engine

Die Aufgabe des Frameworks ist es, dem Benutzer eine simple Grundlage zu geben, die er einfach und möglichst uneingeschränkt erweitern und an seine eigenen Bedürfnisse anpassen kann. Deshalb werden einzelne Präsentationsobjekte in Module aufgeteilt, die durch das Framework verwaltet werden. Das heißt, dass das Framework für zwei verschiedene Bereiche zuständig ist. Zum einen stellt es die nötigen Funktionen zur Verwaltung der Schritte und zum Ablauf der Präsentation bereit. Zum anderen sorgt es für die Kommunikation zu Three.js, welches die Präsentationsobjekte anzeigt. Um diese Trennung auch im Programmcode widerzuspiegeln, sind die Bereiche in zwei verschiedene Dateien aufgeteilt, die **canvas.js** und die **timeline.js**.

Die **canvas.js** beinhaltet Funktionen zur Verarbeitung des durch Three.js erstellten Canvas-Elements und bildet damit den Präsentationsbereich. Die in der Datei enthaltenen Funktionen sind dabei für die folgenden Abläufe zuständig:

- Den Initialisierungsvorgang von Three.js
- Die Generierung von Präsentationsobjekten als Mesh
- Die Verarbeitung von Benutzereingaben, bezogen auf den Präsentationsbereich

Technisch ist der Präsentationsbereich eine Three.js-Szene in der 3D-Objekte angezeigt werden. Das bedeutet, dass die Szene mehrfach gerendert und in verschiedenen Fenstern angezeigt werden kann. Dazu wird ein zweites Browserfenster geöffnet, in dem ein weiteres Canvas-Element erzeugt wird, welches ebenfalls die Szene zeigt. Dieser Vorgang wird auch von der *canavs.js* verwaltet.

In einer weiteren Datei, der **timeline.js**, befinden sich Grundfunktionen des Frameworks, die für den Hauptablauf des Programms zuständig sind. Dazu zählen unter andrem die Funktionsweise der Zeitleiste, das Abspielen von Animationen und das Verwalten der Präsentationsobjekte.

Ein weiterer Punkt ist das Generieren von Präsentationsobjekten aus Aufgaben-Typen. Der genaue Aufbau und die Funktion von Aufgaben-Typen wird im Abschnitt 5.2 „Aufgaben-Typen“ näher erklärt. Die Aufgabe der *timeline.js* ist es, bei der Initialisierung verschiedene Aufgaben-Typen zu laden und bei Bedarf Präsentationsobjekte aus ihnen abzuleiten.

Für den Benutzer werden die verschiedenen Typen in einem Overlaymenü präsentiert. Dort kann er dann einen Aufgaben-Typen auswählen, was dazu führt, dass das Framework das entsprechende Präsentationsobjekt erstellt und in die Präsentation integriert. Außerdem wird ein passendes HTML-Element erzeugt, welches die Aufgabe in der Zeitleiste widerspiegelt.

Da der Benutzer die Möglichkeit hat, Aufgaben-Typen selbst zu erstellen oder weitere Typen einzubinden, muss das Overlay, welches die einzelnen Typen anzeigt, dynamisch aufgebaut und erweiterbar sein. Dazu werden bei der Initialisierung alle vorhandenen Typen ausgelesen und entsprechende Menüpunkte im Overlaymenü erstellt. Die einzelnen Aufgaben-Typen werden anschließend in verschiedene Katagorien sortiert.

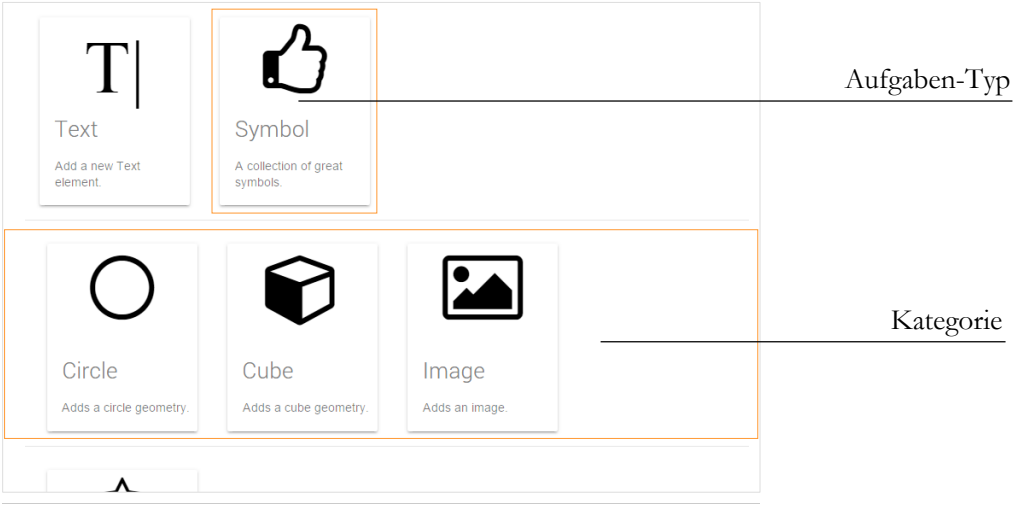

Abbildung 18 Overlaymenü - Kategorien

Wie in Abschnitt 5.2 „Aufgaben-Typen“ noch genauer erklärt, befinden sich alle Aufgaben-Typen gesammelt in einem Objekt, dem *typeContainer*. Die Aufgabe des Frameworks ist es nun, für alle Typen einen passenden Menüpunkt zu erzeugen. Der Aufbau eines Menüpunkts setzt sich dabei immer aus einem Bild, einer Überschrift und einem Beschreibungstext zusammen. Diese Daten werden für jeden Typen speziell definiert und finden sich in der Variable *meta* des Aufgaben-Typen wieder.

```
meta:{
       category: "text",
       name: "Symbol",
       background: htmlElement,
       description: "A collection of great symbols."
}
```

Die *meta*-Variable ist ein Objekt, welches verschiedene Stringvariablen beinhaltet. Diese werden zum Aufbau des Menüpunktes genutzt.

Anschließend werden die Menüpunkte in verschiedene Kategorien gruppiert. Dabei gibt es vier verschiedene Kategorien: Text, Geometrien, Animationen und Entfernen. Dieser ganze Vorgang ist in der Methode *initTaskTypes* wiederzufinden.

Während der Initialisierung der *timeline.js* werden nicht nur die Aufgaben-Typen geladen, sondern auch Funktionen für Benutzerelemente festgelegt. Dies beinhaltet die meisten Schaltflächen und Menüs der Benutzeroberfläche. Exemplarisch werden einige dieser Vorgänge in den kommenden Abschnitten näher erläutert.

Während der Benutzer eine Präsentation bearbeitet, kann er zu jedem Zeitpunkt einen neuen Schritt zur Zeitleiste hinzufügen. Dafür muss der Benutzer lediglich eine Schaltfläche am Ende der Zeitleiste klicken. Im Code führt dies dazu, dass die Methode *addStep* aufgerufen wird. Diese Methode erzeugt ein neues DIV-Element, welches mit nötigen CSS-Klassen versehen wird. Dadurch wird dem neuen Schrittelement ein einheitliches Aussehen gegeben. Außerdem wird der Inhalt des DIVs mit einigen HTML-Elementen gefüllt, die eine Überschrift für den Schritt bilden und eine Schaltfläche bieten, mit der neue Aufgaben in den Schritt eingefügt werden können. Die Überschifft ist lediglich das englische Wort „Step“, gefolgt von einer Zahl, die die Position des Schrittes widerspiegelt. Zum Schluss wird das erstellte DIV-Element noch in die Reihe von anderen DIV in der Zeitleiste eingefügt.

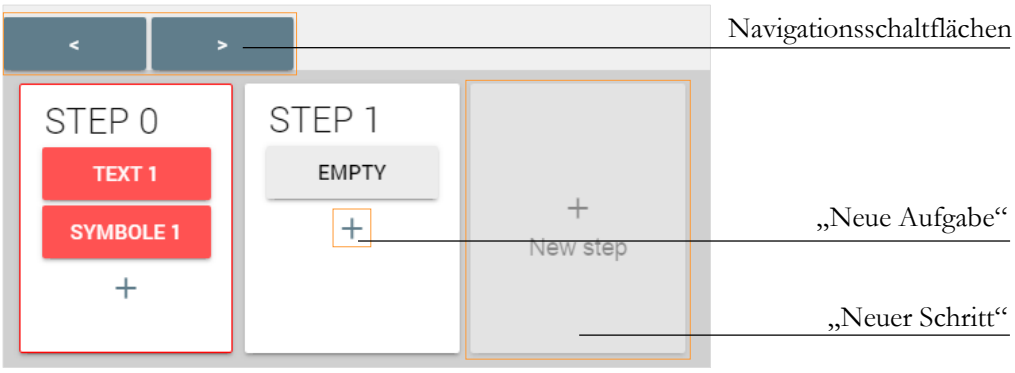

Abbildung 19 Zeitleiste mit zwei Schritten

Weitere, etwas komplexere Vorgänge, werden durch die Vor- und Zurück-Schaltflächen in Gang gesetzt. Diese haben die Aufgabe, dem Benutzer eine Möglichkeit zu geben, durch die Präsentation zu navigieren. Verknüpft sind diese Schaltflächen mit den Methoden *nextStep* und *prevStep* der *timeline.js*.

Da der Benutzer zu jedem Zeitpunkt die Möglichkeit hat, zu einem anderen Schritt der Präsentation zu wechseln, entsteht das Problem, dass Animationen möglicherweise abgebrochen werden müssen. Dies kann dazu führen, dass es zu Fehldarstellungen kommt.

Um das Problem etwas genauer zu beschreiben, nehmen wir als simples Beispiel an, dass wir ein Präsentationsobjekt in einer Bewegung von links nach rechts animieren wollen. Dafür speichern wir die linke Position im ersten Schritt und die zweite Position im zweiten Schritt. Ist nun Schritt eins ausgewählt und der Benutzer klickt auf die „Nächster Schritt"-Schaltfläche, wird die Animation wie geplant angezeigt und das Präsentationsobjekt wandert von links nach rechts. Der derzeit aktive Schritt ist nun Schritt zwei. Möchte der Benutzer nun wieder einen Schritt zurückspringen kann er dies durch die „Schritt zurück"-Schaltfläche tun. Nun stellt sich aber die Frage, wie sich die Animation und damit auch das entsprechende Präsentationsobjekt verhält. Da wir Schritt eins die linke Position des Präsentationsobjekts zugewiesen haben, muss sich das Objekt korrekter Weise auch wieder an dieser Position befinden, wenn der erste Schritt ausgewählt wird. Da wir aber von Schritt zwei kommen, befindet sich das Objekt noch auf der rechten Seite des Präsentationsbereichs. Eine Lösung für das Problem wäre es, das Objekt einfach zurückspringen zu lassen, so dass es sofort seine korrekte Position auf der rechten Seite einnimmt. Dies wäre eine visuell unschöne Lösung, da die erstellte Bewegung des Präsentationsobjekts entfällt. Die bessere Variante ist es, die zuvor gesehene Animation rückwärts abzuspielen. Die Aufgabe des Frameworks ist es nun, diese Situation richtig zu erkennen und die entsprechende Animation korrekt abzuspielen. Es muss feststellen, dass der Benutzer einen Schritt zurück gesprungen ist und die Animation auch dementsprechend rückwärts gezeigt werden muss.

Dies ist eine sehr simple Darstellung eines Problems, was immer dann auftritt, wenn der Benutzer zu einem anderen Schritt wechselt. Deutlicher wird das Problem wenn sich mehrere Schritte in der Zeitleiste befinden, die Objekte hinzufügen oder entfernen. Da der Benutzer auch die Möglichkeit hat, zu einem bestimmten Schritt zu springen, kann es passieren, dass Schritte übersprungen werden. Ist der aktuelle Schritt z.B. Schritt vier und der Benutzer springt zu Schritt eins, werden Schritt zwei und drei übersprungen. Wird in diesen Schritten ein Objekt hinzugefügt oder entfernt, kann es nun passieren, dass der aktuelle Schritt fehlerhaft angezeigt wird. Um diese Art von Fehler zu vermeiden, gibt es verschiedene Ansätze.

Die erste Idee wäre, den Zustand von jedem Präsentationsobjekt in jedem einzelnen Schritt zu speichern. Das heißt, wenn der Benutzer zu einem Schritt wechselt, arbeitet das Framework alle Präsentationsobjekte ab und bearbeitet diese entsprechend der Werte, die

im gewählten Schritt hinterlegt sind. Dies wäre allerdings eine recht ineffiziente Methode, die unnötig viele Werte speichern müsste.

Ein anderes Vorgehen wäre es, immer nur die Werte der Präsentationsobjekte zu speichern, die in dem jeweiligen Schritt geändert werden. Wenn der Benutzer zu einem Schritt wechselt und dabei andere Schritte übersprungen werden, müsste das Framework nun alle zwischenliegenden Schritte abarbeiten. Dies würde sicherstellen, dass alle Modifikationen an einem Objekt berücksichtigt werden.

Die zweite Idee ähnelt der implementierten Variante am stärksten. Allerdings wurde sie noch nicht so effizient umgesetzt, wie hier beschrieben. Für den Benutzer sollte dies allerding keine spürbaren Auswirkungen auf die Bedienung des Programmes haben. Dies ist ein Bereich, der noch optimiert werden könnte, um z.B. die Akkulaufzeit von portablen Geräten zu verlängern.

5.2 Aufgaben-Typen

Die im Programmcode als TaskTypes bezeichneten Aufgaben-Typen stellen jeweils eine bestimmte Kategorie von Präsentationsobjekten, wie z.B. Texte, Symbole und Geometrien dar. Dabei ist ein TaskType ein JavaScript-Objekt, welches einer bestimmten Struktur folgt. Auf diesen TaskTypes basiert das dynamisch erweiterbare System, welches es dem Benutzer ermöglichen soll, eigene Präsentationsobjekte zu programmieren und in Präsentationen einzubinden.

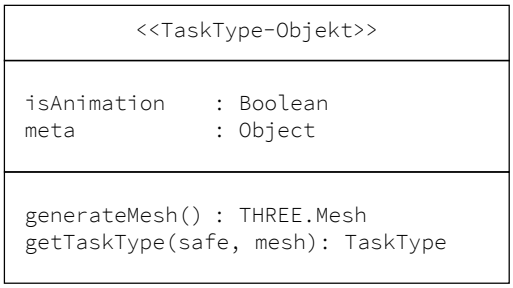

Abbildung 20 UML eines TaskType-Objekts

Einige standardmäßig definierte TaskTypes finden sich in der *tasktypes.js* wieder und werden dort einem Objekt, dem typeContainer, als Attribute angehangen. Dieses Attribut referenziert dabei ein Objekt, in welchem das Framework die drei Attribute *isAnimation*, *generateMesh* und *getTasktype* erwartet. Diese drei Variablen ermöglichen es dem Framework später, aus den abstrakten Aufgaben-Typen konkrete Präsentationsobjekte zu erstellen. Alle weiteren Attribute werden vom Framework ignoriert und können zur Implementierung des Aufgaben-Typs frei definiert und genutzt werden.

Um die neue Aufgabe zu erstellen, wird vom Framework zuerst die Variable *isAnimation* überprüft, die durch einen *boolean*-Wert angibt, ob es sich bei dem Aufgaben-Typen um eine Animation handelt. Ist dies nicht der Fall, wird die Methode *generateMesh* aufgerufen, die als Rückgabewert ein Three.js-Mesh liefert. Dieses Mesh wird vom Framework auf dem Präsentationsbereich platziert und entsprechend dem aktuellen Schritt angezeigt oder animiert. Das Generieren eines Meshes wird immer dann ausgeführt, wenn der Benutzer ein neues Präsentationsobjekt dieses Typs hinzufügt. Das Framework speichert dann für jede Aufgabe das entsprechende generierte Mesh-Objekt.

Zuletzt wird vom Framework die Methode *getTaskType* aufgerufen, die die zwei Attribute *mesh* und *safe* entgegennimmt. Im *mesh*-Attribut findet sich das durch die *generateMesh*-Methode generiete Mesh-Objekt wieder, während die *safe*-Variable ein JavaScript-Objekt beinhaltet.

Die *safe*-Variable spielt eine wichtige Rolle bei der Speicherung der Präsentation und wird dazu verwendet, Eigenschaften der durch den TaskType erzeugten Aufgabe zu sichern.

Diese Eigenschaften, wie z.B. Größe oder Farbe eines Objekts, sollten in dem bereitgestellten *safe*-Objekt abgelegt werden. Dafür muss lediglich eine neue Variable in dem Objekt angelegt werden. Bei der nächsten Speicherung werden dort abgelegte Werte vom Framework mit gesichert, um sie später wieder laden zu können.

Wird vom Benutzer eine vorhandene Präsentation geladen, erstellt das Framework zunächst eine leere Aufgabe. Deren Typ wird anschließend durch die in der Speicherung abgelegten Daten bestimmt. Von diesem Typen wird wieder die *getTaskType*-Methode aufgerufen. Dieses Mal wird allerdings für die *safe*-Variable kein leeres Objekt übergeben, sondern eine Kopie des alten *safe*-Objekts, welches die Eigenschaften beinhaltet, die zuvor gesetzt wurden.

Das bedeutet, dass die *safe*-Variable zwei Zustände haben kann, entweder ein leeres oder eine zuvor erstelltes Objekt, welches zuvor gespeicherte Werte enthält. Das führt dazu, dass in der *getTaskType*-Methode geprüft werden muss, ob ein Attribut in der *safe*-Variable gesetzt ist, ansonsten muss auf einen Standartwert zurückgegriffen werden. Diese Bedingung lässt sich in JavaScript in einer Zeile überprüfen:

```
var attr = safe.attribute ? safe.attribute : standardValue;
```

Diese Zeile Code überprüft, ob der Wert *attribute* im *safe*-Objekt existiert und gesetzt ist. Ist dies der Fall, wird der Wert in die Variable *attr* kopiert. Ist dies jedoch nicht der Fall, wird auf den Wert, der sich in der Variable *standardValue* befindet, zurückgegriffen und dieser kopiert. Dieses Prinzip stellt sicher, dass sich anschließend in der Variable *attr* je nach Fall entweder der Standartwert oder der Wert aus einer geladenen Datei befindet.

Die Speicherung ist die einzige Funktion des Frameworks, die mit den Eigenschaften einer Aufgabe arbeitet. Sonst werden diese Attribute nur lokal in der *getTaskType*-Methode verwendet und nicht weiter durch das Framework genutzt. Dies soll eine hohe Flexibilität in der Funktionsweise der programmierten Aufgaben ermöglichen, führt allerdings auch dazu, dass der Vorgang zur Bearbeitung der Werte vom Benutzer programmiert werden muss.

Die einzige sonstige Schnittstelle zum Framework bildet das *taskType*-Objekt, welches zugleich auch Rückgabewert der Methode *getTaskType* ist. Dieses Objekt folgt wieder einer bestimmten Struktur und definiert für das Framework somit die Funktionsweise und Eigenschaften der Aufgabe. Zunächst muss dieses Objekt jedoch in der *getTaskType* Methode definiert werden.

Durch *new TaskType()* kann ein neues, leeres *taskType*-Objekt erzeugt werden, welches einige vordefinierte Attribute beinhaltet, die standardmäßig leere Objekte oder null sind. Diese können vom Benutzer programmiert werden. Die wichtigsten Attribute sind **name**, **animation** und **menu**, sowie die Methoden **onUpdate** und **onDelete**.

Die Variable **name** wird dazu verwendet, der Aufgabe einen Namen zu geben, der später auf der Benutzeroberfläche im HTML-Element der Aufgabe angezeigt wird. Der Typ der Variable ist ein einfacher String, wobei das Framework später die einzelnen Aufgabennamen der verschiedenen Typen jeweils durchnummeriert. So ist die z.B. die Bezeichnung für den Typen „Text“, später in der Zeitleiste „Text 1“, „Text 2“, usw.

Die Variable **animation** wird nur benötigt, wenn es sich bei der erstellten Aufgabe um eine Animation handelt. Handelt es sich um eine Animation, findet sich in dieser Variable ein Objekt wieder, welches den Ablauf und die Funktion der Animation bestimmt. Eine genauere Beschreibung zum Aufbau dieses Objektes ist im Abschnitt 5.3 „Animationen“ zu finden.

In der Variable **menu** befindet sich ein Objekt, welches vom Framework genutzt wird, um das Aufgabenmenü für diese Aufgabe zu erstellen. Hier befindet sich die eigentliche Schnittstelle, an der der Benutzer Eigenschaften der Aufgabe modifizieren kann. Bei der Initialisierung des *taskType*-Objekts ist das *menu*-Attribut ein leeres Objekt ohne Variablen. Jede hinzugefügte Variable spiegelt einen neuen Menüpunkt wider. Aus Gründen der Erweiterbarkeit muss jede dieser Variablen wieder ein eigenes Objekt sein, welches das Attribut *html* beinhaltet. In dieser Variable findet sich ein HTML-Element wieder, welches später den Menüpunkt im Aufgabenmenü widerspiegelt.

Momentan wird nur die *html*-Variable aus dem Menüpunkt-Objekt vom Framework ausgelesen. Denkbar ist, dies später um einige Variablen zu erweitern, die die Arbeit mit den Menüpunkten erleichtern. Nützlich wäre z.B. eine Variable, die den Menüpunkt nur unter einer bestimmten Bedingung anzeigt oder eine Variable, die das Aussehen des Menüpunktes bestimmen kann.

Theoretisch kann im Aufgabenmenü völlig frei bestimmt werden, welche HTML-Elemente eingefügt und angezeigt werden. Dies erfordert allerdings auch, dass für jeden Menüpunkt ein HTML-Element erzeugt werden muss. Dadurch entsteht die Gefahr, dass dieses möglicherweise stilistisch nicht zur Benutzeroberfläche passt oder sogar falsch angezeigt wird. Um diesen Vorgang zu vereinfachen und zu vereinheitlichen, werden in der *tasktypes.js* einige Methoden angeboten, die das Generieren der HTML-Elemente übernehmen. Eine dieser Methoden ist die *createTaskMenuButton*-Funktion. Sie erzeugt einen

Button für das Aufgabenmenü und stellt die Möglichkeit bereit, eigene HTML-Elemente in einem Dropdown anzuzeigen. Dieses Dropdownfeld wird sichtbar, sobald der Benutzer auf den zugehörigen Menüpunkt klickt.

```
function createTaskMenuButton(text, content){
    var div = document.createElement("div");
    div.className = …
    div.innerHTML = text || "";

    div.dropdown = document.createElement("div");
    div.dropdown.className = …
    if(content)div.dropdown.appendChild(content);

    div.onclick = function(){
        //show dropdown
    };

    return div;
}
```

Zur Erstellung einer solchen Schaltfläche nimmt die Methode *createTaskMenuButton* zwei Argumente entgegen: *text*, eine Zeichenkette zur Benennung des Menüpunktes und *content*, ein HTML-Element, welches später im Dropdown angezeigt werden soll. Die Methode erzeugt dazu zwei DIV-Elemente. Das erste bildet dabei die beschriftete Schaltfläche und das zweite den Dropdown. Der Rückgabewert der Methode ist das Element, welches die Schaltfläche bildet. Möchte der Benutzer das Dropwon-Element an eigene Bedürfnisse anpassen, ist es in der Variable *dropdown* des zurückgegebenen Elements wiederzufinden.

Das fertige Menü mit einem Dropdown ist in Abbildung 21 „Aufgabenmenü mit Dropdown“ zu sehen. Über dieses kann der Benutzer nun Änderungen am Präsentationsobjekt vornehmen.

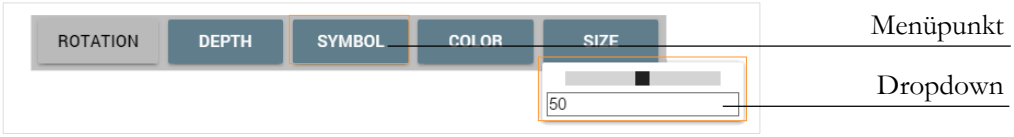

Abbildung 21 Aufgabenmenü mit Dropdown

Zuletzt beinhaltet das *taskType*-Objekt noch die beiden Methoden **onUpdate** und **onDelete**. Die erste Methode *onUpdate* wird immer dann aufgerufen, wenn die entsprechende Aufgabe in einen anderen Schritt verschoben wird. Dementsprechend erhält die *onUpdate*-Methode auch ein Funktionsattribut, welches eine Zahl ist, die die Schrittnummer widerspiegelt, in die die Aufgabe verschoben wurde. Im Gegensatz dazu erhält die *onDelete*-Methode kein Funktionsattribut. Sie wird aufgerufen, wenn eine Aufgabe gelöscht wird. Sie kann dazu genutzt werden, um z.B. für Three.js erstellte Objekte wieder zu löschen.

Um den Aufbau eines typischen *taskType*-Objekts und dessen Menü noch etwas zu verdeutlichen, wird nun exemplarisch eine Aufgabe programmiert, die ein Präsentationsobjekt zum Präsentationsbereich hinzufügt. Dieser Typ von Aufgabe wird im Folgenden als „Mesh-TaskType“ bezeichnet. Das folgende Beispiel ist ein Auszug aus dem Code, der ein Symbol in den Präsentationsbereich einfügt.

```
typeContainer["ov-add-symbol"] ={
    meta:{…},
    isAnimation: false,
    standartSize: 50,
    standartColor: 0,
    generateMesh: function(){
              …
       return mesh;
    },
    getTaskType: function(mesh, safe){
       …
       var symbolElement = createTaskMenuButton("Symbol",
htmlElement);

       var taskType = new TaskType();
       taskType.name = "Symbol";
       taskType.menu = new Menu3D(mesh);
       taskType.menu.symbol = {
              html: symbolElement
       };
       taskType.menu.color =
              new meshColorPicker(function(color) {…});
       taskType.menu.size =
              new sliderTextField("Size", function(p){…});
       return taskType;
    }
};
```

Die zwei Attribute *standartSize* und *standartColor* sind frei definierte Variablen. Sie werden vom Framework ignoriert und werden für die Umsetzung des Mesh-TaskTypes verwendet. Durch diese Variablen wird die Funktion bereitgestellt, dass Symbole die später hinzugefügt werden, automatisch die zuletzt benutze Farbe bzw. Größe haben. Dies soll es dem Benutzer vereinfachen, mehrere Symbole mit gleichem Aussehen zu erzeugen. Die eigentliche Generierung des Symbols geschieht in der *generateMesh*-Funktion. Das hier erzeugte Three.js-Mesh wird anschließend der *getTaskType*-Methode übergeben. Dieser Ablauf geschieht automatisch und wird vom Framework durchgeführt. In der *getTaskType*-Methode werden dann die eigentlichen Funktionen des Mesh-TaskTypes bestimmt. Diese Funktionen werden dann in das Menü eingebunden, welches sich in dem neu erzeugten TaskType-Objekt befindet.

Das Menü wird dazu aus vorgefertigten und eigenen Elementen zusammengestellt. Durch die Verwendung von *new Menu3D* werden automatisch einige Schaltflächen in das Menü eingefügt, welche Funktionen wie Rotation und Verschiebung auf der Tiefenachse bereitstellen. Damit der Benutzer die Größe und Farbe des Symbols ändern kann, werden auch wieder vordefinierte Elemente genutzt. So generiert beispielsweise die *meshColorPicker*-Methode automatisch die nötigen HTML-Elemente, die die Farbauswahl ermöglichen. Durch eine übergebene Funktion kann bestimmt werden, wie sich die Auswahl einer Farbe auf das Präsentationsobjekt auswirkt. Im Gegensatz dazu wird der Menüpunkt zur Symbolauswahl selbst erstellt. Dazu wird *createTaskMenuButton*-Methode verwendet und ihr ein zuvor definiertes HTML-Element übergeben.

Momentan muss der Benutzer noch direkten Zugang zu den JavaScript-Dateien besitzen, um eigene Aufgaben-Typen zu definieren. Eine Weiterentwicklung dieses Systems wäre es, eine Art Plug-In-System zu entwerfen, welches den direkten Upload von Erweiterungen ermöglicht. Alternativ wäre es auch denkbar, eine Sammlung von Aufgaben-Typen zu erstellen, aus denen sich der Benutzer die gewünschten Aufgaben-Typen aussuchen kann.

5.3 Animationen

Eine Animation ist ein spezieller Typ von Aufgabe, der besondere Eigenschaften hat und gesondert vom Framework bearbeitet wird. Ebenso wie der „Mesh-TaskType“ hat auch der „Animations-TaskType“ eine feste Objektstruktur, die es ermöglicht, eigene Animationen zu definieren und einzubinden. Ein „Animation-TaskType“ wird definiert, indem die *isAnimation*-Variable, welche sich im *typeContainer*-Objekt befindet, auf wahr gesetzt wird. Dies zeigt dem Framework, dass es sich bei dieser Aufgabe um eine Animation handelt. Dadurch wird dem *getTaskType*-Aufruf nur das *safe*-Objekt übergeben. Das mesh Objekt entfällt, da es an dieser Stelle nicht benötigt wird.

Wie auch beim „Mesh-TaskType“ wird dem Framework ein *taskType*-Objekt übergeben, sobald der Benutzer eine Animation zur Präsentation hinzufügt. Damit das Framework erkennt, dass es sich bei dem *taskType* -Objekt um eine Animation handelt, muss lediglich das *animation*-Attribut definiert werden. Der Wert dieses Attributes muss auch an dieser Stelle wieder ein Objekt sein, welches bestimmte Eigenschaften gesetzt haben muss. Um die Programmierung zu vereinfachen, kann auch hier wieder ein vorgefertigtes Objekt konstruiert werden, dessen Attribute anschließend angepasst werden können. Ein leeres Animationsobjekt kann mit dem Befehl *new TaskAnimation()* erzeugt werden und bringt die folgende Attribute mit sich: *duration*, *loop*, *easing*, *onInit*, *onStart*, *onLoop*, *onEnd* und *onEdit*. Die mit dem Präfix „on-“ beginnenden Variablen referenzieren dabei Methoden.

In der folgenden Übersicht sind die Attribute des Animationsobjekts noch einmal aufgeführt und kurz beschrieben. Auf die genaue Funktion und Verwendung der Methoden wird später noch präziser eingegangen.

Typ	Variable	Beschreibung
Zahl	duration	Dauer der Animation in Millisekunden.
Boolean	loop	Gibt an, ob sich die Animation wiederholt.
Methode	easing	Verwendete Easing-Funktion.
	onInit	Übergibt das zu animierende Präsentationsobjekt.
	onStart	Wird zum Start der Animation aufgerufen.
	onLoop	Wird zur Ausführung der Animation aufgerufen.
	onEnd	Wird zum Ende der Animation aufgerufen.
	onEdit	Aufgerufen, wenn das Präsentationsobjekt bearbeitet wird.

Der Benutzer kann bei der Erstellung eines „Animations-TaskTypes“ die Attribute des Animationsobjektes dann ersetzen. Der folgende Codeausschnitt verdeutlicht diesen Vorgang noch einmal:

```
var taskType = new TaskType();
taskType.animation = new TaskAnimation();

taskType.animation.onInit = function(mesh){...};
taskType.animation.onStart = function(){...};
taskType.animation.onLoop = function(progress){...};
taskType.animation.onEnd = function(){...};
taskType.animation.onEdit = function(){...};

return taskType;
```

Um das Zusammenspiel der einzelnen Methoden und Variablen etwas genauer zu verstehen, wird im Folgenden der Ablauf einer Animation abstrahiert und erklärt.

Jede Animation ist ein Ablauf, der aus den drei Phasen Beginn, Schleife und Ende besteht. Zu Beginn der Animation, also der ersten Phase, werden verschieden Werte ermittelt und gespeichert. Diese werden immer in einen Anfangs- und einen Endzustand unterteilt.

Beispielhaft kann dies an einer Bewegung von einer zu einer anderen Position erklärt werden. Auch hier werden zuerst Anfangs- und Endwert berechnet. Die berechneten Werte entsprechen in diesem Fall den Start- und Endpositionen der Bewegung und würden als Koordinaten gespeichert werden. Die Ausführung der eigentlichen Animation geschieht dann in der Schleife. Die Animation ist in dem Fall die Bewegung von Start- zum Endpunkt. Diese Bewegung wird mithilfe von Interpolation berechnet.

Interpolation bezeichnet den Vorgang, bei dem die Bewegung zwischen Start- und Endwert berechnet wird. Der Fortschritt der Animation wird dabei durch eine Dezimalzahl zwischen null und eins angeben, wobei null den Start und eins das Ende der Animation beschreibt. Diese Zahl kann auch als prozentualer Fortschritt der Animation angesehen werden, bei der der Maximalwert eins einhundert Prozent entspricht. Berechnet wird diese Zahl durch die Dauer der Animation und der bereits verstrichenen Zeit. Dabei wird die Formel

```
Fortschritt = verstrichene Zeit/Dauer der Animation
```

verwendet. Mithilfe der berechneten Dezimalzahl kann zwischen Start- und Endwert interpoliert werden. Die resultierende Position bildet sich aus der Addition des mit dem

Fortschrittswerts multiplizierten Endwerts und dem von eines subtrahierten Fortschrittswert multipliziert mit dem Startwert. Der Vorgang wird durch die folgende Formel beschrieben:

```
Position = Endposition*Fortschritt + Startposition*(1-Fortschritt)
```

Diese Formel besitzt den Vorteil, dass sie allgemeingültig für alle Animationen ist. Der Benutzer kann selbst entscheiden, wie sich Animationen entsprechend dem prozentualen Fortschritts verhalten und daraus eigene Vorgänge ableiten. Außerdem können Animationen vom Framework simpel manipuliert und angepasst werden. So können Animationen z.B. einfach rückwärts abgespielt werden, indem der prozentuale Fortschritt, anstatt von null zu eins, umgekehrt von eins zu null läuft. Auch die Geschwindigkeit der Animationen kann so von außen angepasst werden, indem z.B. die vergangene Zeit mit einem anderen Wert berechnet wird.

Dieses allgemeingültig beschriebene System von Animationen findet sich in der Umsetzung wieder. Die Eigenschaften der Animation werden dabei durch die Variablen *duration* und *loop* beschrieben. Die eigentliche Aufgabe wiederum wird durch die verschiedenen Methoden beschrieben. Die Dauer und Reihenfolge in denen diese Methoden bearbeitet werden, hängt dabei von den Attributen der Animation ab.

Nachdem diese Attribute der Animation bestimmt sind, können die eigentlichen Funktionen der Animation definiert werden. Der Aufbau dieser Funktionen orientiert sich an dem zuvor beschriebenen Ablauf, der allerdings um ein paar Zusatzschritte ergänzt wurde.

Die erste Funktion ist die *onInit*-Funktion. Sie wird zuerst vom Framework aufgerufen und erhält als Attribut ein Mesh-Objekt. Dieses gibt der Animation das zu animierende Präsentationsobjekt, welches in den nachfolgenden Funktionen manipuliert werden kann. Diese Manipulation beginnt mit der *onStart*-Methode. Sie repräsentiert den Beginn einer Animation und sollte den aktuellen Anfangszustand und gewünschten Endzustand ermitteln. Sie wird lediglich einmal zu Beginn der Animation aufgerufen und enthält keine Funktionsattribute. Ebenso verhält es sich auch bei der *onEnd*-Funktion, die zum Schluss einer Animation aufgerufen wird.

Der eigentliche Ablauf der Animation wird in der *onLoop*-Funktion beschrieben. Diese Funktion wird in einer Schleife aufgerufen, solange die zuvor definierte Dauer der Animation noch nicht abgelaufen ist. Anders als die *onStart*- oder *onEnd*-Funktionen, besitzt die *onLoop*-Funktion ein Methodenattribut, welches den Fortschritt der Animation beschreibt. Bei dem Attribut handelt es sich um eine Dezimalzahl zwischen null und eins,

die den aktuellen prozentualen Fortschritt repräsentiert. Im Abschnitt 5.4 „Easing von Animationen“ wir der Einfluss auf von Easing auf diese Dezimalzahl beschrieben.

Die letzte Methode ist die *onEdit*-Funktion. Sie wird immer dann aufgerufen, wenn der Benutzer das entsprechende Präsentationsobjekt bearbeitet, es z.B. im Präsentationsbereich verschiebt. So können Änderungen, die das Präsentationsobjekt betreffen, gespeichert werden.

Der Vorteil, der durch die Aufteilung in verschiedene Funktionen entsteht, ist, dass die Animation einfach manipuliert werden kann. Möchte man die Animation z.B. rückwärts ablaufen lassen, werden die Methoden einfach in der umgekehrten Reihenfolge aufgerufen, beginnend mit der *onEnd*-, dann *onLoop*- und zum Schluss *onStart*-Methode. In diesem Fall muss beachtet werden, dass die Fortschrittsvariable der *onLoop*-Funktion von eins zu null läuft. Diese Variante wird eingesetzt, wenn das Framework eine Animation rückwärts abspielt, da der Benutzer z.B. einen Schritt zurückgeht.

Nützlich ist dieses System außerdem, wenn der Benutzer zwischen Schritten der Präsentation springt oder eine laufende Animation abgebrochen werden muss. In diesem Fall kann direkt der Anfangs- oder Endzustand einer Animation gesetzt werden, indem die *onLoop*-Funktion mit dem Wert null oder eins, gefolgt von der *onStart* bzw. *onEnd*-Methode, aufgerufen wird.

Um den direkten Startzustand einer Animation zu setzten, wird also die Befehlsfolge

```
onLoop(0);
onStart();
```

aufgerufen und um den direkten Endzustand zu setzten, werden die Befehle

```
onLoop(1);
onEnd();
```

aufgerufen. Dieses System kommt z.B. zum Einsatz, wenn der Benutzer zwischen Schritten springt und alle vorherigen und folgenden Animationen korrekt gesetzt werden müssen.

Für die Umsetzung wurden verschiedene Abläufe getestet wie Präsentationsobjekte animiert werden können. In der derzeitigen Form, funktioniert das Hinzufügen einer Animation wie folgt:

Einfügen eines Präsentationsobjektes.

Einfügen der „Property Saver"-Animation.

Nachdem diese Aufgabe hinzugefügt wurde wird ein Dialog gezeigt, der Präsentationsobjekte auflistet. Hier kann der Benutzer das Präsentationsobjekt auswählen, welches die Animation erhalten soll.

Dieser Typ von Animation funktioniert wie ein Speicherpunkt, in dem der Zustand des Präsentationsobjektes gesichert wird.

Auswahl eines Schrittes, zu dessen Beginn die Animation gezeigt werden soll.

Dieser Schritt sollte sich hinter der zuvor hinzugefügten Animations-Aufgabe befinden.

Erneutes hinzufügen der „Property Saver"-Animation.

Diese Aufgabe bildet einen weiteren Speicherpunkt. Momentan werden hauptsächlich Transformationen des Präsentationsobjektes gespeichert, also Position, Rotation und Skalierung.

Anpassen des Präsentationsobjektes

Während noch der gleiche Schritt ausgewählt ist, kann das Präsentationsobjekt im Präsentationsbereich neu platziert werden.

Testen der Animation

Springt der Benutzer nun zwischen den Schritten hin und her, wird die Animation abgespielt. In der Zeitleiste befinden sich jetzt also minimal drei Aufgaben: Präsentationsobjekt und erste Animation in Schritt eins, zweite Animationsaufgabe in Schritt zwei.

Anpassen der Animation

Der Benutzer kann weitere Animationsschritte hinzufügen oder vorhandene anpassen, indem er die jeweilige Animations-Aufgabe in der Zeitleiste anklickt und anschließend das Präsentationsobjekt im Präsentationsbereich anpasst.

5.4 Easing von Animationen

Ein wichtiger Vorgang ist das Easing von Animationen. Es sorgt für ein optisch schöneres und spannenderes Ergebnis. Easing hat den Effekt, dass der Ablauf einer Animation nicht zu langweilig oder zu statisch wirkt. Erreicht wird dies, indem die Bewegungen nicht linear von Start- zum Endpunkt laufen, sondern sich am Beginn und Ende verlangsamen bzw. beschleunigen.

Technisch kann dieser Effekt erreicht werden, indem die dezimale Fortschrittsvariable, die der *onLoop*-Methode übergeben wird, nicht linear von null zu eins läuft. Dafür werden einige Formeln bereitgestellt, die für die Umrechnung zwischen einem linearem Ablauf und dem Ablauf mit Easing verantwortlich sind. Als Eingabe in die Funktion wird dafür jeweils einfach der Dezimalwert zwischen null und eins gewählt, welcher durch eine Formel umgerechnet wird und wieder als Wert zwischen null und eins ausgegeben wird.

Die Website easings.net[20] bietet eine Übersicht von verschiedenen Easing-Funktionen an und stellt diese als Diagramm dar. Die in „Abbildung 22 Darstellung von Easingfunktionen“ gezeigten Funktionen entsprechen denen, die auch zur Entwicklung des Programmes genutzt wurden. Die Diagramme zeigen den Verlauf einer Animation. Dabei spiegelt die x-Achse den zeitlichen Verlauf und die y-Achse den Wert zwischen null und eins wieder.

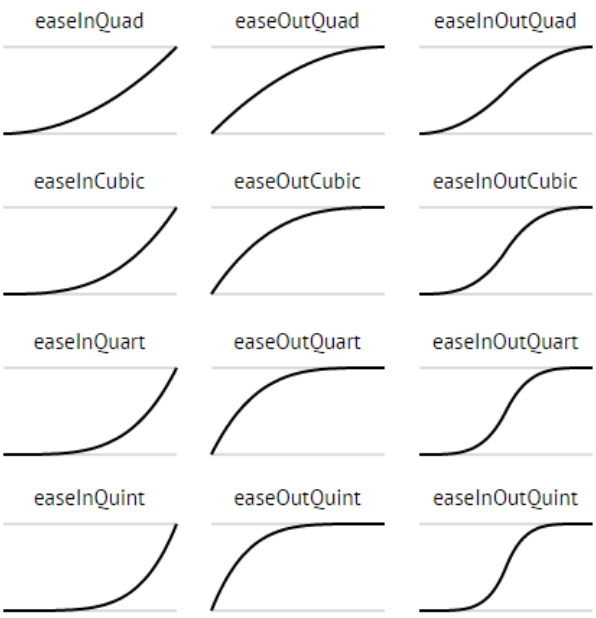

Abbildung 22 Darstellung von Easingfunktionen

[20] http://easings.net/ | 5.12.2015

5.5 TaskObjects

Im Framework wird jede Aufgabe durch ein JavaScript-Objekt repräsentiert. Wenn der Benutzer eine neue Aufgabe in die Zeitleiste hinzufügt, wird diese zunächst als leere Aufgabe eingefügt und erhält die Bezeichnung „Empty“. In diesem Zustand erfüllt die Aufgabe noch keinen Zweck und wird im Präsentationsablauf ignoriert.

Im Framework wird diese leere Aufgabe durch ein Aufgaben-Objekt repräsentiert: das *taskObject*. Dieses Objekt beinhaltet Informationen wie z.B. eine Identifikationsnummer, die Schrittnummer in der die Aufgabe platziert ist oder das HTML-Element, welches die Aufgabe repräsentiert. Außerdem beinhaltet es ein Attribut *taskType*, welches später den gewählten Typen der Aufgabe repräsentiert. Solange die Aufgabe noch leer ist, bleibt dieser Wert null.

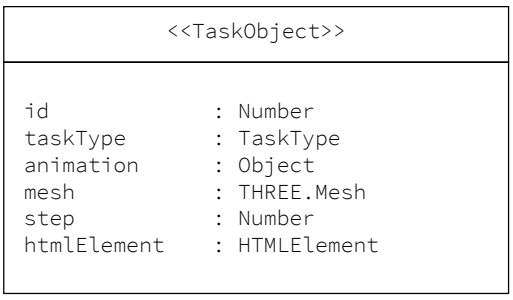

Abbildung 23 UML eines TaskObjects

Durch Klicken auf diese Aufgabe öffnet sich der Typauswahl-Dialog und der Benutzer kann auswählen, welchen Typen diese Aufgabe angehören soll. Dies hat zur Folge, dass das erstellte TaskObject vom Framework modifiziert wird. Zum einen wird das HTML-Element, welches die Aufgabe repräsentiert, angepasst und zum anderen wird das *taskType* Attribut gesetzt. Dabei handelt es sich um ein Objekt, das durch den Aufruf der *getTaskType*-Methode des gewählten Aufgaben-Typs erzeugt wurde.

Die Verwendung dieser Objektstruktur soll dafür sorgen, dass jede Aufgabe durch ein eigenes Objekt widergespiegelt wird, welches alle nötigen Informationen zur Aufgabe beinhaltet und global zu Verfügung stellt.

5.6 Speicherfunktion

Die Programmierung der Speicherfunktion ist in der JavaScript-Datei *saveMechanism.js* wiederzufinden. In ihr befinden sich drei Methoden, die für das Erstellen und Laden von gespeicherten Präsentationen zuständig sind. Die erste Methode *saveToFile* ist dafür zuständig, aus der derzeitigen Präsentation eine Datei zu erstellen und diese als Download anzubieten. Der grundsätzliche Ablauf ist so aufgebaut, dass ein JSON-String, der den Inhalt der Präsentation widerspiegelt, als Zeichenkette generiert wird. Diese Zeichenkette wird dem Benutzer als Textdatei in einem ZIP-Container zum Download angeboten.

Um den gewünschten JSON-String zu generieren, wird ein Array mit einer Schleife durchlaufen, welches alle sich in der Zeitleiste befindlichen Aufgaben beinhaltet. Diese werden dann einzeln abgearbeitet. Für jede Aufgabe wird ein neues Objekt *taskSave* erstellt, indem Attribute angelegt werden, die den Typen der Aufgabe widerspiegeln. Der folgende Code zeigt die verschiedenen Attribute, die dem *taskSave*-Objekt hinzugefügt werden:

```
var taskSave = {};
if(task.animation){
	taskSave.animation = {};
	taskSave.animation.duration = task.animation.duration;
	taskSave.animation.meshAddTaskID = task.mesh.addTask.id;
}
else if(task.mesh){
	taskSave.mesh = JSON.stringify(task.mesh.toJSON());
}
taskSave.id = task.id;
taskSave.step = task.step;
taskSave.taskType = {};
taskSave.taskType.typeID = task.taskType.typeID;
taskSave.taskType.name = task.taskType.name;
taskSave.safe = task.taskType.safe;
```

Bei der Erstellung dieses Objektes wird zwischen „Mesh-TaskTypes“ und „Animation-TaskTypes“ unterschieden. Während für ein „Mesh-TaskType“ das erstellte Three.js-Mesh-Objekt als JSON-String angehangen wird, wird für ein „Animation-TaskTypes“ ein Objekt mit Daten angehängt. Weitere Attribute des *taskSave*-Objekts sind z.B. die Aufgaben-ID, die Schrittnummer, in der sich die Aufgabe befindet, der Typ der Aufgabe oder der für den Benutzer sichtbare Name der Aufgabe.

Eine der größten Herausforderungen für das korrekte Speichern von Aufgaben war es, vom Benutzer gesetzte Eigenschaften einer Aufgabe wieder korrekt zu laden. Zu diesen Eigenschaften zählen unter anderem die Farbe, Größe und Position eines Präsentationsobjektes oder auch Werte einer Animations-Aufgabe. Damit diese Werte bei der Speicherung nicht verloren gehen, wird das das *safe*-Attribut des *taskType*-Objekts verwendet. Diese in diesem Objekt gespeicherten Werte werden während der Speicherung berücksichtigt und mit in das *taskSave*-Objekt gelegt. Dies ist der eigentliche Zweck, warum das *safe*-Objekt eingeführt wurde.

Sind alle nötigen Attribute im *taskSave*-Objekt angelegt, wird mithilfe der JSON.stringify-Methode aus dem *taskSave*-Objekt ein String generiert. Nachdem alle Aufgaben als String generiert wurden, werden diese in eine Textdatei geschrieben, die in einem ZIP-Container abgelegt wird. Dieser Vorgang wird mithilfe der Bibliothek JSZip umgesetzt.

Zum Schluss wird dem Benutzer der generierte ZIP-Container als Download angeboten. Dazu wird die Methode *saveAs* verwendet, welche von FileSaver.js bereitgestellt wird. In diesem Schritt wird auch die Dateiendung von *.zip* zu *.ppno* geändert. Dies dient lediglich der besseren Übersicht für den Benutzer, um gespeicherte Dateien schneller wiederzufinden und einfacher als Präsentation zu identifizieren.

Zum Laden einer zuvor gespeicherten Präsentation werden zwei Methoden benötigt: *loadFromFile* und *parseSave*. Die *loadFromSave*-Methode dient nur als Empfänger, die eine zu ladende Datei entgegennimmt und einliest. Sobald die Datei eingelesen ist, wird diese an die *parseSave*-Methode weitergereicht. Diese Methode ist für zwei Dinge zuständig: zum einen dem Laden und Parsen der Text Datei und zum anderen dem Erstellen der entsprechenden Aufgaben in der Zeitleiste.

Mithilfe der JSZip-Bibliothek wird die sich in dem ZIP-Container befindliche Textdatei als String geladen, welcher dann wiederrum mithilfe von JSON geparst wird. Dadurch erhält man ein Array von *taskSave*-Objekten, die zuvor beim Speichern erstellt wurden.

Als letzter Schritt wird aus jedem *taskSave*-Objekt wieder eine Aufgabe erstellt, die auf der Zeitleiste platziert wird. Hier wird auch wieder zwischen „Mesh-TaskTypes“ und „Animation-TaskTypes“ unterschieden. Der Ablauf zur Erstellung dieser Aufgaben ähnelt dabei dem, der angewandt wird wenn der Benutzer eine Aufgabe erzeugt. Der einzige signifikante Unterschied ist, dass der *getTaskType*-Methode das zuvor gesicherte *safe*-Objekt übergeben wird. Handelt es sich um ein „Mesh-TaskType“ wird außerdem das entsprechende Mesh-Objekt geparsed und übergeben. Der Aufruf der *generateMesh*-Methode wird also übersprungen.

6 ERWEITERBARKEIT

Für das Präsentationsprogramm ist eine Vielzahl von Erweiterungen denkbar. Diese können durch die Programmierung in JavaScript und das offen gehaltene Framework mit relativ wenig Aufwand umgesetzt werden. Zudem wurde das Framework an einigen Stellen so entwickelt, dass einige der benötigten Funktionen bereits in einer einfachen Form vorhanden sind. Diese müssten dann nur erweitert und verknüpft werden.

6.1 Weitere allgemeine Funktionen

Der derzeitige allgemeine Funktionsumfang des Präsentationsprogramms ist noch immer relativ gering. Es sind noch einige Erweiterungen denkbar, die aus anderen Programmen bekannt sind. So fehlt es in der erarbeiteten Version z.B. noch an einer Notizfunktion. Diese soll es dem Benutzer ermöglichen, Notizen für einzelne Schritte anzulegen, welche ihn dann während der Präsentation unterstützen können.

Eine weitere, vor allem für Einsteiger, nützliche Funktion sind vordefinierte Templates. Diese sollen es ermöglichen, schnell eine einfache optisch ansprechende Präsentation zu erstellen. Der Benutzer soll dazu lediglich seinen Inhalt zu Schritten hinzufügen, welche bereits formatiert sind. Denkbar sind auch vordefinierte Layouts, ähnlich wie sie bei PowerPoint gefunden werden können.

6.2 Multi-User und Second Screen

Eine wertvolle Erweiterung ist es, Präsentationen zu ermöglichen, die auf mehreren Geräten gleichzeitig angezeigt werden können. Der Gedanke dahinter ist, dass Zuschauer die Möglichkeit haben sollen, eine laufende Präsentation auf einem ihrer Geräte mitzuverfolgen. Denkbar ist dies auf jedem Gerät, welches einen Webbrowser anbietet. Über eine URL können dann alle Zuschauer die gezeigten Folien direkt auf ihrem Laptop, Smartphone oder ähnlichem mitverfolgen.

Abbildung 24 Second Screen fürs Fernsehen

Denkt man diese Funktion weiter, werden schnell viele nützliche Vorteile sichtbar. Zuschauer hätten die Möglichkeit, Schritte in der Präsentation zurück zu gehen, um sich Informationen noch einmal anzuschauen. Auch das direkte Abspeichern von Anmerkungen oder Notizen durch den Zuschauer ist denkbar.

Außerdem kann dieses System auch als eine einfache Plattform genutzt werden, mit der sich die Zuschauer die Präsentation noch einmal zu einem späteren Zeitpunkt anschauen könnte. So haben alle Zuschauer die Möglichkeit, eine gesehene Präsentation wiederholt zu betrachten.

Auch für den Präsentierenden kann dies von Vorteil sein, besonders wenn es zu mehreren regelmäßigen Präsentationen kommt. Dann könnte eine Onlinesammlung von Präsentationen erstellt werden, die einfach verwaltet werden kann. So können dann z.B. Zeiträume bestimmt werden, in denen sich Präsentation noch einmal anschauen lassen.

6.3 Interaktive Elemente

In seinen Grundzügen unterstützt das Präsentationsprogramm bereits interaktive Elemente, also Präsentationsobjekte, die während einer Präsentation noch modifiziert werden können. Momentan können Elemente allerdings nur verschoben oder rotiert werden. Dieses System kann noch erweitert werden, um das Programm auch als einen „Schaukasten“ zu nutzen. Das heißt, dass die Präsentation nicht aktiv vorgestellt wird, sondern z.B. auf Ausstellungen Besuchern zur Verfügung steht. Diese können dann selbständig durch die Präsentation navigieren. Dabei könnten sie dann bestimmte Themen auswählen, zu denen sie Informationen haben möchten oder eigenständig 3D-Modelle oder ähnliches betrachten.

In Kombination mit der zuvor vorgestellten Idee, die Präsentation auf mehreren Geräten anzuzeigen, ist es auch denkbar, dass Zuschauer über ihr Gerät direkt in eine Präsentation eingreifen und Einfluss nehmen können. Dies kann z.B. in Form einer Umfrage stattfinden. Denkbar wäre ein Szenario, in dem der Präsentierende eine Frage mit mehreren Antwortmöglichkeiten stellt und ein Zuschauer direkt über sein Gerät eine dieser Antworten auswählen kann. Anschließend wird in der Präsentation direkt das Ergebnis der Umfrage angezeigt und kann in den Ablauf der Präsentation mit eingebunden werden.

6.4 Gemeinsames Arbeiten

Eine etwas aufwendigere Funktion wäre das Erstellen einer Präsentation von mehreren Benutzern gleichzeitig. Der Gedanke dahinter ist, dass es einer Gruppe von Benutzern ermöglicht werden soll, gleichzeitig an einer Präsentation zu arbeiten. Dies ist ein Modell,

wie es von verschiedenen Onlinediensten wie z.B. Google Docs[21] bekannt ist. Dort können mehrere Personen zur gleichen Zeit ein Dokument oder auch eine Folienpräsentation erstellen und bearbeiten.

Im Zusammenhang mit dem hier erstellten Präsentationsprogramm bietet sich dies an, da es sich gut eignet, um eine klare Aufgabenteilung zu schaffen. So können die Aufgaben klar in Inhalt, Gestaltung, Animation und Programmierung aufgeteilt und verschiedenen Personen zugeordnet werden.

[21] https://www.google.de/intl/de/docs/about/ | 5.12.2015

7 Fazit

Das Ziel dieser Arbeit war es, ein Programm zu entwickeln, mit dem zeitgemäße Präsentationen erstellt werden können. Dazu wurde das Bedienparadigma bestehender Präsentationsprogramme näher betrachtet und in ausgewählten Punkten untersucht. Auf Grundlage dieser Erkenntnisse wurde das Konzept für ein neues Programm erstellt und umgesetzt.

Das erdachte Konzept baute dabei auf alte Eigenschaften von Präsentationsprogrammen auf und modifizierte diese so, dass sie mit einigen neuen Ideen zusammenarbeiten. Außerdem wurden Strukturen übernommen, die aus anderen Kategorien von Software bekannt sind, wie dem Filmschnitt oder der Webtechnik. So basiert z.B. die grundsätzliche Idee, Präsentationen in einer Zeitleiste anzuordnen, auf bekannten Filmschnittprogrammen.

Die Umsetzung des zu erstellenden Programms teilte sich grob in zwei Bereiche auf: das Framework und die Benutzeroberfläche. Die Gestaltung einer einfach zu bedienenden Benutzeroberfläche sollte dafür sorgen, dass auch Benutzer ohne Programmierkenntnisse Präsentationen mit diesem Programm erstellen können. Durch die frühzeitige Entwicklung dieser Oberfläche konnte sich auch die Entwicklung des Frameworks entsprechend anpassen, wodurch einige Aspekte der Entwicklung einfacher wurden.

Die für die Umsetzung verwendete Programmiersprache JavaScript in Kombination mit HTML und CSS bot sich besonders an, um schnell eine gute Bedienoberfläche zu konstruieren, die sich dynamisch anpassen kann. Dies führt auch dazu, dass das Programm durch Verwendung dieser Techniken, plattform- und geräteübergreifend funktioniert und in Zukunft einfach gewartet und erweitert werden kann.

Das erhaltene Programm bietet sich an, um Präsentationen zu erstellen, die Objekte und Animationen dreidimensional darstellen können. Außerdem werden nötige Funktionen geliefert, die es auch Benutzern ohne Programmierfähigkeiten ermöglichen, simple und moderne Präsentationen zu erstellen. Andererseits können Benutzer eigenen Programmcode erstellen und dadurch das Programm beliebig um Funktionen erweitern. Dies ist aufgrund des Frameworks möglich, welches so programmiert wurde, dass es Elemente, die auf der Präsentation angezeigt werden, in Module unterteilt. Durch diese allgemein gehaltene Programmierung ist das System einfach erweiterbar, ohne dass Elemente in ihrer Komplexität eingeschränkt werden.

Außerdem wurde ein simples System zum Erstellen von Animationen entwickelt. Dieses ermöglicht es dem Benutzer in jedem Schritt der Präsentation die Anordnung und Eigenschaften von Objekten zu ändern. Der eigentliche Verlauf der Animationen wird vom Framework berechnet.

Der als Teil der Bachelorarbeit umgesetzte Programmcode bildet dabei eine Grundlage, die darauf ausgelegt ist, erweitert zu werden. Dies gilt nicht nur für den internen Ablauf des Programms, sondern auch für seine Benutzeroberfläche. Sie ist ein wichtiger Bestandteil des Projekts und wurde schon früh in die Entwicklung eingebunden.

All diese Eigenschaften sollen eine offene Plattform bilden, die sich auch in Zukunft noch an die Bedürfnisse des Benutzers anpassen kann. Dem Benutzer sind dadurch keine Grenzen gesetzt, eine moderne Präsentation zu gestalten.

ABBILDUNGSVERZEICHNIS

Abbildung 1 Tageslichtprojektor 2
Abbildung 2 Logo der Software Prezi 3
Abbildung 3 PowerPoint - Auswahl an Geometrien 4
Abbildung 4 PowerPoint - Layout Auswahlmenü 4
Abbildung 5 PowerPoint - Verschiedene Animationstypen 5
Abbildung 6 PowerPoint - Unterteilung des Fensters 6
Abbildung 7 Microsoft Ribbon 6
Abbildung 8 PowerPoint - Ribbon-Menüpunkte 7
Abbildung 9 Aufbau des Zeitstrahls 8
Abbildung 10 Anordnung der Elemente auf der Tiefenachse 9
Abbildung 11 Benutzeroberfläche 16
Abbildung 12 Funktion des Hauptmenüs 17
Abbildung 13 Aufbau der Zeitleiste 18
Abbildung 14 Darstellung des Aufgabenmenüs 19
Abbildung 15 Präsentationsobjekt-Auswahldialog 21
Abbildung 16 Auswahl an OpenSource-Schriftarten 21
Abbildung 17 Auswahl an Symbolen der FontAwesome Bibliothek 22
Abbildung 18 Overlaymenü - Kategorien 26
Abbildung 19 Zeitleiste mit zwei Schritten 27
Abbildung 20 UML eines TaskType-Objekts 30
Abbildung 21 Aufgabenmenü mit Dropdown 33
Abbildung 22 Darstellung von Easingfunktionen 41
Abbildung 23 UML eines TaskObjects 42
Abbildung 24 Second Screen fürs Fernsehen 45

Abbildung 1 Tageslichtprojektor
https://commons.wikimedia.org/wiki/File:OverheadReise1.JPG
By Bomas13 (Own work) [CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

Abbildung 2 Logo der Software Prezi
https://commons.wikimedia.org/wiki/File%3APrezi_logo_transparent_2012.svg
Prezi Inc. [CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

Abbildung 22 Darstellung von Easingfunktionen
Entnommen und modifiziert von http://easings.net/
OpenSource Website – Veröffentlicht unter der „GNU Version 3"-Lizenz (https://github.com/ai/easings.net)

Abbildung 24 Second Screen fürs Fernsehen
https://commons.wikimedia.org/wiki/File%3ASecondscreen_wettendass_Ipad-app.jpg
By pr_ip Primus Inter Pares [CC BY-SA 2.0 (http://creativecommons.org/licenses/by-sa/2.0)], via Wikimedia Commons

LITERATURVERZEICHNIS

Carota. (05. 12 2015). Von GitHub: https://github.com/danielearwicker/carota abgerufen

Easing-Funktionen. (5. 12 2015). Von GitHub Gist: https://gist.github.com/gre/1650294 abgerufen

Easings Cheat Sheet. (5. 12 2015). Von http://easings.net/ abgerufen

facetype.js. (5. 12 2015). Von GitHub: https://github.com/gero3/facetype.js abgerufen

FileSaver.js. (5. 11 2015). Von GitHub: https://github.com/eligrey/FileSaver.js abgerufen

Font Awesome. (5. 12 2015). Von https://fortawesome.github.io/Font-Awesome/ abgerufen

JsZip. (5. 12 2015). Von GitHub: https://github.com/Stuk/jszip abgerufen

Keynote für Mac. (5. 12 2012). Von https://www.apple.com/de/mac/keynote/ abgerufen

LibreOffice. (5. 12 2015). Von https://www.libreoffice.org abgerufen

Material Design Lite. (5. 12 2015). Von http://www.getmdl.io abgerufen

MathBox.js. (4. 12 2015). Von GitHub: https://github.com/unconed/MathBox.js/ abgerufen

Microsoft PowerPoint. (5. 12 2015). Von https://products.office.com/de-de/powerpoint abgerufen

Online-Präsentationstool Prezi. (5. 12 2015). Von https://prezi.com abgerufen

Ribbon. (20. 7 2015). Von Wikipedia: https://de.wikipedia.org/wiki/Ribbon abgerufen

Seifert, J. W. (1997). *Visualisieren, Präsentieren, Moderieren*. GABAL.

Spectrum. (5. 12 2015). Von GitHub: https://github.com/bgrins/spectrum abgerufen

Three.js. (5. 12 2015). Von GitHub: https://github.com/mrdoob/three.js/ abgerufen

Wittens, S. (5. 8 2013). *Making WebGL Dance*. Von https://www.youtube.com/watch?v=GNO_CYUjMK8 abgerufen

Wittens, S. (5. 12 2015). *Hackery, Math & Design*. Von http://acko.net abgerufen

VERSICHERUNG ÜBER SELBSTSTÄNDIGKEIT

Hiermit versichere ich, dass ich die vorliegende Arbeit „Moderne Präsentationen: Erarbeitung eines Präsentationsprogramms unter Verwendung aktueller Browser-Technologien“ ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Alle Passagen, die ich wörtlich aus der Literatur oder aus anderen Quellen wie z.B. Internetseiten übernommen habe, habe ich deutlich als Zitat mit Angabe der Quelle kenntlich gemacht.

Hamburg, den _____ _____